# INTEGRAL UNIVERSITY, LUCKNOW

## DIRECTORATE OF DISTANCE EDUCATION

MCA-205 Paper Code :PIJ/M

# PROGRAMMING IN JAVA

# CONTENTS

# UNIT -1 ABOUT LANGUAGE

**Contents**

- Language
  - Machine language
  - Assembly language
  - High language
- Java as a programming plateform
- Features of java
- History of java
- Common misconcepts
- Review & self assessment Question
- Further Readings

## Machine Language

A computer's native language, which differs among different types of computers, is its machine language—a set of built-in primitive instructions. These instructions are in the form of binary code, so if you want to give a computer an instruction in its native language, you have to enter the instruction as binary code. For example, to add two numbers, you might have to write an instruction in binary code, like this:

**1101101010011010**

## Assembly Language

Programming in machine language is a tedious process. Moreover, programs written in machine language are very difficult to read and modify. For this reason, assembly language was created in the early days of computing as an alternative to machine languages. Assembly language uses a short descriptive word, known as a mnemonic, to represent each of the machine-language instructions. For example, the mnemonic **add** typically means to add numbers and **sub** means to subtract numbers. To add the numbers **2** and **3** and get the result, you might write an instruction in assembly code like this:

add 2, 3, result

Assembly languages were developed to make programming easier. However, because the computer cannot execute assembly language, another program—called an assembler—is used to translate assembly-language programs into machine code

## High-Level Language

In the 1950s, a new generation of programming languages known as high-level languages

emerged. They are platform independent, which means that you can write a program in a highlevel language and run it in different types of

machines. High-level languages are English-like and easy to learn and use. The instructions in a high-level programming language are called statements. Here, for example, is a high-level language statement that computes the area of a circle with a radius of **5**:

area = 5 * 5 * 3.14159;

There are many high-level programming languages, and each was designed for a specific purpose.

A program written in a high-level language is called a source program or source code. Because a computer cannot execute a source program, a source program must be translated into machine code for execution. The translation can be done using another programming tool called an interpreter or a compiler.

An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away.

The World Wide Web is an electronic information repository that can be accessed on the Internet from anywhere in the world. The Internet, the Web's infrastructure, has been around for more than forty years. The colorful World Wide Web and sophisticated Web browsers are the major reason for the Internet's popularity.

Java initially became attractive because Java programs can be run from a Web browser. Such programs are called applets. Applets employ a modern graphical interface with buttons, text fields, text areas, radio buttons, and so on, to interact with users on the Web and process their requests. Applets make the Web responsive, interactive, and fun to use. Applets are embedded in an HTML file. HTML (Hypertext Markup Language) is a simple scripting language for laying out documents, linking documents on the Internet, and bringing images, sound, and video alive on the Web. Today, you can use Java to develop rich Internet applications.

A rich Internet application (RIA) is a Web application designed to deliver the same features and functions normally associated with desktop applications.

Java is now very popular for developing applications on Web servers. These applications process data, perform computations, and generate dynamic Web pages. Many commercial Websites are developed using Java on the backend.

Java is a versatile programming language: you can use it to develop applications for desktop computers, servers, and small handheld devices. The software for Android cell phones is developed using Java.

## Java as a Programming Platform

"As a computer language, Java's hype is overdone: Java is certainly a good programming language. There is no doubt that it is one of the better languages available to serious programmers. We think it could potentially have been a great programming language, but it is probably too late for

that. Once a language is out in the field, the ugly reality of compatibility with existing code sets in."

Java has a lot of nice language features—It has its share of warts, and newer additions to the language are not as elegant as the original ones because of the ugly reality of compatibility.

There are lots of programming languages out there, and few of them make much of a splash. Java is a whole platform, with a huge library, containing lots of reusable code, and an execution environment that provides services such as security, portability across operating systems, and automatic garbage collection.

As a programmer, you will want a language with a pleasant syntax and comprehensible semantics (i.e., not C++). Java fits the bill, as do dozens of other fine languages. Some languages give you portability, garbage collection, and the like, but they don't have much of a library, forcing you to roll your own if you want fancy graphics or networking or database access.

Well, Java has everything—a good language, a high-quality execution environment, and a vast library. That combination is what makes Java an irresistible proposition to so many programmers.

## Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

- Simple
- Object-Oriented
- Network Savy
- Secure
- Architecture neutral
- Portable
- Interpreted
- High Performance
- Multithreaded
- Dynamic

## Simple

We wanted to build a system that could be programmed easily without a lot of esoteric training and which leveraged today's standard practice. So even though we found that C++ was unsuitable, we designed Java as

closely to C++ as possible in order to make the system more comprehensible. Java omits many rarely used, poorly understood, confusing features of C++ that, in our experience, bring more grief than benefit.

The syntax for Java is, indeed, a cleaned-up version of the syntax for C++. There is no need for header files, pointer arithmetic (or even a pointer syntax), structures, unions, operator overloading, virtual base classes, and so on. (See the C++ notes interspersed throughout the text for more on the differences between Java and C++.) The designers did not, however, attempt to fix all of the clumsy features of C++. For example, the syntax of the switch statement is unchanged in Java. If you know C++, you will find the transition to the Java syntax easy.

If you are used to a visual programming environment (such as Visual Basic), you will not find Java simple. There is much strange syntax (though it does not take long to get the hang of it). More important, you must do a lot more programming in Java. The beauty of Visual Basic is that its visual design environment almost automatically provides a lot of the infrastructure for an application. The equivalent functionality must be programmed manually, usually with a fair bit of code, in Java. There are, however, third-party development environments that provide “drag-and-drop”-style program development.

Another aspect of being simple is being small. One of the goals of Java is to enable the construction of software that can run stand-alone in small machines. The size of the basic interpreter and class support is about 40K bytes; adding the basic standard libraries and thread support (essentially a self-contained microkernel) adds an additional 175K.

This was a great achievement at the time. Of course, the library has since grown to huge proportions. There is now a separate Java Micro Edition with a smaller library, suitable for embedded devices.

## Object Oriented

Simply stated, object-oriented design is a technique for programming that focuses on the data (= objects) and on the interfaces to that object. To make an analogy with carpentry, an “object-oriented” carpenter would be mostly concerned with the chair he was building, and secondarily with the tools used to make it; a “non-object oriented” carpenter would think primarily of his tools. The object-oriented facilities of Java are essentially those of C++. Object orientation has proven its worth in the last 30 years, and it is inconceivable that a modern programming language would not use it. Indeed, the object-oriented features of Java are comparable to those

of C++. The major difference between Java and C++ lies in multiple inheritance, which Java has replaced with the simpler concept of interfaces, and in the Java meta class model.

## Network-Savvy

Java has an extensive library of routines for coping with TCP/IP protocols like HTTP and FTP. Java applications can open and access objects across the Net via URLs with the same ease as when accessing a local file system.

We have found the networking capabilities of Java to be both strong and easy to use. Anyone who has tried to do Internet programming using another language will revel in how simple Java makes onerous tasks like opening a socket connection. The remote method invocation mechanism enables communication between distributed objects.

## Robust

Java is intended for writing programs that must be reliable in a variety of ways. Java puts a lot of emphasis on early checking for possible problems, later dynamic (runtime) checking, and eliminating situations that are error-prone. . . . The single biggest difference between Java and C/C++ is that Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data.

This feature is also very useful. The Java compiler detects many problems that, in other languages, would show up only at runtime. As for the second point, anyone who has spent hours chasing memory corruption caused by a pointer bug will be very happy with this feature of Java.

If you are coming from a language like Visual Basic that doesn't explicitly use pointers, you are probably wondering why this is so important. C programmers are not so lucky.

They need pointers to access strings, arrays, objects, and even files. In Visual Basic, you do not use pointers for any of these entities, nor do you need to worry about memory allocation for them. On the other hand, many data structures are difficult to implement in a pointerless language. Java gives you the best of both worlds. You do not need pointers for everyday constructs like strings and arrays. You have the power of pointers if you need it, for example, for linked lists. And you always have complete safety, because you can never access a bad pointer, make memory allocation errors, or have to protect against memory leaking away.

## Secure

Java is intended to be used in networked/distributed environments. Toward that end, a lot of emphasis has been placed on security. Java enables the construction of virus-free, tamper-free systems.

In the first edition of Core Java we said: "Well, one should 'never say never again,'" and we turned out to be right. Not long after the first version of the Java Development Kit was shipped, a group of security experts at Princeton University found subtle bugs in the security features of Java 1.0. Sun Microsystems has encouraged research into Java security, making publicly available the specification and implementation of the virtual machine and the security libraries. They have fixed all known security bugs quickly.

In any case, Java makes it extremely difficult to outwit its security mechanisms. The bugs found so far have been very technical and few in number.

From the beginning, Java was designed to make certain kinds of attacks impossible, among them:

- **Overrunning the runtime stack—a common attack of worms and viruses**
- **Corrupting memory outside its own process space**
- **Reading or writing files without permission**

A number of security features have been added to Java over time. Since version 1.1, Java has the notion of digitally signed classes. With a signed class, you can be sure who wrote it. Any time you trust the author of the class, the class can be allowed more privileges on your machine.

## Architecture Neutral

The compiler generates an architecture-neutral object file format—the compiled code is executable on many processors, given the presence of the Java runtime system.

The Java compiler does this by generating byte code instructions which have nothing to do with a particular computer architecture. Rather, they are designed to be both easy to interpret on any machine and easily translated into native machine code on the fly.

This is not a new idea. More than 30 years ago, both Niklaus Wirth's original implementation of Pascal and the UCSD Pascal system used the same technique.

Of course, interpreting byte codes is necessarily slower than running machine instructions at full speed, so it isn't clear that this is even a good

idea. However, virtual machines have the option of translating the most frequently executed byte code sequences into machine code, a process called just-in-time compilation. This strategy has proven so effective that even Microsoft's .NET platform relies on a virtual machine.

The virtual machine has other advantages. It increases security because the virtual machine can check the behavior of instruction sequences. Some programs even produce byte codes on the fly, dynamically enhancing the capabilities of a running program.

## Portable

Unlike C and C++, there are no "implementation-dependent" aspects of the specification. The sizes of the primitive data types are specified, as is the behavior of arithmetic on them.

For example, an int in Java is always a 32-bit integer. In C/C++, int can mean a 16-bit integer, a 32-bit integer, or any other size that the compiler vendor likes. The only restriction is that the int type must have at least as many bytes as a short int and cannot have more bytes than a long int. Having a fixed size for number types eliminates a major porting headache. Binary data is stored and transmitted in a fixed format, eliminating confusion about byte ordering. Strings are saved in a standard Unicode format.

The libraries that are a part of the system define portable interfaces. For example, there is an abstract Window class and implementations of it for UNIX, Windows, and the Macintosh.

As anyone who has ever tried knows, it is an effort of heroic proportions to write a program that looks good on Windows, the Macintosh, and ten flavors of UNIX. Java 1.0 made the heroic effort, delivering a simple toolkit that mapped common user interface elements to a number of platforms. Unfortunately, the result was a library that, with a lot of work, could give barely acceptable results on different systems. (And there were often different bugs on the different platform graphics implementations.) But it was a start. There are many applications in which portability is more important than user interface slickness, and these applications did benefit from early versions of Java. By

Now, the user interface toolkit has been completely rewritten so that it no longer relies on the host user interface. The result is far more consistent and, we think, more attractive than in earlier versions of Java.

## Interpreted

The Java interpreter can execute Java byte codes directly on any machine to which the interpreter has been ported. Since linking is a more

incremental and lightweight process, the development process can be much more rapid and exploratory.

Incremental linking has advantages, but its benefit for the development process is clearly overstated. Early Java development tools were, in fact, quite slow. Today, the byte codes are translated into machine code by the just-in-time compiler.

## High Performance

While the performance of interpreted byte codes is usually more than adequate, there are situations where higher performance is required. The byte codes can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on.

In the early years of Java, many users disagreed with the statement that the performance was "more than adequate." Today, however, the just-in-time compilers have become so good that they are competitive with traditional compilers and, in some cases, even outperform them because they have more information available. For example, a just-in-time compiler can monitor which code is executed frequently and optimize just that code for speed. A more sophisticated optimization is the elimination (or "inlining") of function calls. The just-in-time compiler knows which classes have been loaded. It can use inlining when, based upon the currently loaded collection of classes, a particular function is never overridden, and it can undo that optimization later if necessary.

## Multithreaded

The benefits of multithreading are better interactive responsiveness and real-time behavior. If you have ever tried to do multithreading in another language, you will be pleasantly surprised at how easy it is in Java. Threads in Java also can take advantage of multiprocessor systems if the base operating system does so. On the downside, thread implementations on the major platforms differ widely, and Java makes no effort to be platform independent in this regard. Only the code for calling multithreading remains the same across machines; Java offloads the implementation of multithreading to the underlying operating system or a thread library.

Nonetheless, the ease of multithreading is one of the main reasons why Java is such an appealing language for server-side development.

## Dynamic

In a number of ways, Java is a more dynamic language than C or C++. It was designed to adapt to an evolving environment. Libraries can freely add

new methods and instance variables without any effect on their clients. In Java, finding out runtime type information is straightforward.

This is an important feature in those situations in which code needs to be added to a running program. A prime example is code that is downloaded from the Internet to run in a browser. In Java 1.0, finding out runtime type information was anything but straightforward, but current versions of Java give the programmer full insight into both the structure and behavior of its objects. This is extremely useful for systems that need to analyze objects at runtime, such as Java GUI builders, smart debuggers, pluggable components, and object databases.

Java Applets and the Internet

The idea here is simple: Users will download Java byte codes from the Internet and run them on their own machines. Java programs that work on web pages are called applets. To use an applet, you only need a Java-enabled web browser, which will execute the byte codes for you. You need not install any software. Because Sun licenses the Java source code and insists that there be no changes in the language and standard library, a Java applet should run on any browser that is advertised as Java-enabled. You get the latest version of the program whenever you visit the web page containing the applet.

Most important, thanks to the security of the virtual machine, you need never worry about attacks from hostile code.

When the user downloads an applet, it works much like embedding an image in a web page. The applet becomes a part of the page, and the text flows around the space used for the applet. The point is, the image is alive. It reacts to user commands, changes its appearance, and sends data between the computer presenting the applet and the computer serving it.

## History of Java

Java history is interesting to know. The history of java starts from Green Team. Java team members (also known as Green Team), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

1) James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.

2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called "Greentalk" by James Gosling and file extension was .gt.

4) After that, it was called Oak and was developed as a part of the Green project.

Why Oak name for java language?

5) Why Oak? Oak is a symbol of strength and choosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.

6) In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.

## Why Java name for java language:

7) Why they choosed java name for java language? The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling "Java was one of the top choices along with Silk". Since java was so unique, most of the team members preferred java.

8) Java is an island of Indonesia where first coffee was produced (called java coffee).

9) Notice that Java is just a name not an acronym.

10) Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called Java one of the Ten Best Products of 1995.

12) JDK 1.0 released in(January 23, 1996).

## Java Version History

There are many java versions that has been released. Current stable release of Java is Java SE 8.

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan, 1996)
- JDK 1.1 (19th Feb, 1997)
- J2SE 1.2 (8th Dec, 1998)
- J2SE 1.3 (8th May, 2000)

- J2SE 1.4 (6th Feb, 2002)
- J2SE 5.0 (30th Sep, 2004)
- Java SE 6 (11th Dec, 2006)
- Java SE 7 (28th July, 2011)
- Java SE 8 (18th March, 2014)

## Common Misconceptions about Java

We close this chapter with a list of some common misconceptions about Java, along with commentary.

**Java is an extension of HTML.**

Java is a programming language; HTML is a way to describe the structure of a web page. They have nothing in common except that there are HTML extensions for placing Java applets on a web page.

**I use XML, so I don't need Java.**

Java is a programming language; XML is a way to describe data. You can process XML data with any programming language, but the Java API contains excellent support for XML processing. In addition, many important third-party XML tools are implemented in Java.

### Java is an easy programming language to learn

No programming language as powerful as Java is easy. You always have to distinguish between how easy it is to write toy programs and how hard it is to do serious work.

The Java libraries contain thousands of classes and interfaces, and tens of thousands of functions. Luckily, you do not need to know every one of them, but you do need to know surprisingly many to use Java for anything realistic.

### Java will become a universal programming language for all platforms

This is possible, in theory, and it is certainly the case that every vendor but Microsoft seems to want this to happen. However, many applications, already working perfectly well on desktops, would not work well on other devices or inside a browser. Also, these applications have been written to take advantage of the speed of the processor and the native user interface library and have been ported to all the important platforms anyway.

Among these kinds of applications are word processors, photo editors, and web browsers. They are typically written in C or C++, and we see no benefit to the end user in rewriting them in Java.

**Java is just another programming language**

Java is a nice programming language; most programmers prefer it over C, C++, or C#. But there have been hundreds of nice programming languages that never gained widespread popularity, whereas languages with obvious flaws, such as C++ and Visual Basic, have been wildly successful.

Why? The success of a programming language is determined far more by the utility of the support system surrounding it than by the elegance of its syntax. Are there useful, convenient, and standard libraries for the features that you need to implement? Are there tool vendors that build great programming and debugging environments? Does the language and the toolset integrate with the rest of the computing infrastructure?

Java is successful because its class libraries let you easily do things that were hard before, such as networking and multithreading. The fact that Java reduces pointer errors is a bonus and so programmers seem to be more productive with Java, but these factors are not the source of its success.

**Now that C# is available, Java is obsolete.**

C# took many good ideas from Java, such as a clean programming language, a virtual machine, and garbage collection. But for whatever reasons, C# also left some good stuff behind, in particular security and platform independence. If you are tied to Windows, C# makes a lot of sense. But judging by the job ads, Java is still the language of choice for a majority of developers.

**Java is proprietary, and it should therefore be avoided.**

Sun Microsystems licenses Java to distributors and end users. Although Sun has ultimate control over Java through the "Java Community Process," they have involved many other companies in the development of language revisions and the design of new libraries. Source code for the virtual machine and the libraries has always been freely available, but only for inspection, not for modification and redistribution. Up to this point, Java has been "closed source, but playing nice."

This situation changed dramatically in 2007, when Sun announced that future versions of Java will be available under the General Public License, the same open source license that is used by Linux. It remains to be seen how Sun will manage the governance of Java in the future, but there is no question that the open sourcing of Java has been a very courageous move that will extend the life of Java by many years.

**Java is interpreted, so it is too slow for serious applications.**

In the early days of Java, the language was interpreted. Nowadays, except on "micro" platforms such as cell phones, the Java virtual machine uses a just-in-time compiler. The "hot spots" of your code will run just as fast in Java as they would in C++, and in some cases, they will run faster.

Java does have some additional overhead over C++. Virtual machine startup time is slow, and Java GUIs are slower than their native counterparts because they are painted in a platform-independent manner.

People have complained for years that Java applications are too slow. However, today's computers are much faster than they were when these complaints started. A slow Java program will still run quite a bit better than those blazingly fast C++ programs did a few years ago. At this point, these complaints sound like sour grapes, and some detractors have instead started to complain that Java user interfaces are ugly rather than slow.

**All Java programs run inside a web page**

All Java applets run inside a web browser. That is the definition of an applet—a Java program running inside a browser. But most Java programs are stand-alone applications that run outside of a web browser. In fact, many Java programs run on web servers and produce the code for web pages. Most of the programs in this book are stand-alone programs. Sure, applets can be fun. But stand-alone Java programs are more important and more useful in practice.

**Java programs are a major security risk**

In the early days of Java, there were some well-publicized reports of failures in the Java security system. Most failures were in the implementation of Java in a specific browser.

Researchers viewed it as a challenge to try to find chinks in the Java armor and to defy the strength and sophistication of the applet security model. The technical failures that they found have all been quickly corrected, and to our knowledge, no actual systems were ever compromised. To keep this in perspective, consider the literally millions of virus attacks in Windows executable files and Word macros that cause real grief but surprisingly little criticism of the weaknesses of the attacked platform. Also, the ActiveX mechanism in Internet Explorer would be a fertile ground for abuse, but it is so boringly obvious how to circumvent it that few researchers have bothered to publicize their findings.

Some system administrators have even deactivated Java in company browsers, while continuing to permit their users to download executable files, ActiveX controls, and Word documents. That is pretty ridiculous—

currently, the risk of being attacked by hostile Java applets is perhaps comparable to the risk of dying from a plane crash; the risk of being infected by opening Word documents is comparable to the risk of dying while crossing a busy freeway on foot.

## JavaScript is a simpler version of Java

JavaScript, a scripting language that can be used inside web pages, was invented by Netscape and originally called LiveScript. JavaScript has a syntax that is reminiscent of Java, but otherwise there are no relationships (except for the name, of course). A subset of JavaScript is standardized as ECMA-262. JavaScript is more tightly integrated with browsers than Java applets are. In particular, a JavaScript program can modify the document that is being displayed, whereas an applet can only control the appearance of a limited area.

**With Java, I can replace my computer with a $500 "Internet appliance."**

When Java was first released, some people bet big that this was going to happen.we have believed it is absurd to think that home users are going to give up a powerful and convenient desktop for a limited machine with no local storage. We found the Java-powered network computer a plausible option for a "zero administration initiative" to cut the costs of computer ownership in a business, but even that has not happened in a big way.

On the other hand, Java has become widely distributed on cell phones. We must confess that we haven't yet seen a must-have Java application running on cell phones, but the usual fare of games and screen savers seems to be selling well in many markets

## Review & Self Assessment Questio

1. What is language?
2. What is World Wide Web?
3. Java is a programming Platform or not Define it.
4. How you can say that java is architecture Neutral?
5. What are the features of java?
6. Define the term dynamic.
7. What are the common miss-concepts about java.

## Further Readings

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

# UNIT-2 INTRODUCRION TO JAVA

## Contents

- Java
- Platform
- Java is platform independent
- Types of java program
- Java architecture
- Java Virtual machine
- Java Runtime Environment
- Java Standard Library
- Review & self assessment Question
- Further Readings

## Introduction to JAVA

Java is a programming language and a platform.Java is a high level, robust, secured and object-oriented programming language.

**Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

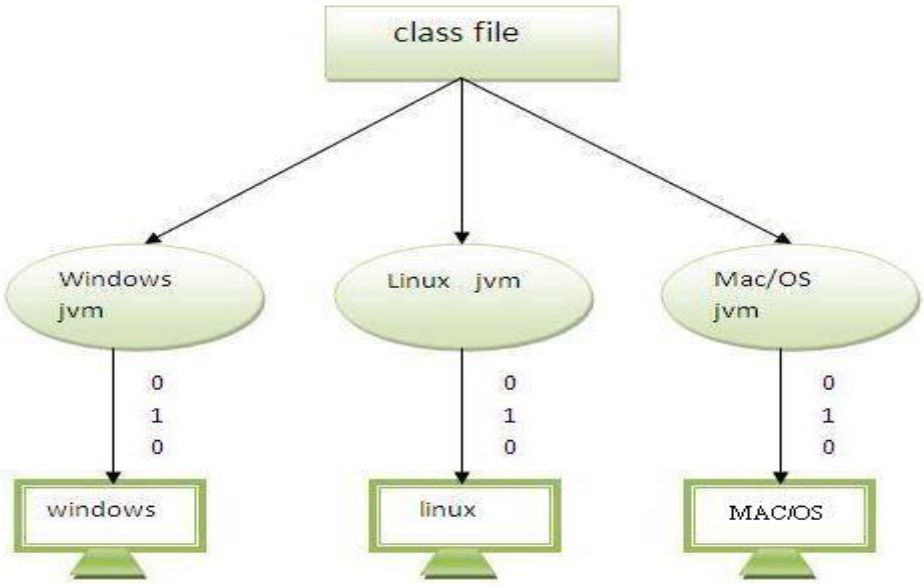
**Java Example:**

```
class Simple
{
   public static void main(String args[ ])
{
    System.out.println("Hello Java");
   }
}
```

## Java is Platform Independent

A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms.It has two components:

- **Runtime Environment**
- **API(Application Programming Interface)**



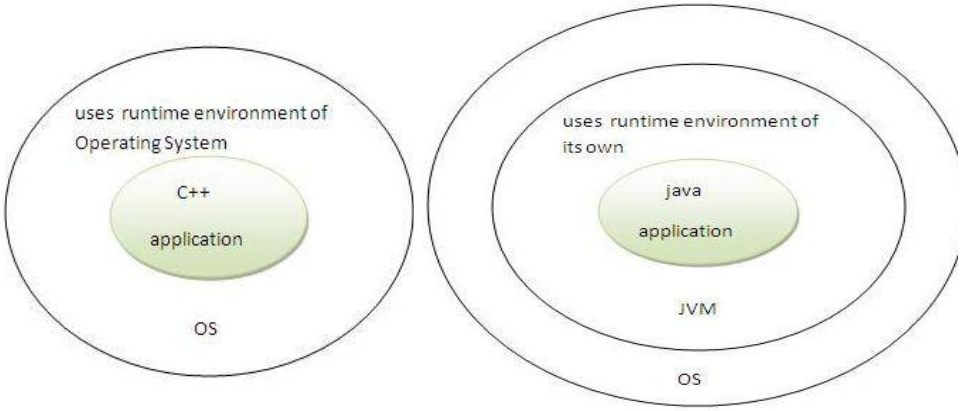

Java code can be run on multiple platforms e.g.Windows,Linux,Sun Solaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode.This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere (WORA).

Secured

Java is secured because:

No explicit pointer

Programs run inside virtual machine sandbox.




Classloader- adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Bytecode Verifier- checks the code fragments for illegal code that can violate access right to objects.

Security Manager- determines what resources a class can access such as reading and writing to the local disk.

## The difference between C++ and Java

### JAVA

- 1-Java does not support pointers, templates, unions, operator overloading, structures etc. The Java language promoters initially said "No pointers!", but when many programmers questioned how you can work without pointers, the promoters began saying "Restricted pointers." Java supports what it calls "references". References act a lot like pointers in C++ languages but you cannot perform arithmetic on pointers in Java. References have types, and they're type-safe. These references cannot be interpreted as raw address.
- 2- Java support automatic garbage collection. It does not support destructors as C++ does.
- 3- Java does not support conditional compilation and inclusion
- 4- Java has built in support for threads. In Java, there is a Thread class that you inherit to create a new thread and override the run() method.
- 4- java does not support default arguments. There is no scope resolution operator (::) in Java. The method definitions must always occur within a class, so there is no need for scope resolution there either.
- 5-There is no goto statement in Java. The keywords const and goto are reserved, even though they are not used.
- 6- Java doesn't provide multiple inheritance, at least not in the same sense that C++ does.
- 7- Exception handling in Java is different because there are no destructors. Also, in Java, try/catch must be defined if the function declares that it may throw an exception

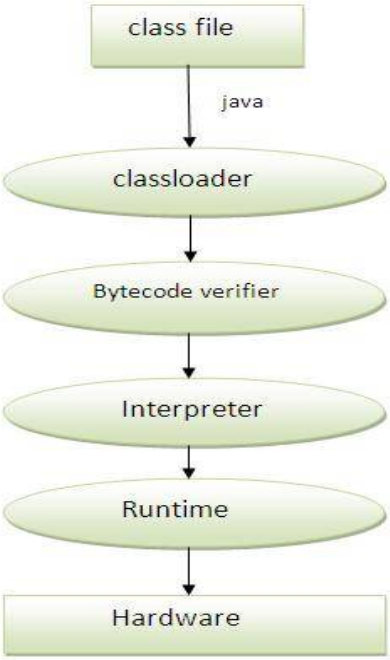
### C++

- C++ supports structures, unions, templates, operator overloading, pointers and pointer arithmetic.
- C++ support destructors, which is automatically invoked when the object is destroyed.

- Conditional inclusion (#ifdef #ifndef type) is one of the main features of C++.
- C++ has no built in support for threads. C++ relies on non-standard third-party libraries for thread support.
- C++ supports default arguments. C++ has scope resolution operator (::) which is used to to define a method outside a class and to access a global variable within from the scope where a local variable also exists with the same name.
- C++ has goto statement. However, it is not considered good practice to use of goto statement
- C++ does support multiple inheritance. The keyword virtual is used to resolve ambiguities during multiple inheritance if there is any.
- While in C++, you may not include the try/catch even if the function throws an exception.

## What happens at runtime:

At runtime, following steps are performed:



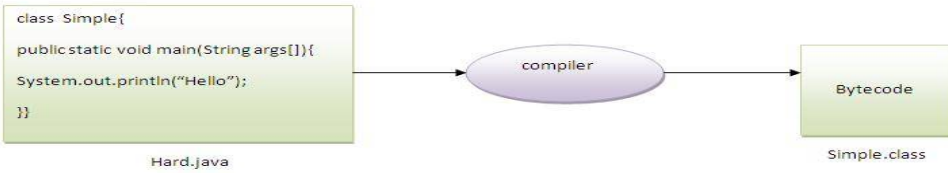

**Classloader**: is the subsystem of JVM that is used to load class files.

**Bytecode Verifier**: checks the code fragments for illegal code that can violate access right to objects.

**Interpreter**: read bytecode stream then execute the instructions.

- Can you save a java source file by other name than the class name?

Yes, if the class is not public. It is explained in the figure given below:




## Types of Java Program

- There are two types of Java programs
- Application Programs
- 2. Applet Programs

## Application Programs

Application programs are stand-alone programs that are written to carry out certain tasks on local computer such as solving equations, reading and writing files etc.

The application programs can be executed using two steps

1. Compile source code to generate Byte code using Javac compiler.
2. Execute the byte code program using Java interpreter.

## Applet programs:

Applets are small Java programs developed for Internet applications. An applet located in distant computer can be downloaded via Internet and executed on a local computer using Java capable browser. The Java applets can also be executed in the command line using appletviewer,which is part of the JDK.

## Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

## Servlet

Servlet technology is used to create web application (resides at server side and generates dynamic web page).

Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology.

There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

## Servlet

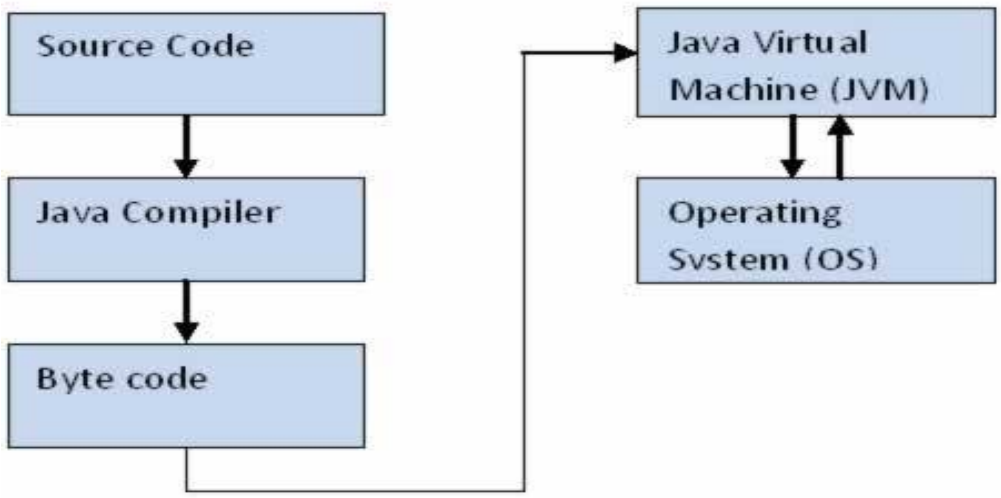
Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

## Java Architecture

- Compilation and interpretation in Java

Java combines both the approaches of compilation and interpretation. First, java compiler compiles the source code into bytecode. At the run time, Java Virtual Machine (JVM) interprets this bytecode and generates machine code which will be directly executed by the machine in which java program runs. So java is both compiled and interpreted language.



## Java Virtual Machine (JVM)

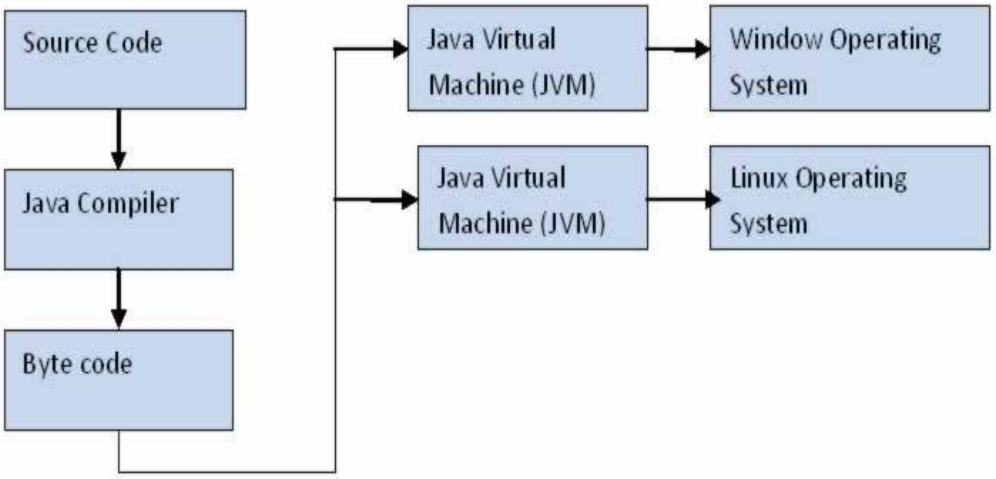
JVM is a component which provides an environment for running Java programs. JVM interprets the bytecode into machine code which will be executed the machine in which the Java program runs.

### ➢ Why Java is Platform Independent

Platform independence is one of the main advantages of Java. In another words, java is portable because the same java program can be executed in multiple platforms without making any changes in the source code. You just need to write the java code for one platform and the same program will run in any platforms.

As we discussed early, first the Java code is compiled by the Java compiler and generates the bytecode. This bytecode will be stored in class files. Java Virtual Machine (JVM) is unique for each platform. Though JVM is unique for each platform, all interpret the same bytecode and convert it into machine code required for its own platform and this machine code will be directly executed by the machine in which java program runs. This makes Java platform independent and portable.

Let's make it more clear with the help of the following diagram. Here the same compiled Java bytecode is interpreted by two different JVMS to make it run in Windows and Linux platforms.




## Java Runtime Environment (JRE) and Java Architecture in Detail

Java Runtime Environment contains JVM, class libraries and other supportingcomponents.

As you know the Java source code is compiled into bytecode by Java compiler. This bytecode will be stored in class files. During runtime, this bytecode will be loaded, verified and JVM interprets the bytecode into machine code which will be executed in the machine in which the Java program runs.

A Java Runtime Environment performs the following main tasks respectively.

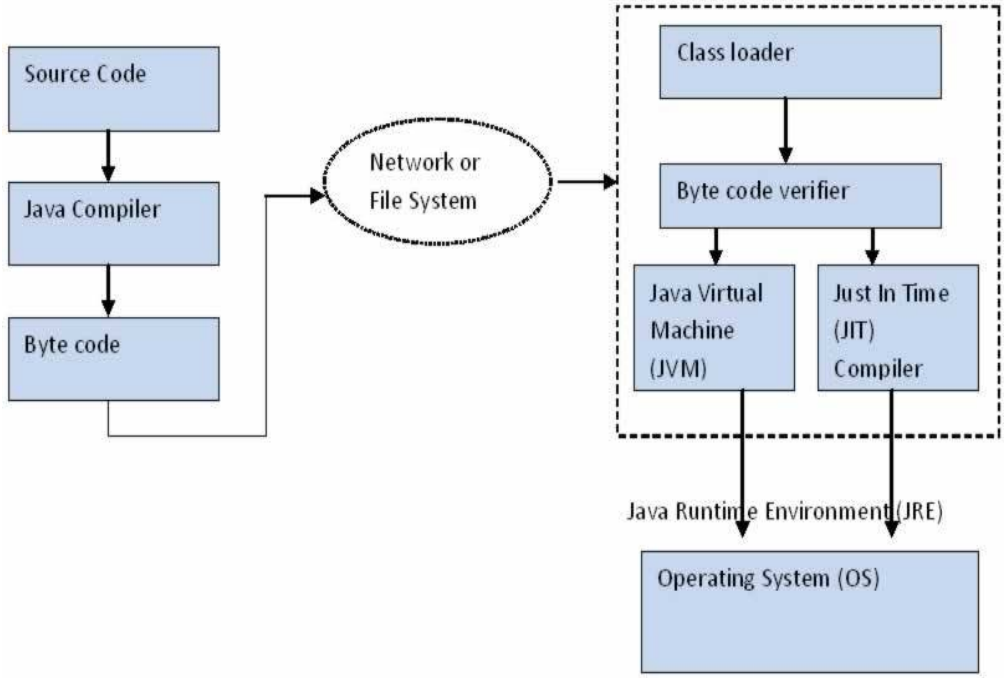
- Loads the class

  This is done by the class loader
- Verifies the bytecode

  This is done by bytecode verifier.
- Interprets the bytecode

  This is done by the JVM

These tasks are described in detail in the subsequent sessions.A detailed Java architecture can be drawn as given below.

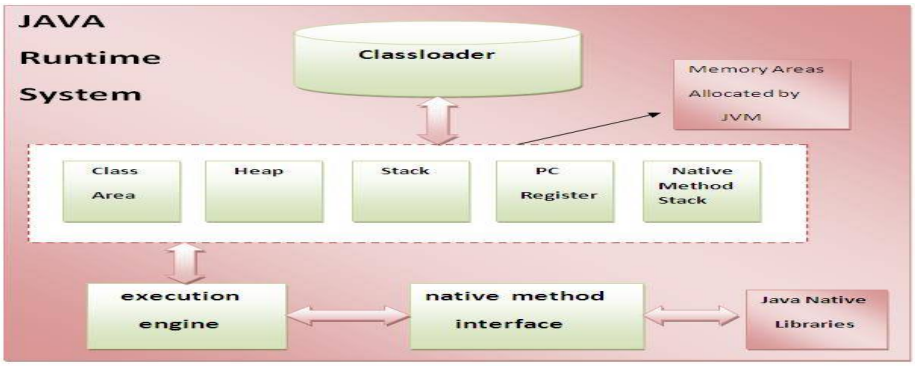



## JAVA STANDARD LIBRARY (JSL)

JSL is a library written in Java that provides a framework for general searching on graphs. While the standard search algorithms depth-first, breadth-first and A* are already provided by JSL you can easily plug in any other search algorithm you like. JSL must not be confused with libraries for full text or web search.

# Internal Architecture of JVM

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.




1) **Classloader:**

Classloader is a subsystem of JVM that is used to load class files.

2) **Class (Method) Area:**

Class (Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) **Heap**:

It is the runtime data area in which objects are allocated.

4) **Stack:**

- Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return.
- Each thread has a private JVM stack, created at the same time as thread.
- A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) **Program Counter Register:**

PC (program counter) registers. It contains the address of the Java virtual machine instruction currently being executed.

6) **Native Method Stack:**

It contains all the native methods used in the application.

7) **Execution Engine:**

It contains:

- 1) A virtual processor
- 2) Interpreter:Read bytecode stream then execute the instructions.

- 3) Just-In-Time(JIT) compiler:It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

## Review & Self Assessment Question

1. What is Application programming interface (API).
2. Differentiate C++ and Java Language.
3. What Java Virtual Machine (JVM).
4. What is Java standard Library (JSL)?

## Further Readings

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

**DETAILED PRACTICE**

# UNIT-3 JAVA TOKENS

## Contents

- Data types
- Tokens
- Variable
- Scope of variables
- Member variable
- Local variable
- Method parameter
- Operator
- Keyword
- Review & self assessment Question
- Further Readings

## Simple Program of Java

To create a simple java program, you need to create a class that contains main method.

- **Creating hello java example**

```
class Simple
{
   public static void main(String args [ ])
{
    System.out.println("Hello Java");
   }
}
```

## Understanding first java program:

Let's see what is the meaning of class, public, static, void, main, String [ ], System.out.println().

- **Class :**

  Class keyword is used to declare a class in java.

- **Public:**

  Public keyword is an access modifier which represents visibility, it means it is visible to all.

- **Static:**

Static is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.

- **Void:**

  Void is the return type of the method, it means it doesn't return any value.

- **Main**:

  Main represents startup of the program.

- **String[] args**

  String[ ] args is used for command line argument.

  - **System.out.println () is used print statement**.

## How many ways can we write a java program:

There are many ways to write a java program. The modifications that can be done in a java program are given below:

1) By changing sequence of the modifiers, method prototype is not changed.

static public void main (String args[ ])

2) Subscript notation in java array can be used after type, before variable or after variable.

public static void main(String[ ] args)

public static void main(String [ ]args)

public static void main(String args[ ])

3) You can provide var-args support to main method by passing 3 ellipses (dots)

public static void main(String... args)

4) Having semicolon at the end of class in java is optional.

Valid java main method signature

public static void main(String[ ] args)

public static void main(String [ ]args)

public static void main(String args[ ])

public static void main(String... args)

static public void main(String[ ] args)

public static final void main(String[ ]args)

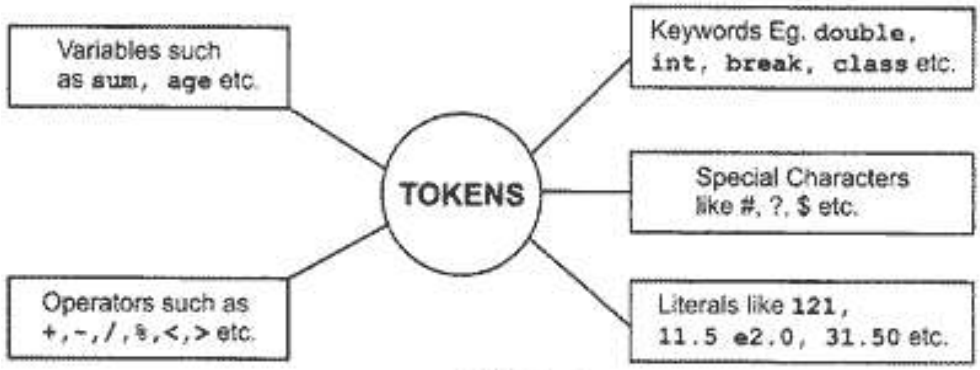
final public static void main(String[ ]args)

final strictfp public static void main(String[ ]args)

## DATA TYPES, VARIABLES AND OPERATORS

**Tokens: -** A Program is made up of Classes and Methods and in the Methods are the Container of the various Statements And a Statement is made up of Variables, Constants, operators etc.

**Tokens** are the various Java program elements which are identified by the compiler. A token is the smallest element of a program that is meaningful to the compiler. Tokens supported in Java include keywords, variables, constants, special characters, operations etc.




*Tokens in Java*

When you compile a program, the compiler scans the text in your source code and extracts individual tokens. While tokenizing the source file, the compiler recognizes and subsequently removes whitespaces (spaces, tabs, newline and form feeds) and the text enclosed within comments.

Now let us consider a program

```
//Print Hello
Public class Hello
{
Public static void main(String args [ ])
{
System.out.println(“Hello Java”);
}}
```

The source code contains tokens such as public, class, Hello, {, public, static, void, main, (, String, [ ], args, {, System, out, println, (, "Hello Java", }, }. The resulting tokens· are compiled into Java bytecodes that is capable of being run from within an interpreted java environment. Token

are useful for compiler to detect errors. When tokens are not arranged in a particular sequence, the compiler generates an error message.

Tokens are the smallest unit of Program There is Five Types of Tokens

1) Keywords
2) Identifier
3) Literals
4) Operators
5) Separators

## Keyword

In Java, a keyword is a word with a predefined meaning in Java programming language syntax. Reserved for Java, keywords may not be used as identifiers for naming variables, classes, methods or other entities.

## Reserved KeyWords

There are 50 reserved keywords in the Java programming language. Because these keywords have predefined functions, they are usually highlighted with different colors in most integrated development environments (IDE) used by Java programmers.

Common Java keywords include abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, for, while, switch, this and int. The keywords const and goto are reserved but not currently used by the Java language. False, null and true are reserved keywords for literal values.

Identifiers

An identifier is a sequence of one or more characters. The first character must be a valid first character (letter, $, _) in an identifier of the Java programming language, hereafter in this chapter called simply “Java”. Each subsequent character in the sequence must be a valid nonfirst character (letter, digit, $, _) in a Java identifier. (For details, see the Java SE API documentation of the isJavaIdentifierStart and isJavaIdentifierPart methods of the Character class.) The question mark (?) is a reserved character in the query language and cannot be used in an identifier.

A query language identifier is case-sensitive, with two exceptions:

- Keywords

Identification variables

## Java Constants

Constant in Java refer to fixed values that do not change during the execution of the program. Java supports several types of constants. They are

**1) Integer constant**

Integer constant refers to sequence of numbers without decimal point. There are three types of integer constants

**Decimal integers:** They consist of combination digits 0 through 9 withnegative sign.

**Example:**

105 -245 0

**octal integers:** They consist of combination of digits 0 through 7 withleading zero.

Example:

074 0 034

**Hexadecimal integer:** They consist of combination of digits 0 through 9 and letter a through f or A through F with leading 0x or 0X.

Example:

0X7A4 0Xa 0xA4

**2) Real constants**

Real constant refers to sequence of numbers with decimal point. They can be represented in two forms decimal notation and exponential notation

**Decimal notation :** The deciaml notation has three parts the integer part, the decimal point and the fractional part and can be used to represent sinle precision numbers.

Example:

2.15 -7.8 0.78

**Exponential notation:** The exponential notation is used to represent double precision numbers and has the following synatx

**mantissa e/E Exponent**

The mantissa is either a real number expressed in decimal notaion or an integer. The exponent is an integer withan optinal + or - sign. e2 means $10^2$.

Example:

0.97e4 12e-2 -1.2E-4

**3) Character constant**

A single character enclosed within single quotes is called character constant.

Example:

'5' 'E' '$'

**4) String Constants:**

A sequence of characters enclosed within double quotes is called string constant.

Example:

"Hai" "Java programming" "1998"

**5) Back slash character constant**

They are special character constants that are used in output methods.

Example:

'\b' - Back space

'\t' -Horizontal tab

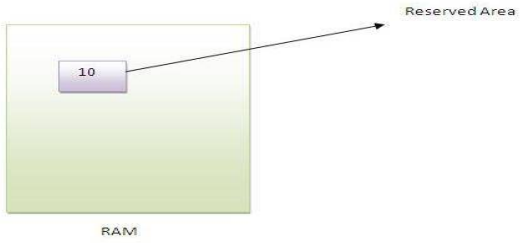
'\n' -New line

'\r' –Carriage return

These constants are also known as escape sequences.

## Variable:

Variable is name of reserved area allocated in memory.




int data=50;//Here data is variable

## Scope of Variables:

The scope of a Java variable is defined by the block of code within which the variable is accessible.

The scope also determines when the variable is created (memory set aside to contain the data stored in the variable) and when it possibly becomes a candidate for destruction (memory returned to the operating system for recycling and re-use).

## Scope categories

The scope of a variable places it in one of the following four categories:

- Member Variable
- Local Variable
- Method Parameter
- Exception handler parameter

## Member variable

A member variable is a member of a class (class variable) or a member of an object instantiated from that class (instance variable). It must be declared within a class, but not within the body of a method of the class.

## Local variable

A local variable is a variable declared within the body of a method or within a block of code contained within the body of a method.

## Method parameters

Method parameters are the formal arguments of a method. (A method is a function defined inside of a class.) Method parameters are used to pass values into and out of methods. The scope of a method parameter is the entire method for which it is a parameter.

## DATA TYPE:

The Java programming language is strongly-typed, which means that all variables must first be declared before they can be used. A variable's data type determines the values it may contain, plus the operations that may be performed on it. Java has two groups of data types

- Primitive data types
- Reference data types

## Java Primitive Data Types

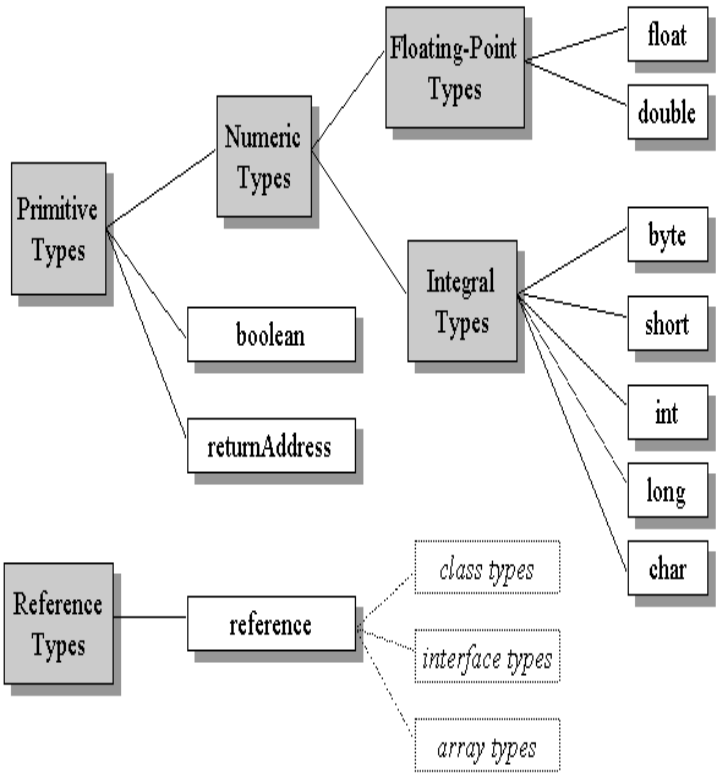
| Data Type | Purpose | Contents | Default Value* |
|---|---|---|---|
| Boolean | Truth value | true or false | Fales |
| Char | Character | Unicode characters | \u0000 |
| Byte | Signed integer | 8 bit two's complement | (byte) 0 |
| Short | Signed integer | 16 bit two's complement | (short) 0 |
| Int | Signed integer | 32 bit two's complement | 0 |
| Long | Signed integer | 64 bit two's complement | 0L |
| Float | Real number | 32 bit IEEE 754 floating point | 0.0f |
| Double | Real number | 64 bit IEEE 754 floating point | 0.0d |

## Java Reference Data Types

In Java a reference data type is a variable that can contain the reference or an address of dynamically created object.

These type of data type are not predefined like primitive data type.

- **The reference data types are:**
  - **Arrays**
  - **Classes**
  - **Interfaces**




## Operators in java

- **Operators:**

Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.

- **Operands**

An operands are the values on which the operators act upon.

## An operand can be:

**A numeric variable** - integer, floating point or character
**Any primitive type variable** - numeric and boolean
Reference variable to an object
**A literal** - numeric value, boolean value, or string.
**An array element**, "a[2]"
char primitive, which in numeric operations is treated as an unsigned two byte integer

## Types of Operators:

The operators can be classified into three types based on the **Number of operands**

1. Unary if it acts on a single operand (Eg : ++, --)
2. Binary if it requires two operands. (Eg: +, * )
3. Ternary if it requires three operands. The conditional operator is the only ternary operator in Java.

The operators can be classified into the following categories **based on the operations they perform**

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Bitwise Operators
5. Assignment Operators
6. Conditional Operators
7. Special Operator

## 1. Arithmetic Operators

| Operator | Name | Example | Result | Description |
|---|---|---|---|---|
| a + b | Addition | 3 + 5 | 8 | Adds the two operands( Can also be used for String Concatenation) |
| a – b | Subtraction | 3 – 5 | -2 | Subtracts the second operand from the first Operand. |
| a * b | Multiplication | 3 * 5 | 15 | Multiplie both the operands. |
| a / b | Division | 15 /3 | 5 | Divides the first operand by the second. |
| a % b | Modulo Division | 3 % 2 | 1 | Returns the remainder |

| | | | | |
|---|---|---|---|---|
| | | | | after dividing the first number by the second |

## 2.LOGICAL OPERATOR

Relational operators enable you to compare two variables to determine whether they are equal or if one is greater than the other, and so on. But when you want to check to see if a variable is in between a range of values or to check multiple conditions we use logical operators.

The following table lists the logical operators

| Operator | Name | Example | Result | Description |
|---|---|---|---|---|
| a && b | Logical AND | (3>5) && (2>5) | True | Returns true if the both operands is true. |
| a \|\| b | Logical OR | (3>5) && (10>5) | True | Returns true if any of the operands is true |
| ! a | NOT | ! (3>= 5) | True | Complements the result of the expression |

## 3.Bitwise Operator

Java's bitwise operators operate on individual bits of integer (int and long) values. If an operand is shorter than an int, it is promoted to int before doing the operations.

It helps to know how integers are represented in binary. For example the decimal number 3 is represented as 11 in binary and the decimal number 5 is represented as 101 in binary. Negative integers are store in two's complement form. For example, -4 is 1111 1111 1111 1111 1111 1111 1111 1100.

The following table lists bitwise operators

| **Operator** | **Name** | **Example** | **Result** | **Description** |
|---|---|---|---|---|
| a& b | and | 3 & 5 | 1 | 1 if both bits are 1. |
| a\|b | or | 3 \| 5 | 7 | 1 if either bit is 1. |
| a ^ b | xor | 3 ^ 5 | 6 | 1 if both bits are different. |
| ~a | not | ~3 | -4 | Inverts the bits. |
| n<<p | left shift | 3 <<< 2 | 12 | Shifts the bits of n left p positions. Zero bits are |

| | | | | |
|---|---|---|---|---|
| | | | | shifted into the low-order positions. |
| n>>p | right shift | 5 >> 2 | 1 | Shifts the bits of n right p positions. If n is a 2's complement signed number, the sign bit is shifted into the high-order positions. |
| n>>>p | right shift | -4 >>> 28 | 15 | Shifts the bits of n right p positions. Zeros are shifted into the high-order positions. |

## 4.Increment and Decrement :

The ++ and the – are java's increment and decrement operators.The increment operator increases its operand by one. The decrement operator decreases its operand by one.

**For example**:

This statement: x = x + 1 can be rewritten like this by use for the increment operator x++;Similarly, this statement x = x –1 is equivalent to x--;

Increment and decrement operators can be applied to all integers and floating point types.These operators are unique in that they can appear both in postfix form (x--, x++) and prefix form (--x, ++x), where they precede the operand.

In postfix form, the previous value is obtained for use in the expression, and then the operand is modified.

**For example:**

```
x =  42;
y =  ++x;
```

In this case, y is set to 43 as you would expect, because the increment occurs before x is assigned to y. thus, the line y=++x; is the equivalent of these two statements:

```
x =  42;
y =  x++;
```

The value of x is obtained before the increment operator is executed, so the value of y is 42. of course, in both cases x is set to 43. Here, the line y = x++; is the equivalent of these two statements:

```
y =  x;
```

x = x + 1;

## Example:

```
/* Demo increment and decrement operator */
class InDec
{   public static void main (String args[ ] {
        int a = 1;
        int b = 2;
        int c;
        int d;
        c = ++b;
        d = a++;
        c++;
        System.out.println("a = " + a);
        System.out.println("b= " + b);
        System.out.println("c = " + c);
        System.out.println("d = " + d);
    }
}
```

## 5.Relational Operators

Relational Operators are used for comparing numerals and the characters. The result of applying relational operators is either true or false. The following table lists the relational operators.

| Operator | Name | Example | Result | Description |
|---|---|---|---|---|
| a > b | Greater than | 3 > 5 | False | Returns true if the first operand is greater than the second operand |
| a < b | Lesser than | 5 < 5 | False | Returns true if the first operand is lesser than the second operand |
| a >= b | Greater than or equal | 3 >= 5 | False | Returns true if the first operand is greater than or equal to the second operand. |
| a <= b | Lesser than or equal to | 15 <= 15 | True | Returns true if the first operand is lesser than or equal to the second operand. |

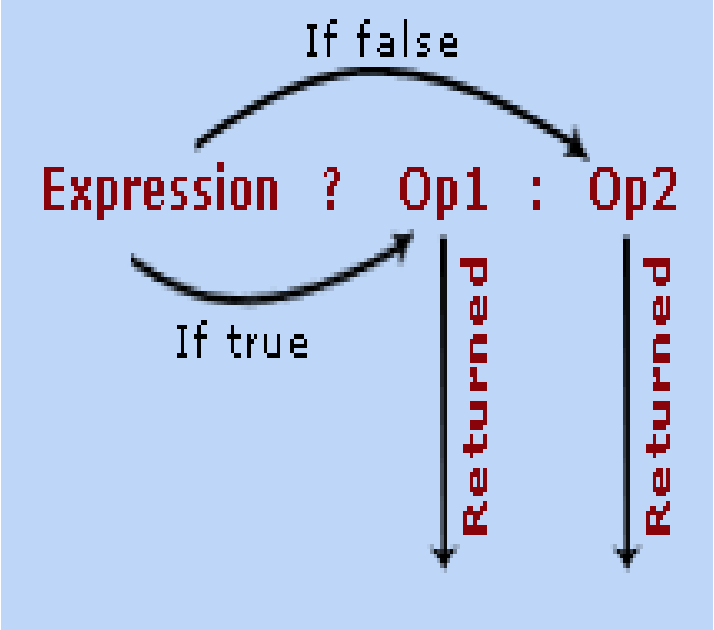
| | | | | |
|---|---|---|---|---|
| a == b | Equal To | 3 == 2 | False | Returns true if the first operand is equal to the second operand. |
| a != b | Not equal to | 3 != 2 | True | Returns true if the first operand is not equal to the second operand. |

## 6.Conditional Operaotr

Java supports another conditional operator that is known as the ternary operator "?:" It is basically used as an short hand for simple if..else

```
boolean expression ? operand1 : operand2;
```

The "?:" operator evaluates an expression which may also be an operand and returns operand1 if the expression istrue; otherwise returns operand2, if the expression is false. We can understand this thing with the help of a diagram shown as:




## Example:

```
int a=14, b=20;
int  x= (a>b) ? a : b;
```

In the above example the value of x is assigned the value of b since the condition is false.

# 7.Special Operators:

- **Instanceof Operator**

Java provides a run-time operator instanceof to compare a class and an instance of that class. This operator "instanceof" compares an object to a specified class type. The instanceof operator is defined to know about an object'srelationship with a class. It evaluates to true, if the object or array is an instance of the specified type; otherwise it returns false. The instanceof operator can be used with the arrays and objects. It can't be used with primitive data types and values.

Its syntax is

```
object instanceof type
```

**Example:**

```
class X
{
    int i, j;
}
class Y
{
   int i, j;
}
class Z extends X
{
    int k;
}
public class InstanceOfDemo
{
public static void main(String args[ ])
 {
        X x = new X();
        Y y = new Y();
        Z z = new Z();
        if(x instanceof X)
            System.out.println("x is instance of X");
        X obj;
        obj = z; // X reference to c
        if(obj instanceof Z)
            System.out.println("obj is instance of Z");
```

}}

- **Dot operator**

This operator is used to access the variables and methods of a class.

**Example1**

student.mark

Here we are accessing the variable “mark” of the “student” object

- ## Assignment Operator

Assignment operator is the most common operator almost used with all programming languages. It is represented by "=" symbol in Java which is used to assign a value to a variable lying to the left side of the assignment operator. But, If the value already exists in that variable then it will be overwritten by the assignment operator (=). This operator can also be used to assign the references to the objects.

**Syntax:** <variable> = <expression>;

**Example:** int a = 8;

## Precedence:

Which Operator Goes First?

- High Precedence (performed first)

| |
|---|
| **( )** |
| **unary -, +** |
| *** / %** |
| **+ -** |

- Low Precedence (performed last)

## Punctuation symbols

Separators are Java's punctuation. Just as periods, semicolons, question marks, pairs of parenthesis, and other punctuation add structure to written text, Java separators help the Java compiler interpret components of Java source code.

**( and ) Pairs of parenthesis surround**:

- arguments for a method in a method invocation
- method parameters and their type in a method declaration
- parts/pieces of an expression for the purposes of controling evaluation order or simply for clarity

- the mechanics of a for statement (the initialization, test, and update pieces)
- the test part of an if statement
- the test part of a while statement

[ and ]

- pairs of square brackets surround array indicies.

{ and }

- pairs of squiggly brackets surround a few things:

the body of a class
the body of a method
a block of statements to treated as a unit
sets of literals for array initialization

**; A semicolon terminates:**

Statements expression lists, e.g. the mechanics of a for statement (the initialization, test, and update pieces) field declarations

**, Acomma is used to separate items in sets/lists:**

type and identifier pairs in parameter lists litterals in an array initialization

**. a period is used to:**

separate pieces (identifiers) of a name, e.g. an object identifier and a field identifier or an object identifier and method identifier separate the whole number part from the fractional number part of a floating-point literal; it is the decimal point

## Unicode System

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

### Why java uses Unicode System?

Before Unicode, there were many language standards:

ASCII (American Standard Code for Information Interchange) for the United States.

ISO 8859-1 for Western European Language.

KOI-8 for Russian.

GB18030 and BIG-5 for chinese, and so on.

### This caused two problems:

A particular code value corresponds to different letters in the various language standards.

The encodings for languages with large character sets have variable

length.Some common characters are encoded as single bytes, other require two or more byte.

To solve these problems, a new language standard was developed i.e. Unicode System.

In unicode, character holds 2 byte, so java also uses 2 byte for characters.

- lowest value:\u0000
- highest value:\uFFFF

## Review & Self Assessment Question

1. What is data type?
2. What is variable and constant?
3. Define the term Token?
4. What is Reserved Keywords?
5. What is the Relational Operator in java?

## Further Readings

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

# UNIT:4 CONTROL STRUCTURE AND ITERATIVE STATEMENT

## Contents

- Input statemenet
- Output statement
- Control flow statemenet
- Decision making ststemenet
- Looping statemenet
- Loop control statement
- Type casting
- Review & self assessment Question
- Further Readings

## Input/Ouput

Programs are useless unless there is interaction with the outside world. You must be able to read input from the user write output where the user can see it

### There are two ways of doing this:

- console input/output:
- GUI input/output

Java is a language designed primarily for developing programs with a graphical interface. As a result, performing console I/O (particularly reading input) is rather awkward. Note: Many older languages were designed for console I/O only.

To help ease you into the details of I/O (which can be a mess) the author of your text has written several classes that provide a clean I/O graphical interface.

## Input Statement :

One of the strengths of Java is the huge libraries of code available to you. This is code that has been written to do specific jobs. All you need to do is to reference which library you want to use, and then call a method into action. One really useful class that handles input from a user is called the

Scanner class. The Scanner class can be found in the java.util library. To use the Scanner class, you need to reference it in your code. This is done with the keyword import.

```
import java.util.Scanner;
```

The import statement needs to go just above the Class statement:

```
import java.util.Scanner;
public class StringVariables
{
}
```

This tells java that you want to use a particular class in a particular library - the Scanner class, which is located in the java.util library.

The next thing you need to do is to create an object from the Scanner class. (A class is just a bunch of code. It doesn't do anything until you create a new object from it.)

**To create a new Scanner object the code is this:**

```
Scanner user_input = new Scanner( System.in );
```

So instead of setting up an int variable or a String variable, we're setting up a Scanner variable. We've called ours user_input. After an equals sign, we have the keyword new. This is used to create new objects from a class. The object we're creating is from the Scanner class. In between round brackets we have to tell java that this will be System Input (System.in).

To get the user input, you can call into action one of the many methods available to your new Scanner object. One of these methods is called next. This gets the next string of text that a user types on the keyboard:

```
String first_name;
first_name = user_input.next( );
```

So after our user_input object we type a dot. You'll then see a popup list of available methods. Double click next and then type a semicolon to end the line. We can also print some text to prompt the user:

```
String first_name;
        System.out.print("Enter your first name:");
        first_name = user_input.next( );
```

Notice that we've used print rather than println like last time. The difference between the two is that println will move the cursor to a new line after the output, but print stays on the same line.

We'll add a prompt for a family name, as well:

```
String family_name;
        System.out.print("Enter    your    family name:");
        family_name = user_input.next( );
```

This is the same code, except that java will now store whatever the user types into our family_name variable instead of our first_name variable.

To print out the input, we can add the following:

```
String full_name;
full_name=first_name+"    "+family_name;
System.out.println("You are " + full_name);
```

We've set up another String variable, full_name. We're storing whatever is in the two variables first_name and family_name. In between the two, we've added a space. The final line prints it all out in the Output window.

## Output Statement :

There are no I/O statements in the Java language. The I/O methods belong to classes in the java.io package. Any source or destination for I/O is considered a stream of bytes. There are three objects that can be used for input and output.

- System.out can be used to write output to the console.
- System.err can be used to write error messages to the console.
- System.in can be used to handle input from the console.

The System.out object has two methods - print() and println(). The println() method will print a carriage return and line feed after printing so that the next output will be printed on a new line. The method print() will keep the cursor on the same line after printing.

In Java 1.5 you can also use the method printf(). The advantage of using the printf() method is that you can now format your output easily. The structure of the printf syntax is as follows:

System.out.printf (format, args);

## Control flow statements:

A program is a group of statements that are executed to achieve a predetermined task. Statements in a program are generally executed in a sequential manner, which is called sequential execution or sequential control flow. However, by putting the decision-making statement in the program, the normal flow of the program can be controlled. Statements that control the flow of the program are called control statements.

Control statements are used in programming languages to cause the flow of control to advance and branch based on changes to the state of a program.

The kinds of control flow statements supported by different languages vary, but can be categorized by their effect:

Continuation at a different statement (jump),

Executing a set of statements only if some condition is met (choice),

Executing a set of statements zero or more times, until some condition is met (loop),

Executing a set of distant statements, after which the flow of control may possibly return (subroutines, coroutines, and continuations),

Stopping the program, preventing any further execution (halt).

Java supports three types of control structures

* Decision-making statements(if-then,if-then-else,switch),
* Looping statements(for,while,do-while),
* Branching statements(break, continue, return)

## Decision Making Statement

The decision-making statements are used to change the flow of the program based on conditions. The various decision-making statements are

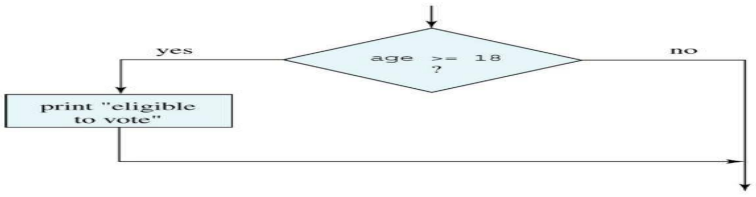
- if statement
- if..else statement
- if..else if Ladder
- Nested if statements
- switch statement

## If Statement

Ensures that a statement is executed only when a condition is true Conditions typically involve comparison of variables or quantities for equality or inequality

**Example**

```
if (age >= 18)
System.out.println("You are eligible to vote.");
```

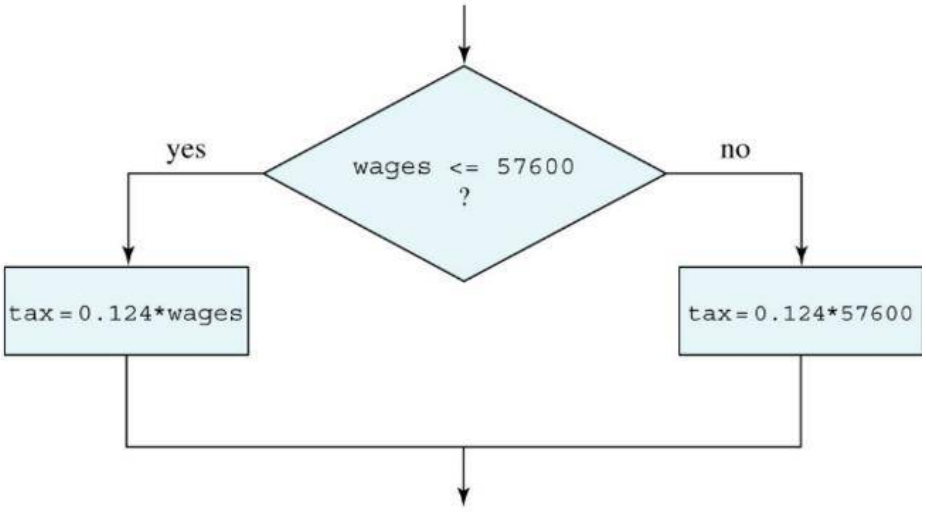


## If Statement with Optional else

An if statement may have an optional else clause that will only be executed when the condition is false

**Example:**

```
if (wages <= 57600)
        tax = 0.124 * wages;
else
        tax = 0.124 * 57600;
```




## Compound Alternatives

To execute more than one statement conditionally, you may use { } to define a compound statement for either (or both) conditional alternatives

**Example:**

```
if (firstNumber <= secondNumber)
{
        quotient = secondNumber / firstNumber;
        remainder = secondNumber % firstNumber;
}
```

**Cascading if-else Statements**

```
else
{
        quotient = firstNumber / secondNumber;

        remainder = firstNumber % secondNumber;
}
```

## Cascading if-else Statements

**Example:**

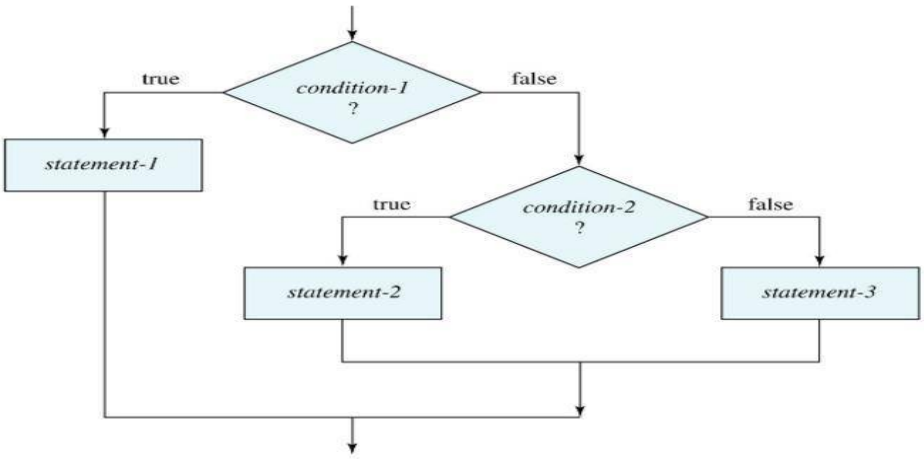
format:

```
if (condition-1)
        statement-1;
    else
if (condition-2)
        statement-2;

        else
          statement-3;
```

Another format:

```
        if (condition-1)
statement-1;
     else if (condition-2)
statement-2;
else
          statement-3;
```



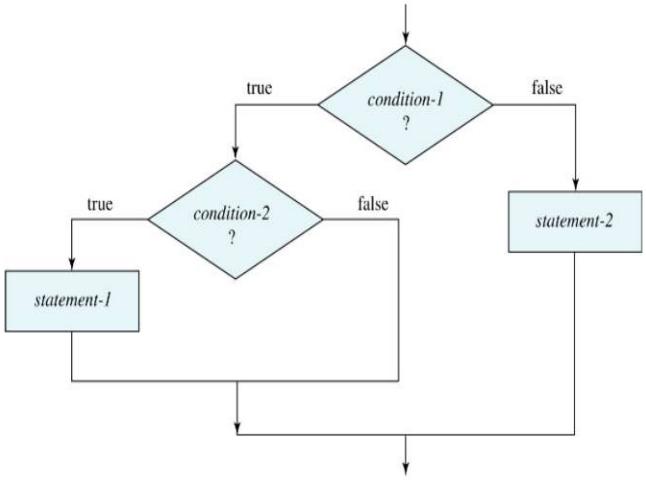

**Dangling else**

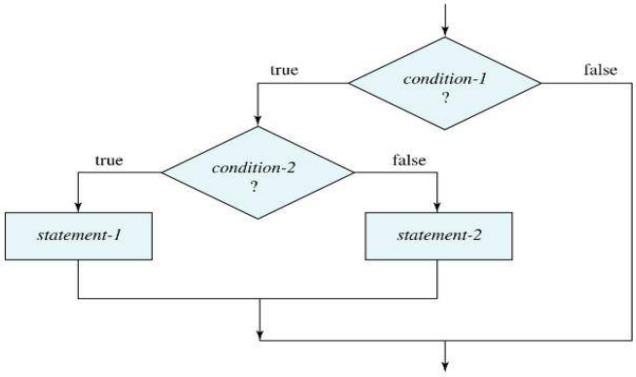
Code written:

```
if (condition-1)
  if (condition-2)
 statement-1
    else
      statement-2;
```

Interpreted as:

```
        if (condition-1)
 if (condition-2
 statement-2
else
statement-2;
```




## Switch Statement:

Used to accomplish multi-way branching based on the value of an integer selector variable

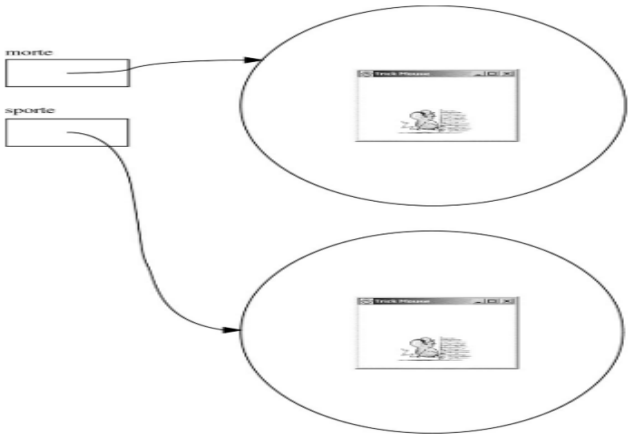
**Example:**

```
switch (number of passengers)
{
        case 0: out.println("The Harley");

        case 1: out.println("The Dune Buggy");

         default: out.println("The Humvee");

}
```




## Looping Statements

Looping statements are used to execute single or group of statements repeatedly until thegiven condition is true. Java supports three types of Looping statements

1. **While Loop**
2. **Do...while Loop**
3. **For Loop**

| Loop Type | Description |
|---|---|
| While Loop in Java | Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body. |
| For loop in Java | Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| Do…. While loop in java | Like a while statement, except that it tests the condition at the end of the loop body |

## While Statement :

**The while Statement:**

This executes a block of statements as long as a specified condition is true.

**Syntax:**

```
    while(condition)
{
    statement(s);
```

}

next_statement;

The (condition) may be any valid Java expression.

The statement(s) may be either a single or a compound (a block) statement.

**Execution:**

When program execution reaches a while statement, the following events occur:

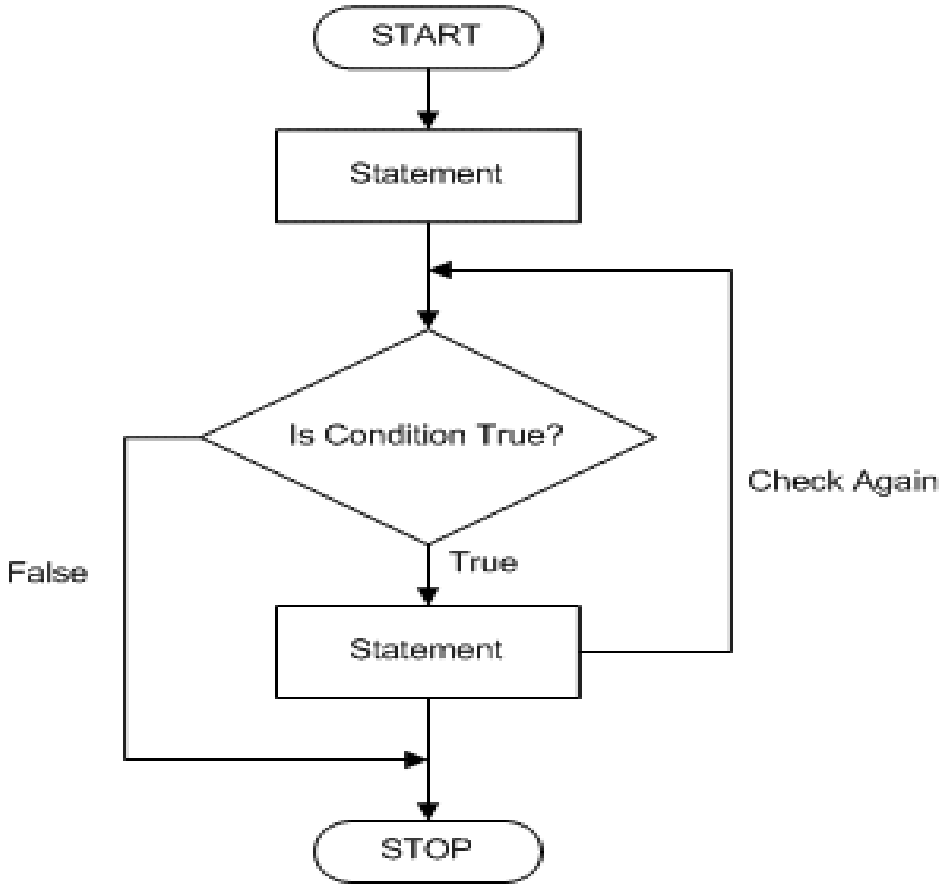
The (condition) is evaluated.

If (condition) evaluates as false the while statement terminates and execution passes to the first statement following statement(s) that is the next_statement.

If (condition) evaluates as true the statement(s) iside the loop are executed.

Then, the execution returns to step number 1.

## Flow chart:



## Do..while Statement:

do..while loop is used when you need to run some statements and processes once at least before checking the condition to execute the loop.

**Syntax:**

```
do
{
   statements;
}
while(condition);
        next Statement;
```

**Execution:**

1. The statement in the do..while block is executed once.

2.Evaluate the condition.

3. If the condition evaluates to true goto step1. If the condition is false then goto step 4.

4. Execute the statement following the do..while statement.

**Example:**

```
/* Program to find the reverse of the number */
public class DoWhile
{
        public static void main(String[] args)
        {
                int n = 12345;
                int t,r = 0;
                System.out.println("The original number : " + n);
                do
                {
                        t = n % 10;
                        r = r * 10 + t;
                        n = n / 10;
                }while (n &gt; 0);
                System.out.println("The reverse number : " + r);
        }
}
```

## For Loop :

The for loop statement is similar to while loop. Usually it is used to execute set of statements repeatedly a known number of times.

**Syntax:**

**for**( initialization; termination; increment/decrement)

{

statements;

}

next Statement;

## Execution:

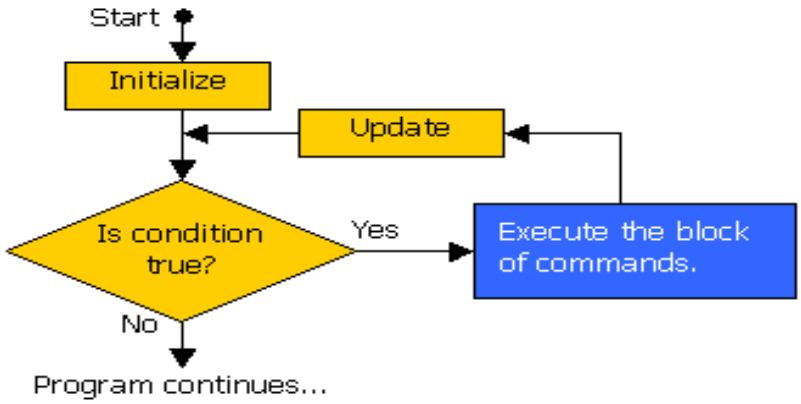
When the loop first starts, the initialization portion of the loop is executed. Generally, this is an expression that sets the value of the loop control variable, which acts as a counter that controls the loop.The initialization expression is only executed once.

Next, condition is evaluated. This must be a Boolean expression. It usually tests the loop control variable against a target value. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates.

Next, the increment/ decrement portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable.

The loop then iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.

**Flowchart**




**Example:**

/* Printing numbers from 1 to 5 */

**class** ForDemo

{

```
public static void main(String args[])
{
   for(int i=1; i&lt;=5; i++)
   {
            System.out.println("value of i="i);
   }}}
```

## Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

| Control Statement | Description |
|---|---|
| Break statement in java | Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch. |
| Continue statement in java | Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println () method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

## Creating Method:

Considering the following example to explain the syntax of a method:

```
public static int methodName(int a, int b)
{
  // body
}
```

Here,

Public static: modifier.

int: return type

methodName: name of the method

a, b: formal parameters

int a, int b: list of parameters

Method definition consists of a method header and a method body. The same is shown below:

```
Modifier returnType nameOfMethod (Parameter List)
{
 // method body
}
```

The syntax shown above includes:

**Modifier**: It defines the access type of the method and it is optional to use.

returnType: Method may return a value.

**NameofMethod**: This is the method name. The method signature consists of the method name and the parameter list.

**Parameter List**: The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.

**Method body**: The method body defines what the method does with statements.

**Example:**

Here is the source code of the above defined method called max(). This method takes two parameters num1 and num2 and returns the maximum between the two:

```
/** the snippet returns the minimum between two numbers */
        public static int minFunction(int n1, int n2)
        {
          int min;
          if (n1 > n2)
            min = n2;
  else
           min = n1;

        return min;
}
```

## Method Calling:

For using a method, it should be called. There are two ways in which a method is called i.e. method returns a value or returning nothing (no return value).

The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This

called method then returns control to the caller in two conditions, when: return statement is executed.
reaches the method ending closing brace.
The methods returning void is considered as call to a statement.

- **Lets consider an example**:

  System.out.println("This is tutorialspoint.com!");

The method returning value can be understood by the following example:

int result = sum(6, 9);

- **Example**:

Following is the example to demonstrate how to define a method and how to call it:

```
public class ExampleMinNumber
{
  public static void main(String[] args)
{
    int a = 11;
    int b = 6;
    int c = minFunction(a, b);
    System.out.println("Minimum Value = " + c);
  }
  /** returns the minimum of two numbers */
  public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
      min = n2;
    else
      min = n1;

    return min;
  }}
```

## Type Casting:

Assigning a value of one type to a variable of another type is known as Type Casting.
Example :

```
int x = 10;
byte y = (byte)x;
```

In Java, type casting is classified into two types,

- **Widening Casting(Implicit)**


- **Narrowing Casting(Explicitly done)**


## Widening or Automatic type conversion:

Automatic Type casting take place when, the two types are compatible the target type is larger than the source type

**Example :**

```
public class Test
{
   public static void main(String[ ] args)
   {
     int i = 100;
     long l = i;  //no explicit type casting required
     float f = l;  //no explicit type casting required
     System.out.println("Int value "+i);
     System.out.println("Long value "+l);
     System.out.println("Float value "+f);
}}
```

## Narrowing or Explicit type conversion:

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

Example :

```
public class Test
{
   public static void main(String[] args)
   {
```

```
double d = 100.04;
long l = (long)d;  //explicit type casting required
int i = (int)l;          //explicit type casting required
System.out.println("Double value "+d);
System.out.println("Long value "+l);
System.out.println("Int value "+i);
  }}
```

## Review & Self Assessment Question

1. What is control flow statements?
2. What is decision making statements?
3. Define for loop with example.
4. Define the Break and Continue statement?
5. What is entry controlled loop and Exist controlled Loop.

## Further Readings

Programming in Java by Herbert Schildt
Programming in Java by Kathy Sierra and Bert Bates
Programming in Java by Harimohan Pandey

# UNIT-5 OOP'S CONCEPTS WITH CLASSES AND OBJECTS

## Contents



Object is the physical as well as logical entity whereas class is the logical entity only.

**Object in Java:**

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tengible and intengible). The example of integible object is banking system.

**An object has three characteristics:**

- state: represents data (value) of an object.
- behavior: represents the behavior (functionality) of an object such as deposit, withdraw etc.
- identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But,it is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc.

known as its state. It is used to write, so writing is its behavior.

Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

**Class in Java:**

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

**A class in java can contain**:

data member
method
constructor
block

**class and interface**

Syntax to declare a class:

```
class <class_name>
{  data member;
   method;  }
```

## Simple Example of Object and Class:

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

```
class Student1{
        int id;//data member (also instance variable)
        String name;//data member(also instance variable)
  public static void main(String args[ ]){

    Student1 s1=new Student1();//creating an object of Student
        System.out.println(s1.id);
         System.out.println(s1.name);
 } }
```

## Java Naming conventions:

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

## Advantage of naming conventions in java:

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

| Name | Convention |
|---|---|
| class name | should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| interface name | should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc. |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| variable name | should start with lowercase letter e.g. firstName, orderNumber etc. |
| package name | should be in lowercase letter e.g. java, lang, sql, util etc. |
| constants name | should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc. |

## Method Overloading in Java

If a class have multiple methods by same name but different parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

### Advantage of method overloading

- Method overloading increases the readability of the program

### Different ways to overload the method:

There are two ways to overload the method in java

- By changing number of arguments
- By changing the data type

**Example of Method Overloading by changing the no. of arguments:**

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

```
class Calculation
{
        void sum(int a,int b){System.out.println(a+b);
}
void sum(int a,int b,int c){System.out.println(a+b+c);
}
 public static void main(String args[ ])
{
 Calculation obj=new Calculation();
 obj.sum(10,10,10);
  obj.sum(20,20);  }  }
```

**Example of Method Overloading by changing data type of argument:**

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

```
class Calculation2
{
 void sum(int a,int b)
{
System.out.println(a+b);
}
  void sum(double a,double b)
{
System.out.println(a+b);
}
 public static void main(String args[ ])
{
```

```
        Calculation2 obj=new Calculation2();
        obj.sum(10.5,10.5);
         obj.sum(20,20);
 } }
```

## Constructor:

Constructor in java is a special type of method that is used to initialize the object. Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

### Rules for creating java constructor

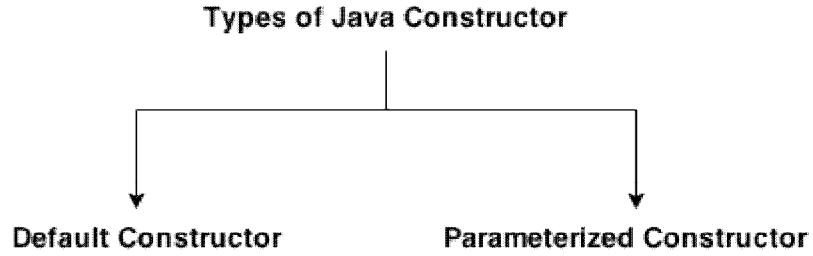
There are basically two rules defined for the constructor.

- Constructor name must be same as its class name
- Constructor must have no explicit return type

### Types of java constructors

There are two types of constructors:

- Default constructor (no-arg constructor)
- Parameterized constructor




## Java Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

```
<class_name>()
{
}
```

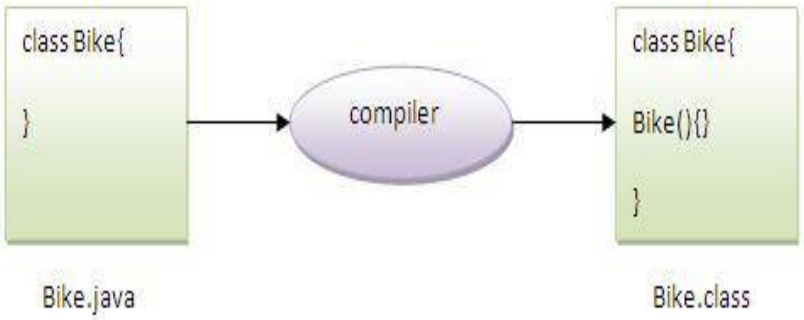
**Example of default constructor:**

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1
{
```

```
        Bike1( )
{
        System.out.println("Bike is created");
}
public static void main(String args[])
{
        Bike1 b=new Bike1();
} }
```

**Rule: If there is no constructor in a class, compiler automatically creates a default constructor**.




## Purpose of default constructor:

Default constructor provides the default values to the object like 0, null etc. depending on the type.

**Example of default constructor that displays the default values:**

```
class Student3
{
int id;
String name;

void display( )
{
        System.out.println(id+" "+name);
        }
public static void main(String args[ ])
{
        Student3 s1=new Student3( );
        Student3 s2=new Student3( );
```

```
        s1.display( );
        s2.display( );
}}
```

**Explanation**:In the above class,you are not creating any constructor so compiler provides you a default constructor.Here 0 and null values are provided by default constructor.

## Java parameterized constructor:

A constructor that have parameters is known as parameterized constructor.

### Use of parameterized constructor

Parameterized constructor is used to provide different values to the distinct objects.

### Example of parameterized constructor:

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4
{
   int id;
   String name;
   Student4(int i,String n)
{
        id = i;
        name = n;
  }
   void display( )
{
        System.out.println(id+" "+name);
}
   public static void main(String args[ ])
{
   Student4 s1 = new Student4(111,"kamlesh");
   Student4 s2 = new Student4(222,"vijay");
        s1.display();
        s2.display();
  }
}
```

## Static keyword:

The static keyword in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

variable (also known as class variable)
method (also known as class method)
block
nested class

1) **Java static variable**

If you declare any variable as static, it is known static variable.

The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.

The static variable gets memory only once in class area at the time of class loading.

**Advantage of static variable**

- It makes your program memory efficient (i.e it saves memory).

**Understanding problem without static variable**

```
class Student
{
    int rollno;
   String name;
  String college="ITS";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created.All student have its unique rollno and name so instance data member is good.Here, college refers to the common property of all objects.If we make it static,this field will get memory only once.

**Example of static variable:**

```
//Program of static variable
class Student8
{
  int rollno;
  String name;
  static String college ="ITS";
  Student8(int r,String n)
```

```
{
        rollno = r;
        name = n;
  }
 void display ( )
{
        System.out.println(rollno+" "+name+" "+college);
}
 public static void main(String args[ ])
{
        Student8 s1 = new Student8(111,"suresh");
        Student8 s2 = new Student8(222,"sandeep");
        s1.display();
        s2.display();
 }}
```

## This keyword:

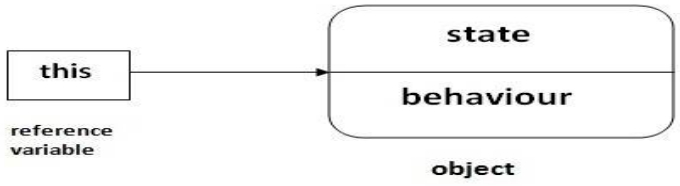
There can be a lot of usage of java this keyword. In java, this is areference variable that refers to the current object.

**Usage of java this keyword**

Here is given the 6 usage of java this keyword.

- this keyword can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- this keyword can be used to invoke current class method (implicitly)
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this keyword can also be used to return the current class instance.

Suggestion: If you are beginner to java, lookup only two usage of this keyword.




1) **The this keyword can be used to refer current class instance variable**.

If there is ambiguity between the instance variable and parameter, this

keyword resolves the problem of ambiguity.

**Understanding the problem without this keyword**

Let's understand the problem if we don't use this keyword by the example given below:

```
class Student10
{
   int id;
   String name;
   Student10(int id,String name)
{
   id = id;
   name = name;
   }
   void display( )
{
        System.out.println(id+" "+name);
}
   public static void main(String args[ ])
{
            Student10 s1 = new Student10(111,"Rahul");
           Student10 s2 = new Student10(321,"Rohan");
           s1.display();
         s2.display();
   } }
```

In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable.

**Solution of the above problem by this keyword**

```
//example of this keyword
class Student11
{
   int id;
   String name;

   Student11(int id,String name)
{
```

```
    this.id = id;
    this.name = name;
  }
    void display(){System.out.println(id+" "+name);
}
    public static void main(String args[])
{
    Student11 s1 = new Student11(111,"Suresh");
    Student11 s2 = new Student11(222,"Vijay");
    s1.display();
    s2.display();
}}
```

## Exploring methods and inheritance

## Aggregation in Java

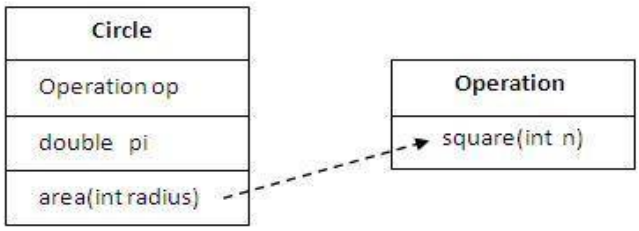
If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```
class Employee
{
int id;
String name;
Address address;//Address is a class
...
}
```

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

### Simple Example of Aggregation



In this example, we have created the reference of Operation class in the Circle class.

```
class operation
{
 int square(int n)
{
        return n*n;
 }}
class Circle
{
       operation op;//aggregation
        double pi=3.14;
        double area(int radius)
{
        op=new operation();
          int rsquare=op.square(radius);//code reusability (i.e. delegates the
           method call).
        return pi*rsquare;
 }
 public static void main(String args[ ])
{
        Circle c=new Circle( );
        double result=c.area(5);
       System.out.println(result);
 }}
```

## When use Aggregation:

Code reuse is also best achieved by aggregation when there is no is-a relationship.

Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

**Understanding meaningful example of Aggregation:**

In this example, Employee has an object of Address, address object contains its own informations such as city, state, country etc. In such case relationship is Employee HAS-A address.

**File Name** : Address.java

```
public class Address
```

```
{
        String city,state,country;
public Address(String city, String state, String country)
{
   this.city = city;
   this.state = state;
   this.country = country;
} }
```

**Emp.java**

```
public class Emp
{
        int id;
        String name;
        Address address;
public Emp(int id, String name,Address address)
{
         this.id = id;
           this.name = name;
         this.address=address;
}
void display( )
{
System.out.println(id+" "+name);
System.out.println(address.city+" "+address.state+" "+address.country);
}
public static void main(String[ ] args)
{
        Address address1=new Address("allahabad","UP","india");
        Address address2=new Address("varansi","UP","india");
        Emp e=new Emp(111,"tanu",address1);
        Emp e2=new Emp(112,"arun",address2);
        e.display();
        e2.display();
}}
```

## Call by Value in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being

done in the called method, is not affected in the calling method.

## Example of call by value

In case of call by value original value is not changed. Let's take a simple example:

```
class Operation
{
 int data=50;
 void change(int data)
{
 data=data+100;//changes will be in the local variable only
 }
 public static void main(String args[ ])
{
  Operation op=new Operation();
   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);
}}
```

## Another Example of call by value

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value. Let's take a simple example:

```
class Operation2
{
 int data=50;
 void change(Operation2 op)
{
          op.data=op.data+100;//changes will be in the instance variable
 }
 public static void main(String args[ ])
{
            Operation2 op=new Operation2();
            System.out.println("before change "+op.data);
            op.change(op);//passing object
            System.out.println("after change "+op.data);
```

}}

## Polymorphism

Polymorphism in java is a concept by which we can perform a single action by different ways. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload static method in java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

## Inheritance

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the IS-A relationship, also known as parent-child relationship.

**Why use inheritance in java**

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

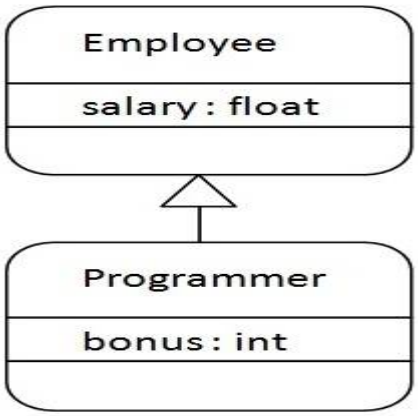
Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class.

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

Understanding the simple example of inheritance




As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is Programmer IS-A Employee.It means that Programmer is a type of Employee.

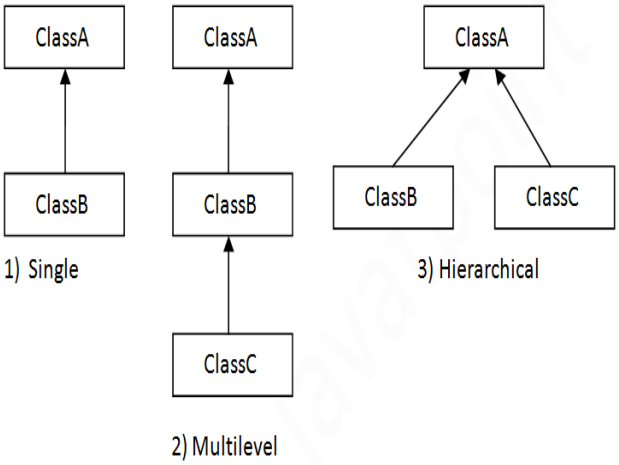
```
class Employee
{
        float salary=40000;
}
class Programmer extends Employee
{
        int bonus=10000;
 public static void main(String args[])
{
        Programmer p=new Programmer();
       System.out.println("Programmer salary is:"+p.salary);
       System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

## Types of inheritance

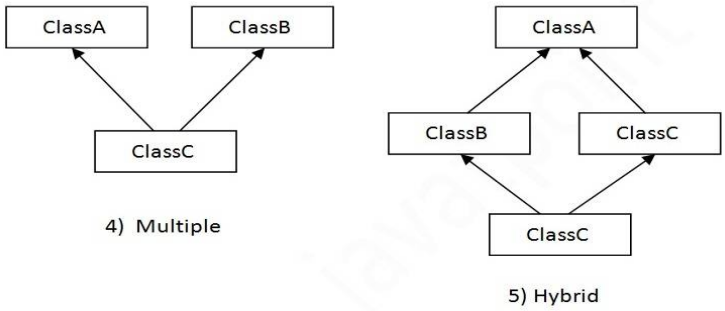
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.




Note: Multiple inheritance is not supported in java through class.

**When a class extends multiple classes i.e. known as multiple inheritance. For Example:**




**Multiple inheritance is not supported in java:**

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

```
class A
{
void msg( )
{
System.out.println("Hello");
```

```
}}
class B
{
void msg( )
{
System.out.println("Welcome");
} }
class C extends A,B
{
//suppose if it were
 Public Static void main(String args[ ] )
{
  C obj=new C();
  obj.msg();//Now which msg( ) method would be invoked?
} }
```

## Method Overriding:

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in java.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

### Usage of Java Method Overriding:

Method overriding is used to provide specific implementation of a method that is already provided by its super class.

Method overriding is used for runtime polymorphism

**Rules for Java Method Overriding:**

- method must have same name as in the parent class
- method must have same parameter as in the parent class.
- must be IS-A relationship (inheritance).

**Understanding the problem without method overriding:**

Let's understand the problem that we may face in the program if we don't use method overriding.

```
class Vehicle
{
  void run( )
{
        System.out.println("Vehicle is running");
```

```
}}
class Bike extends Vehicle
{
  public static void main(String args[ ])
{
          Bike obj = new Bike( );
           obj.run( );
  }  }
```

## Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

```
class Vehicle
{
void run( )
{
        System.out.println("Vehicle is running");
}
}
class Bike2 extends Vehicle
{
void run( )
{
        System.out.println("Bike is running safely");
}
public static void main(String args[ ])
{
Bike2 obj = new Bike2();
obj.run();
}
```

## Runtime Polymorphism

Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

## Upcasting:

When reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:


```
class A
{

}
class B extends A
{

}
A a=new B(  );//upcasting
```

## Example of  Runtime Polymorphism

In this example, we are creating two classes Bike and Splendar. Splendar class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at runtime.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

```
class Bike
{
  void run()
{
        System.out.println("running");
}
}
```

```
class Splender extends Bike
{
  void run( )
{
        System.out.println("running safely with 60km");
}
  public static void main(String args[ ])
{
    Bike b = new Splender();//upcasting
    b.run();
  }
}
```

## Runtime Polymorphism with data member

Method is overridden not the datamembers, so runtime polymorphism can't be achieved by data members.

In the example given below, both the classes have a datamember speedlimit, we are accessing the datamember by the reference variable of Parent class which refers to the subclass object. Since we are accessing the datamember which is not overridden, hence it will access the datamember of Parent class always.

**Rule: Runtime polymorphism can't be achieved by data members**.

```
class Bike
{
        int speedlimit=90;
}
class Honda3 extends Bike
{
        int speedlimit=150;
 public static void main(String args[ ])
{
        Bike obj=new Honda3( );
        System.out.println(obj.speedlimit);//90
}
```

## Runtime Polymorphism with Multilevel Inheritance

Simple example of Runtime Polymorphism with multilevel inheritance.

```
class Animal
```

```
{
void eat( )
{
System.out.println("eating");
}}
class Dog extends Animal
{
void eat( )
{
        System.out.println("eating fruits");
}}
class BabyDog extends Dog
{
void eat( )
{
        System.out.println("drinking milk");
}
public static void main(String args[ ])
{
Animal a1,a2,a3;
        a1=new Animal();
        a2=new Dog();
        a3=new BabyDog();
        a1.eat();
        a2.eat();
        a3.eat();
}}
```

## Abstract class

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

### Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery

There are two ways to achieve abstraction in java

- **Abstract class**
- **Interface**

## Abstract class

A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
abstract class A
{
}
```

## Abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

**Example abstract method**

```
abstract void printStatus(  );//no body and abstract
```

### Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike
{
  abstract void run( );
}
class Honda4 extends Bike
{
void run( )
{
        System.out.println("running safely..");
}
public static void main(String args[ ])
{
         Bike obj = new Honda4();
         obj.run();
}}
```

## Understanding the real scenario of abstract class

In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the

implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the factory method.

A factory method is the method that returns the instance of the class. We will learn about the factory method later.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

```
abstract class Shape
{
abstract void draw( );
}
class Rectangle extends Shape
{
void draw( )
{
        System.out.println("drawing rectangle");
} }
class Circle1 extends Shape
{
void draw( )
{
        System.out.println("drawing circle");}
}
//Method is called by programmer or user
class TestAbstraction1
{
public static void main(String args[ ])
{
Shape s=new Circle1();//In real scenario,object is provided through metho
d e.g. getShape() method
s.draw();
} }
```

**Abstract class having constructor, data member, methods etc.**

An abstract class can have data member, abstract method, method body, constructor and even main() method.

```
//example of abstract class that have method body
 abstract class Bike
{
  Bike( )
```

```
{
        System.out.println("bike is created");
}
  abstract void run();
  void changeGear( )
{
        System.out.println("gear changed");
} }
 class Honda extends Bike
{
 void run( )
{
        System.out.println("running safely..");
}  }
 class TestAbstraction2
{
 public static void main(String args[ ])
{
         Bike obj = new Honda();
         obj.run();
        obj.changeGear();
 } }
```

**Rule: If there is any abstract method in a class, that class must be abstract**.

```
class Bike12
{
        abstract void run();
}
```

## compile time error

**Rule:** If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

**Another real scenario of abstract class**

The abstract class can also be used to provide some implementation of the interface. In such case, the end user may not be forced to override all the methods of the interface.

Note: If you are beginner to java, learn interface first and skip this example.

```
interface A
{
        void a();
        void b();
        void c();
        void d();
}
 abstract class B implements A
{
        public void c( )
        {
                System.out.println("I am C");
        }
}
 class M extends B
{
public void a( )
        {
                System.out.println("I am a");
        }
        public void b( )
        {
                System.out.println("I am b");
        }

        public void d( )
        {
                System.out.println("I am d");
        }
        }
class Test5
{
public static void main(String args[ ])
{
A a=new M( );
```

```
        a.a( );
        a.b( );
        a.c( );
        a.d( );
}}
```

**PACKAGES AND INTERFACE**

A java package is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

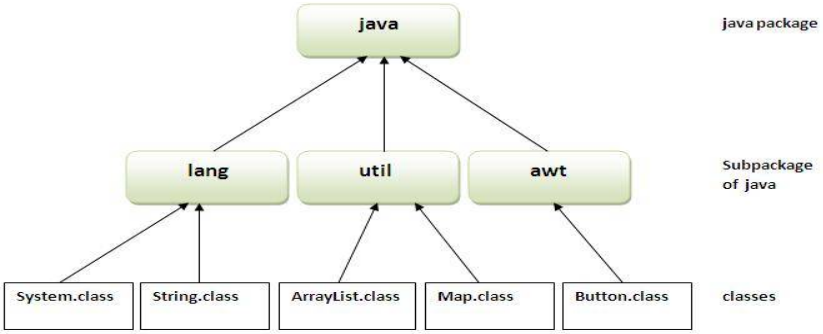
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

## Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.




**Simple example of package**

The package keyword is used to create a package in java.

//save as Simple.java

```
package mypack;
public class Simple
{
 public static void main(String args[ ])
{
   System.out.println("Welcome to package");
 }
}
```

**How to access package from another package**:

There are three ways to access the package from outside the package.

import package.*;

import package.classname;

fully qualified name.

- **Using packagename.***

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A
{
  public void msg( )
{
        System.out.println("Hello");
}
}
//save by B.java
package mypack;
import pack.*;
class B
{
  public static void main(String args[ ])
{
   A obj = new A( );
   obj.msg();
  }}
```

- **Using packagename.classname**

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java
package pack;
```

```
public class A
{
 public void msg( )
{
        System.out.println("Hello");
}}
//save by B.java
package mypack;
import pack.A;
class B
{
 public static void main(String args[ ])
{
  A obj = new A();
  obj.msg();
 }}
```

- **Using fully qualified name**

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

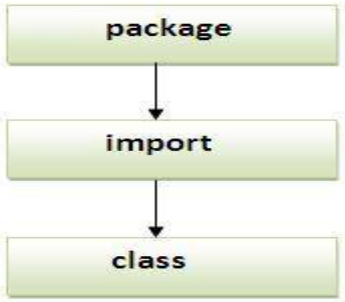
It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A
{
 public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
class B{
 public static void main(String args[]){
  pack.A obj = new pack.A();//using fully qualified name
  obj.msg();
 }
}
```

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Note: Sequence of the program must be package then import then class.




## Subpackage

Package inside the package is called the subpackage. It should be created to categorize the package further.

Let's take an example, Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

Example of Subpackage

```
package com.javatpoint.core;
class Simple
{
  public static void main(String args[ ])
{
   System.out.println("Hello subpackage");
  }
}
```

There are two ways to load the class files temporary and permanent.

**Temporary**

- By setting the classpath in the command prompt
- By -classpath switch

**Permanent**

- By setting the classpath in the environment variables
- By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.
-

```
//save as C.java otherwise Compilte Time Error
class A
{
}
class B
{
}
public class C
{
}
```

**How to put two public classes in a package**

If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same. For example:

```
//save as A.java
  package javatpoint;
public class A
{
}
//save as B.java
package javatpoint;
public class B
{
}
```

## Interface

The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

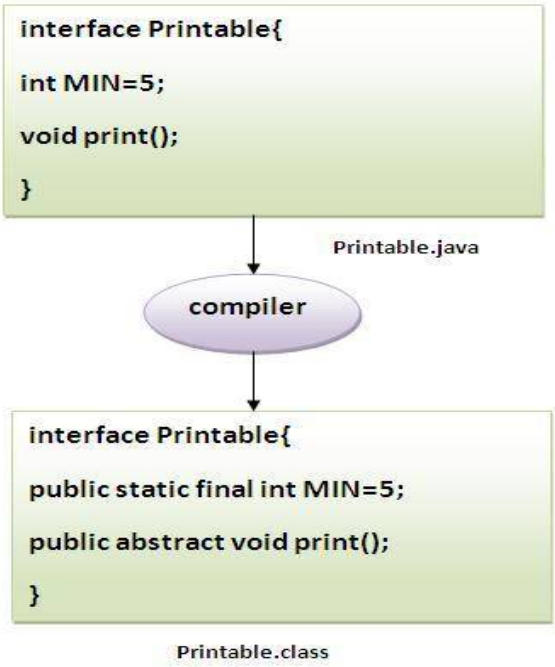
- Java Interface also represents IS-A relationship.
- It cannot be instantiated just like abstract class.

## Uses of Java interface

There are mainly three reasons to use interface. They are given below.

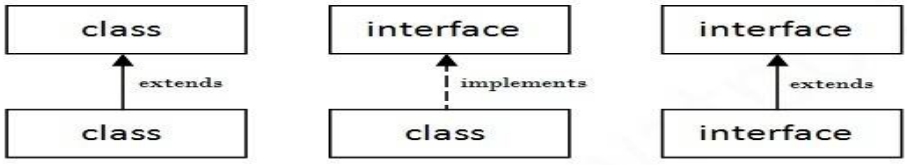
- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

In other words, Interface fields are public, static and final bydefault, and methods are public and abstract.




## Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface.




### Simple example of Java interface

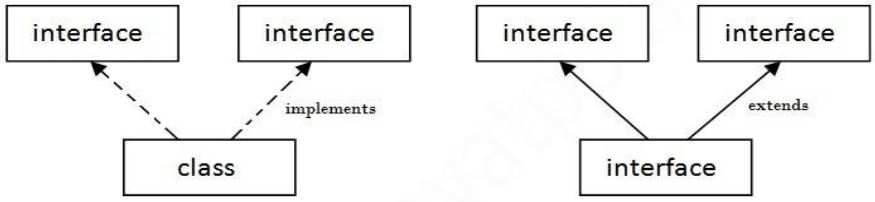
In this example, Printable interface have only one method, its

implementation is provided in the A class.

```
interface printable
{
void print( );
}
class A6 implements printable
{
public void print( )
{
System.out.println("Hello");
}
public static void main(String args[ ])
{
A6 obj = new A6();
obj.print();
 }
}
```

## Multiple inheritance Using interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.




**Multiple Inheritance in Java**

```
interface Printable
{
void print();
}
interface Showable
{
void show();
}
class A7 implements Printable,Showable
```

```
{
public void print( )
{
System.out.println("Hello");
}
public void show( )
{
System.out.println("Welcome");
}
public static void main(String args[ ])
{
A7 obj = new A7( );
obj.print( );
obj.show( );
 }
}
```

## Interface & inheritance

A class implements interface but one interface extends another interface .

```
interface Printable
{
void print( );
}
interface Showable extends Printable
{
void show();
}
class Testinterface2 implements Showable
{
public void print( )
{
System.out.println("Hello");
}
public void show( )
{
System.out.println("Welcome");
}
public static void main(String args[ ])
```

```
{
Testinterface2 obj = new Testinterface2();
obj.print();
obj.show();
 }
}
```

## Review & Self Assessment Question

1. What is class and object?
2. What is java naming convention?
3. What is package?
4. What is interface in java? And why we used.
5. Define abstract class.

## Further Readings

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

**IMPORTANT NOTES**

# UNIT-6 INTRODUCTION TO STRING,ARRAYAND VECTOR

## Contents



In java, string is basically an object that represents sequence of char values.

An array of characters works same as java string. For example:

```
char[ ] ch={'j','a','v','a','t','p','o','i','n','t'};
String s=new String(ch);
```

is same as:

```
String s="java";
```

The java.lang.String class
implements Serializable, Comparable and CharSequence interfaces.

The java String is immutable i.e. it cannot be changed but a new instance is created. For mutable class, you can use StringBuffer and StringBuilder class.

## String

String is a sequence of characters. But in java, string is an object that represents a sequence of characters. String class is used to create string object.

## How to create String object

There are two ways to create String object:

- By string literal
- By new keyword

1) **String Literal**

Java String literal is created by using double quotes. For Example:

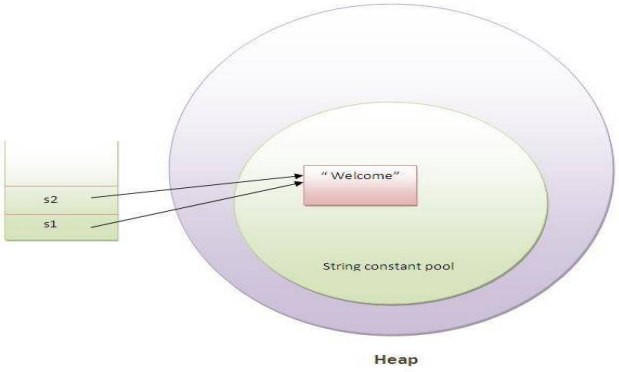
String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool.

**For example:**

String s1="Welcome";

String s2="Welcome";//will not create new instance




In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

**java uses concept of string literal:**

2) **By new keyword**

String s=new String("Welcome");//creates two objects and one reference v ariable

In such case, JVM will create a new string object in normal(non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap(non pool).

## Java String Example

```
public class StringExample
{
public static void main(String args[ ])
{
String s1="java";//creating string by java string literal
```

```
char ch[ ]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

## Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

| No. | Method | Description |
|---|---|---|
| 1 | java –string-charat | returns char value for the particular index |
| 2 | java-string –length | returns string length |
| 3 | java-string-format | returns formatted string |
| 4 | java-string-format | returns formatted string with given locale |
| 5 | java-string-substring | returns substring for given begin index |
| 6 | java-string-substring | returns substring for given begin index and end index |
| 7 | java-string-contains | returns true or false after matching the sequence of char value |
| 8 | java-string-join | returns a joined string |
| 9 | java-string-join | returns a joined string |
| 10 | java-string-equals | checks the equality of string with object |
| 11 | java-string-isempty | checks if string is empty |
| 12 | java-string-concat | concatinates specified string |

| | | |
|---|---|---|
| 13 | java-string-replace | replaces all occurrences of specified char value |
| 14 | java-string-replace | replaces all occurrences of specified CharSequence |
| 15 | java-string-trim | returns trimmed string omitting leading and trailing spaces |
| 16 | java-string-split | returns splitted string matching regex |
| 17 | java-string-split | returns splitted string matching regex and limit |
| 18 | java-string-intern | returns interned string |
| 19 | java-string-indexof | returns specified char value index |
| 20 | java-string-indexof | returns specified char value index starting with given index |
| 21 | java-string-indexof | returns specified substring index |
| 22 | java-string-indexof | returns specified substring index starting with given index |
| 23 | java-string-tolowercase | returns string in lowercase. |
| 24 | java-string-tolowercase | returns string in lowercase using specified locale. |
| 25 | java-string-touppercase | returns string in uppercase. |
| 26 | java-string-touppercase | returns string in uppercase using specified locale |

## String Concatenation in Java

In java, string concatenation forms a new string that is the combination of multiple strings. There are two ways to concat string in java:

- By + (string concatenation) operator
- By concat() method

1) **String Concatenation by + (string concatenation) operator**

Java string concatenation operator (+) is used to add strings.

**For Example:**

```
class TestStringConcatenation1
{
 public static void main(String args[ ])
{
   String s="Ramesh "+" Singh";
 System.out.println(s);//Ramesh  Singh
}}
```

## Substring in Java

A part of string is called substring. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.

You can get substring from the given string object by one of the two methods:

- **public String substring(int startIndex)**: This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
- **public String substring(int startIndex, int endIndex)**: This method returns new String object containing the substring of

### Example of java substring

```
public class TestSubstring
{
 public static void main(String args[ ])
{
  String s=" Ramesh  Tendulkar";
System.out.println(s.substring(6));//Tendulkar
 System.out.println(s.substring(0,6));//Ramesh
 }}
```

## Comparisons Of String :

We can compare string in java on the basis of content and reference.

It is used in authentication (by equals() method), sorting (by compareTo() method)etc.

There are three ways to compare string in java:

- By equals() Method
- By  compareTo() Method

**By equals() method**

1) String compare by equals ( ) method

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

- public boolean equals(Object another) compares this string to the specified object.
- public boolean equalsIgnoreCase(String another) compares this String to another string, ignoring case.

```
class Teststringcomparison1
{
 public static void main(String args[ ])
{
  String s1="Akhilesh ";
  String s2="Akhilesh";
  String s3=new String("Sachin");
  String s4="Furkan";
  System.out.println(s1.equals(s2));//true
  System.out.println(s1.equals(s3));//true
  System.out.println(s1.equals(s4));//false
 }}
class Teststringcomparison2
{
 public static void main(String args[ ])
{
  String s1="Akhilesh";
  String s2="AKHILESH";
  System.out.println(s1.equals(s2));//false
  System.out.println(s1.equalsIgnoreCase(s3));//true
}}
```

2) **String compare by compareTo( ) method**

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

s1 == s2 : returns zero

s1 > s2 : returns +v Integer

s1 < s2 : returns –v Integer

```
class Teststringcomparison4
{
 public static void main(String args[ ])
{
   String s1="Akhilesh";
   String s2="Akhilesh";
   String s3="Furkan";
   System.out.println(s1.compareTo(s2)); //0
   System.out.println(s1.compareTo(s3)); // -v Integer
   System.out.println(s3.compareTo(s1));//  +v Integer
 }}
```

## String concat( )

The java string concat( ) method combines specified string at the end of this string. It returns combined string. It is like appending another string.

**Signature**

The signature of string concat( ) method is given below:

```
public String concat(String anotherString)
```

**Parameter**

**a**notherString : another string i.e. to be combined at the end of this string.

**Returns**

combined string

**Java String concat( ) method example**

```
public class ConcatExample
{
public static void main(String args[ ])
{
String s1=" AKHILESH";
s1.concat(" FURKAN ");
System.out.println(s1);
s1=s1.concat(" is immutable so assign it explicitly");
System.out.println(s1);
}}
```

## String contains( )

The java string contains( ) method searches the sequence of characters in this string. It returns true if sequence of char values are found in this string otherwise returns false.

## Signature

The signature of string contains() method is given below:

public boolean contains(CharSequence sequence)

**Parameter**

sequence : specifies the sequence of characters to be searched.

**Returns**

true if sequence of char value exists, otherwise false.

**Throws**

NullPointerException : if sequence is null.

**Java String contains( ) method example**

```
class ContainsExample
{
public static void main(String args[ ])
{
String name="what do you know about me";
System.out.println(name.contains("do you know"));
System.out.println(name.contains("about"));
System.out.println(name.contains("hello"));
}}
```

## String length

The java string length() method length of the string. It returns count of total number of characters. The length of java string is same as the unicode code units of the string.

**Signature**

The signature of the string length( ) method is given below:

public int length()

**Specified by**

CharSequence interface

**Returns**

length of characters

**Java String length() method example:**

```
public class LengthExample
{
public static void main(String args[ ])
{
String s1="javatpoint";
String s2="python";
```

```
System.out.println("string length is: "+s1.length());//10 is the length of jav
atpoint string
System.out.println("string length is: "+s2.length());//6 is the length of pyth
on string
}}
```

## Java String indexOf

The java string indexOf( ) method returns index of given character value . If it is not found, it returns -1. The index counter starts from zero.

**Signature**

There are 2 types of indexOf method in java. The signature of indexOf methods are given below:

| No. | Method | Description |
|---|---|---|
| 1 | int indexOf(int ch) | returns index position for the given char value |
| 2 | int indexOf(int ch, int fromIndex) | returns index position for the given char value and from index |

## Parameters

ch: char value i.e. a single character e.g. 'a'

fromIndex: index position from where index of the char value or substring is retured substring: substring to be searched in this string

**Returns**

index of the string

**Java String indexOf( ) method example**

```
public class IndexOfExample
{
public static void main(String args[ ])
{
String s1="this is index of example";
//passing substring
int index1=s1.indexOf("is");//returns the index of is substring
int index2=s1.indexOf("index");//returns the index of index substring
System.out.println(index1+"  "+index2);//2 8
//passing substring with from index
int index3=s1.indexOf("is",4);//returns the index of is substring after 4th in
dex
System.out.println(index3);//5 i.e. the index of another is
```

```
//passing char value
int index4=s1.indexOf('s');//returns the index of s char value
System.out.println(index4);//3
}}
```

## Java String charAt()

The java string charAt() method returns a char value at the given index number. The index number starts from 0.

**Signature**

The signature of string charAt() method is given below:

public char charAt(int index)

**Parameter**

index : index number, starts with 0

**Returns**

char value

**Specified by**

CharSequence interface

**Throws**

IndexOutOfBoundsException : if index is negative value or greater than this string length.

### Java String charAt( ) method example

```
public class CharAtExample
{
public static void main(String args[ ])
{
String name="javaprint";
char ch=name.charAt(4);//returns the char value at the 4th index
System.out.println(ch);
}}
```

## Java String equals

The java string equals() method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

The String equals() method overrides the equals() method of Object class.

**Signature**

public boolean equals(Object anotherObject)

**Parameter**

anotherObject : another object i.e. compared with this string.

**Returns**

true if characters of both strings are equal otherwise false.

**Overrides**

equals() method of java Object class.

### Java String equals() method example

```
public class EqualsExample
{
public static void main(String args[])
{
String s1="javap";
String s2="javap";
String s3="JAVAP";
String s4="python";
System.out.println(s1.equals(s2));//true because content and case is same
System.out.println(s1.equals(s3));//false because case is not same
System.out.println(s1.equals(s4));//false because content is not same
}}
```

## Java StringBuffer class

Java StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

Important Constructors of StringBuffer class

**StringBuffer( )**: creates an empty string buffer with the initial capacity of 16.

**StringBuffer(String str)**: creates a string buffer with the specified string.

**StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

**Important methods of StringBuffer class**

**public synchronized StringBuffer append(String s):** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

**public synchronized StringBuffer insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

**public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.

**public synchronized StringBuffer delete(int startIndex, int endIndex):** is used to delete the string from specified startIndex and endIndex.

**public synchronized StringBuffer reverse():** is used to reverse the string.

**public int capacity():** is used to return the current capacity.

**public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum.

**public char charAt(int index):** is used to return the character at the specified position.

**public int length():** is used to return the length of the string i.e. total number of characters.

**public String substring(int beginIndex**): is used to return the substring from the specified beginIndex.

**public String substring(int beginIndex, int endIndex**): is used to return the substring from the specified beginIndex and endIndex.

## Mutable string

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

**1) StringBuffer append() method**

The append( ) method concatenates the given argument with this string.

```
class A
{
public static void main(String args[ ])
{
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
} }
```

**2) StringBuffer insert() method**

The insert() method inserts the given string with this string at the given position.

```
class A
{
```

```
public static void main(String args[ ])
{
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
} }
```

**3) StringBuffer replace( ) method**

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class A
{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}}
```

**4) StringBuffer delete() method**

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class A
{
public static void main(String args[ ])\
{
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}}
```

**5) StringBuffer reverse( ) method**

The reverse() method of StringBuilder class reverses the current string.

```
class A
{
public static void main(String args[ ])
{
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
} }
```

**6) StringBuffer capacity( ) method**

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2.

For example if your current capacity is 16, it will be (16*2)+2=34.

```
class A
{
public static void main(String args[ ])
{
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34
}}
```

**7) StringBuffer ensureCapacity( ) method**

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

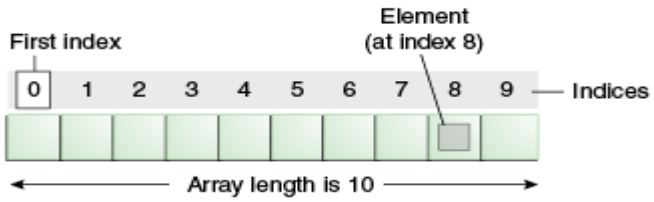
```
class A
{
public static void main(String args[ ])
{
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
} }
```

## Array

Array is a collection of similar type of elements that have contiguous memory location.

Java array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.




## Advantage of Array

- Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.
- Random access: We can get any data located at any index position.

## Disadvantage of Array

- Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

## Types of Array

There are two types of array.

- **Single Dimensional Array**
- **Multidimensional Array**

**Single Dimensional Array**

Syntax to Declare an Array in java

dataType[ ] arr; (or)

dataType [ ]arr; (or)

dataType arr[ ];

**Instantiation of an Array**

arrayRefVar=new datatype[size];

**Example of single dimensional array**

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
class Testarray
{
public static void main(String args[ ])
```

```
{
int a[ ]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
  //printing array
for(int i=0;i<a.length;i++)//length is the property of array
        System.out.println(a[i]);
}}
```

## Declaration, Instantiation and Initialization Array

We can declare, instantiate and initialize the java array together by:

```
int a[ ]={33,3,4,5};//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
class Testarray1
{
public static void main(String args[ ])
{
int a[]={33,3,4,5};//declaration, instantiation and initialization

//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

## Passing Array to method

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get minimum number of an array using method.

```
class Testarray2
{
static void min(int arr[ ])
{
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min>arr[i])
```

```
min=arr[i];
System.out.println(min);
}
public static void main(String args[ ])
{
int a[]={33,3,4,5};
min(a);//passing array to method
}}
```

## Multidimensional array

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

## Example to instantiate Multidimensional Array

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

## Example of Multidimensional Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
class Testarray3
{
public static void main(String args[ ])
{
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
```

```
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
  System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}}}
```

## Copying a java array

We can copy an array to another by the arraycopy method of System class.

**Syntax of arraycopy method**

```
public static void arraycopy(  Object src, int srcPos,Object dest, int destPos
, int length  )
```

**Example of arraycopy method**

```
class TestArrayCopyDemo
{
   public static void main(String[ ] args)
{
      char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
            'i', 'n', 'a', 't', 'e', 'd' };
      char[] copyTo = new char[7];
      System.arraycopy(copyFrom, 2, copyTo, 0, 7);
      System.out.println(new String(copyTo));
   }}
```

## Addition of 2 matrices

Let's see a simple example that adds two matrices.

```
class Testarray5
{
public static void main(String args[])
{
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};
//creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];
//adding and printing addition of 2 matrices
for(int i=0;i<2;i++)
{
```

```
for(int j=0;j<3;j++)
{
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}}}
```

## Wrapper class

Wrapper class in java provides the mechanism to convert primitive into object and object into primitive.

Since J2SE 5.0, autoboxing and unboxing feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known and autoboxing and vice-versa unboxing.

One of the eight classes of java.lang package are known as wrapper class in java. The list of eight wrapper classes are given below:

| Primitive Data Type | Wrapper class |
|---|---|
| Boolean | Boolean |
| Char | Character |
| Byte | Byte |
| Short | Short |
| Int | Integer |
| Long | Long |
| Float | Float |
| Double | Double |

**Wrapper class Example**: Primitive to Wrapper

```
public class WrapperExample1
{
public static void main(String args[ ])
{
//Converting int into Integer
int a=20;
```

```
Integer i=Integer.valueOf(a);//converting int into Integer
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) intern
ally
System.out.println(a+" "+i+" "+j);
}}
```

**Wrapper class Example**: Wrapper to Primitive

```
public class WrapperExample2
{
public static void main(String args[ ])
{
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int
int j=a;//unboxing, now compiler will write a.intValue() internally
System.out.println(a+" "+i+" "+j);
}}
```

## Review & Self Assessment Question

1. What are the uses of new keyword in string?
2. What is substring in java?
3. Why we use concatenate method ( ).
4. What is mutable string .
5. What is wrapper class.

## Further Readings

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

# UNIT-7 EXCEPTION HANDLING

## Contents

- Exception

  Exception Handling
- Hierarchy of Java Exception Classes
- Types of Exception
- Try and Catch
- Throw
- Throws
- Finally
- Review & self assessment Question
- Further Readings

## Exception

Exception is an abnormal condition. In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

## Exception handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

**Advantage of Exception Handling**

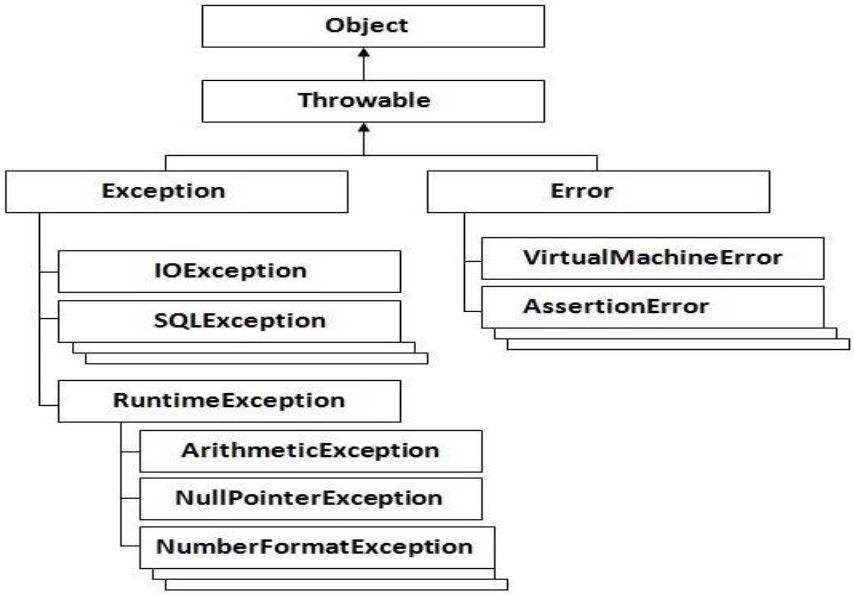
The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling.

For example :

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
```

Suppose there is 7 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 7 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

## Hierarchy of Java Exception classes




## Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

- Checked Exception
- Unchecked Exception
- Error

### 1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

**3) Error**

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

**Common scenarios where exceptions may occur**

There are given some scenarios where unchecked exceptions can occur. They are as follows:

**1) ArithmeticException occurs**

If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

**2) NullPointerException occurs**

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

**3)NumberFormatException occurs**

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

**4) ArrayIndexOutOfBoundsException occurs**

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

### Exception Handling Keywords

There are 5 keywords used in java exception handling.

- try
- catch
- finally
- throw
- throws

## Try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.Java try block must be followed by either catch or finally block.

**Syntax of java try-catch:**

```
try
{
//code that may throw exception
}catch(Exception_class_Name ref)
{       }
```

Syntax of try-finally block

```
try
{
//code that may throw exception
}
finally
{ }
```

## Catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

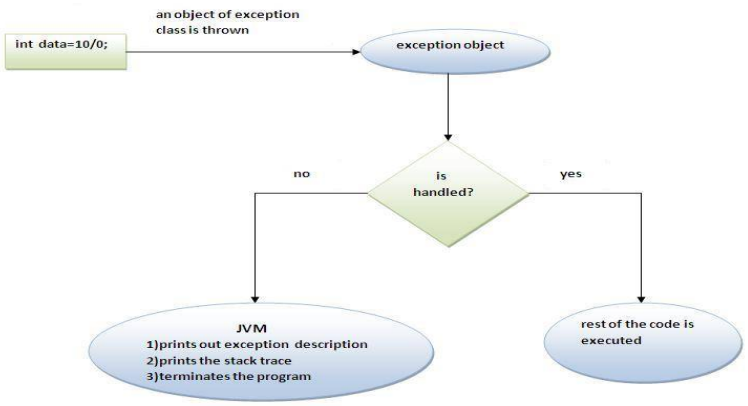
You can use multiple catch block with a single try.

**Problem without exception handling**

Let's try to understand the problem if we don't use try-catch block.

```
public class Testtrycatch1
{
public static void main(String args[ ])
{
  int data=50/0;//may throw exception
  System.out.println("rest of the code...");
        }}
```

**Internal working of java try-catch block**



The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

- Prints out exception description.
- Prints the stack trace (Hierarchy of methods where the exception occurred).
- Causes the program to terminate.

But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

## Multi catch block

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

**Example of java multi-catch block**

```
public class TestMultipleCatchBlock
{
  public static void main(String args[ ])
{
   try
{
          int a[ ]=new int[5];
             a[5]=30/0;
   }
   catch(ArithmeticException e)
{
          System.out.println("task1 is completed");
}
catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 co
mpleted");
}
 catch(Exception e){System.out.println("common task completed");}

          System.out.println("rest of the code...");
 }
}
```

**Rule: At a time only one Exception is occured and at a time only one catch block is executed.**

**Rule: All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception .**

```
class TestMultipleCatchBlock1
{
 public static void main(String args[])
{
  try{
   int a[]=new int[5];
   a[5]=30/0;
  }
  catch(Exception e)
{
        System.out.println("common task completed");
}
  catch(ArithmeticException e)
{
        System.out.println("task1 is completed");
}
  catch(ArrayIndexOutOfBoundsException e)
{
        System.out.println("task 2 completed");
}
          System.out.println("rest of the code...");
 }
}
```

## Throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. The syntax of java throw keyword is given below.

**throw exception;**

```
        throw new IOException("sorry device error);
```

**Throw keyword example**

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
public class TestThrow1
{
        static void validate(int age)
{
    if(age<18)
     throw new ArithmeticException("not valid");
    else
        System.out.println("welcome to vote");
  }
  public static void main(String args[ ])
{
     validate(13);
     System.out.println("rest of the code...");
 } }
```

## Throws keyword

The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

```
return_type method_name( ) throws exception_class_name
{
//method code
}
```

**Advantage of Java throws keyword**

Now Checked Exception can be propagated (forwarded in call stack).

- It provides information to the caller of the method about the exception.

**Example:**

Example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

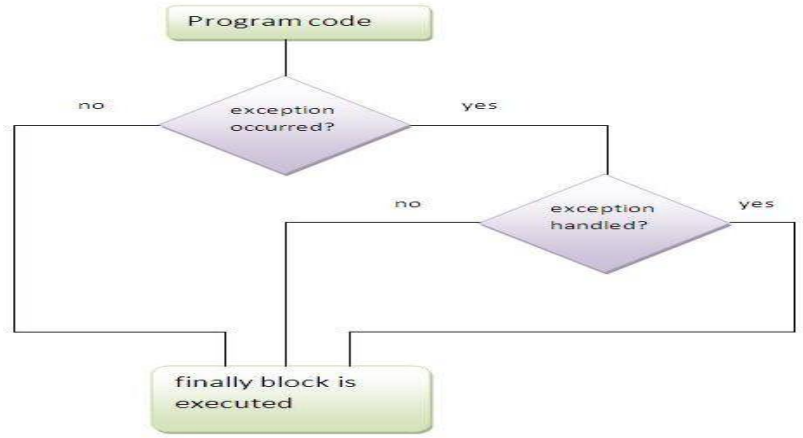
```
import java.io.IOException;
class Testthrows1
{
  void m( )throws IOException
{
    throw new IOException("device error");//checked exception
  }
  void n( )throws IOException
{
    m( );
  }
  void p( )
{
   try{
    n( );
   }catch(Exception e)
{
System.out.println("exception handled");
}
  }
  public static void main(String args[ ])
{
   Testthrows1 obj=new Testthrows1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

**Finally block**

Finally block is a block that is used to execute important code such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block must be followed by try or catch block.




**Usage of Java finally**

The different cases where java finally block can be used.

**Case:1**

The java finally example where exception doesn't occur.

```
class TestFinallyBlock
{
  public static void main(String args[ ])
{
  try
{
   int data=25/5;
          System.out.println(data);
  }
  catch(NullPointerException e)
{
        System.out.println(e);
}
  Finally
{
        System.out.println("finally block is always executed");
}
        System.out.println("rest of the code...");
  }  }
```

**Case:2**

The java finally example where exception occurs and not handled.

```
class TestFinallyBlock1
{
  public static void main(String args[ ])
{
 Try
{
int data=25/0;
 System.out.println(data);
 }
  catch(NullPointerException e)
{
System.out.println(e);
}
 finally
{
        System.out.println("finally block is always executed");}
         System.out.println("rest of the code...");
 } }
```

**Case:3**

The java finally example where exception occurs and handled.

```
public class TestFinallyBlock2
{
  public static void main(String args[ ])
{
  Try
{
  int data=25/0;
          System.out.println(data);
  }
catch(ArithmeticException e)
{
        System.out.println(e);
}
```

```
Finally
        {
        System.out.println("finally block is always executed");}
          System.out.println("rest of the code...");
         }  }
```

## Review & Self Assessment Question

1. What is exception handling?
2. What are the advantage of exception handling in java.
3. Define throws Keyword?
4. What are the hierarchies of Java exception classes.
5. What are the advantage of java throw keyword?

## Further Readings

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

# UNIT- 8 MULTI THREADING

## Contents

- ❖ Multithreading
- ❖ Thread
- ❖ Multitasking
- ❖ Life cycle of Thread
- ❖ Thread Scheduler in java
- ❖ Naming a thread
- ❖ Review & self assessment Question
- ❖ Further Readings

Multithreading in java is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

## Advantage of Java Multithreading

- It doesn't block the user because threads are independent and you can perform multiple operations at same time.
- 2) You can perform many operations together so it saves time.
- 3) Threads are independent so it doesn't affect other threads if exception occur in a single thread.

**Multitasking**

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

- Process-based Multitasking(Multiprocessing)
- Thread-based Multitasking(Multithreading)

**1) Process-based Multitasking (Multiprocessing)**

Each process have its own address in memory i.e. each process allocates separate memory area.

- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

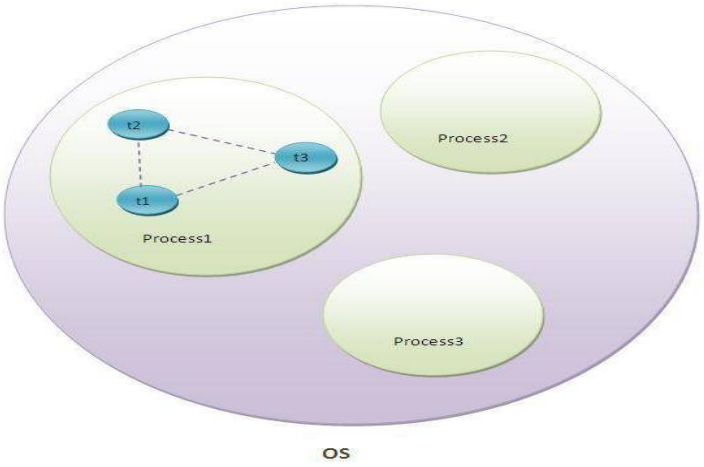
**2) Thread-based Multitasking (Multithreading)**

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.

## Thread in java

A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.

Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.




As shown in the above figure, thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.

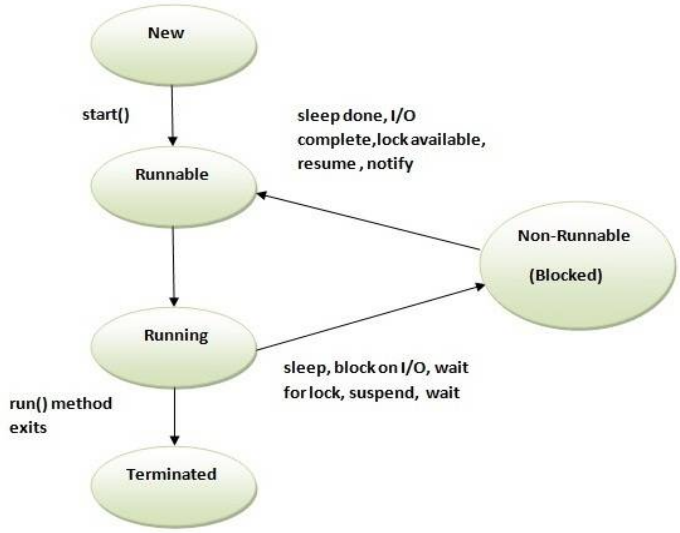
## Life cycle of a Thread

A thread can be in one of the five states. According to sun, there is only 4 states in thread life cycle in java new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

- New
- Runnable
- Running
- Non-Runnable (Blocked)
- Terminated




1) **New**

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

**2) Runnable**

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

**3) Running**

The thread is in running state if the thread scheduler has selected it.

**4) Non-Runnable (Blocked)**

This is the state when the thread is still alive, but is currently not eligible to run.

**5) Terminated**

A thread is in terminated or dead state when its run() method exits.

There are two ways to create a thread:

- By extending Thread class
- By implementing Runnable interface.

## Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

**Commonly used Constructors of Thread class:**

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

**Commonly used methods of Thread class:**

- public void run(): is used to perform action for a thread.
- public void start(): starts the execution of the thread.JVM calls the run() method on the thread.
- public void sleep(long miliseconds): Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
- public void join(): waits for a thread to die.
- public void join(long miliseconds): waits for a thread to die for the specified miliseconds.
- public int getPriority(): returns the priority of the thread.
- public int setPriority(int priority): changes the priority of the thread.
- public String getName(): returns the name of the thread.
- public void setName(String name): changes the name of the thread.
- public Thread currentThread(): returns the reference of currently executing thread.
- public int getId(): returns the id of the thread.
- public Thread.State getState(): returns the state of the thread.
- public boolean isAlive(): tests if the thread is alive.
- public void yield(): causes the currently executing thread object to temporarily pause and allow other threads to execute.
- public void suspend(): is used to suspend the thread(depricated).
- public void resume(): is used to resume the suspended thread(depricated).
- public void stop(): is used to stop the thread(depricated).

- public boolean isDaemon(): tests if the thread is a daemon thread.
- public void setDaemon(boolean b): marks the thread as daemon or user thread.
- public void interrupt(): interrupts the thread.
- public boolean isInterrupted(): tests if the thread has been interrupted.
- public static boolean interrupted(): tests if the current thread has been interrupted.

**Runnable interface:**

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

public void run(): is used to perform action for a thread.

**Starting a thread:**

start( ) method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

**1)By extending Thread class**:

```
class Multi extends Thread
{
public void run( )
{
System.out.println("thread is running...");
}
public static void main(String args[ ])
{
Multi t1=new Multi();
t1.start();
 }}
```

Thread class constructor allocates a new thread object.When you create object of Multi class,your class constructor is invoked(provided by Compiler) fromwhere Thread class constructor is invoked(by super() as

first statement).So your Multi class object is thread object now.

**2)By implementing the Runnable interface**:

```
class Multi3 implements Runnable
{
public void run( )
{
System.out.println("thread is running...");
}
public static void main(String args[ ])
{
Multi3 m1=new Multi3( );
Thread t1 =new Thread(m1);
t1.start( );
 }}
```

## Thread Scheduler in Java

Thread scheduler in java is the part of the JVM that decides which thread should run.

There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.

Only one thread at a time can run in a single process.

The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

## Difference between preemptive scheduling and time slicing

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

## Sleep method in java

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

Syntax : sleep() method in java

The Thread class provides two methods for sleeping a thread:

- public static void sleep(long miliseconds)throws InterruptedException

- public static void sleep(long miliseconds, int nanos)throws InterruptedException

**Example**

```
class TestSleepMethod1 extends Thread
{
 public void run( )
{
  for(int i=1;i<5;i++)
{
    try
{
          Thread.sleep(500);
}catch(InterruptedException e)
{
System.out.println(e);
}
   System.out.println(i);
  } }
 public static void main(String args[ ])
{
          TestSleepMethod1 t1=new TestSleepMethod1();
          TestSleepMethod1 t2=new TestSleepMethod1();
  t1.start();
  t2.start();
 }}
```

Note : if we call run() method directly instead start() method

Each thread starts in a separate call stack.

Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

```
class TestCallRun1 extends Thread
{
 public void run( )
{
   System.out.println("running...");
 }
 public static void main(String args[ ])
```

```
{
 TestCallRun1 t1=new TestCallRun1();
 t1.run();//fine, but does not start a separate call stack
 }}
```

## The join() method:

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

Syntax:

public void join()throws InterruptedException

public void join(long milliseconds)throws InterruptedException

**Example of join() method**

```
class TestJoinMethod1 extends Thread
{
 public void run( )
{
  for(int i=1;i<=5;i++)
{
   try
{
    Thread.sleep(500);
   }catch(Exception e)
{
          System.out.println(e);
}
           System.out.println(i);
  } }
public static void main(String args[ ])
{
 TestJoinMethod1 t1=new TestJoinMethod1();
 TestJoinMethod1 t2=new TestJoinMethod1();
 TestJoinMethod1 t3=new TestJoinMethod1();
 t1.start();
 try
{
  t1.join( );
```

```
 }catch(Exception e)
{
System.out.println(e);
}
 t2.start();
 t3.start();
}}
```

## getName(),setName(String) and getId() method:

- public String getName()
- public void setName(String name)
- public long getId( )

```
class TestJoinMethod3 extends Thread
{
  public void run()
{
   System.out.println("running...");
  }
 public static void main(String args[ ])
{
  TestJoinMethod3 t1=new TestJoinMethod3();
  TestJoinMethod3 t2=new TestJoinMethod3();
  System.out.println("Name of t1:"+t1.getName());
  System.out.println("Name of t2:"+t2.getName());
  System.out.println("id of t1:"+t1.getId());
  t1.start();
  t2.start();
  t1.setName("Sonoo Jaiswal");
  System.out.println("After changing name of t1:"+t1.getName());
 } }
```

## The currentThread() method:

The currentThread() method returns a reference to the currently executing thread object.

Syntax:

```
public static Thread currentThread()
```

## Example of currentThread() method

```
class TestJoinMethod4 extends Thread
{
 public void run( )
{
  System.out.println(Thread.currentThread().getName());
 }  }
 public static void main(String args[ ])
{
  TestJoinMethod4 t1=new TestJoinMethod4( );
  TestJoinMethod4 t2=new TestJoinMethod4( );
  t1.start();
  t2.start();
 } }
```

## Naming a thread:

The Thread class provides methods to change and get the name of a thread.

- public String getName(): is used to return the name of a thread.
- public void setName(String name): is used to change the name of a thread.

**Example of naming a thread:**

```
class TestMultiNaming1 extends Thread
{
  public void run( )
{
   System.out.println("running...");
  }
 public static void main(String args[ ])
{
  TestMultiNaming1 t1=new TestMultiNaming1( );
  TestMultiNaming1 t2=new TestMultiNaming1( );
  System.out.println("Name of t1:"+t1.getName( ));
  System.out.println("Name of t2:"+t2.getName( ));
  t1.start( );
  t2.start( );
  t1.setName("Sonoo ");
  System.out.println("After changing name of t1:"+t1.getName());
```

```
 }
}
```

**The currentThread() method:**

The currentThread() method returns a reference to the currently executing thread object.

Syntax of currentThread() method:

public static Thread currentThread(): returns the reference of currently running thread.

**Example of currentThread() method:**

```
class TestMultiNaming2 extends Thread
{
 public void run( )
{
  System.out.println(Thread.currentThread().getName());
 }
 }
 public static void main(String args[ ])
{
  TestMultiNaming2 t1=new TestMultiNaming2( );
  TestMultiNaming2 t2=new TestMultiNaming2( );
  t1.start();
  t2.start();
 }  }
```

**Priority of a Thread (Thread Priority):**

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

**3 constants defiend in Thread class**:

- public static int MIN_PRIORITY
- public static int NORM_PRIORITY
- public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

## Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread
{
 public void run( )
{
   System.out.println("running thread name is:"+Thread.currentThread(
).getName( ));
   System.out.println("running thread priority is:"+Thread.currentThread(
).getPriority( ));
  }
 public static void main(String args[ ])
{
  TestMultiPriority1 m1=new TestMultiPriority1();
  TestMultiPriority1 m2=new TestMultiPriority1();
  m1.setPriority(Thread.MIN_PRIORITY);
  m2.setPriority(Thread.MAX_PRIORITY);
  m1.start();
  m2.start();
  } }
```

## Review & Self Assessment Question

1. Define the term thread.
2. What is multithreading?
3. Explain life cycle of thread?
4. What is preemptive scheduling in java?
5. What is thread scheduler in java?

## Further Readings

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

# UNIT-9 APPLET

## Contents

- Applet
- Hierarchy of Applet
- Life cycle of Applet
- Displaying Graphics in Applet
- AWT
- Event Handling
- Java AWT Hierarchy
- Review & self assessment Question
- Further Readings

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

**Advantage of Applet**

There are many advantages of applet.

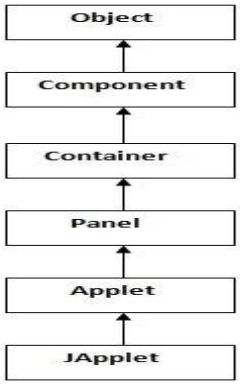
They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

**Drawback of Applet**

Plugin is required at client browser to execute applet.

## Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

**Lifecycle of Java Applet**

- Applet is initialized.
- Applet is started.
- Applet is painted.
- Applet is stopped.
- Applet is destroyed.

**Lifecycle methods for Applet**:

- The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

**java.applet.Applet class**

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

- public void init(): is used to initialized the Applet. It is invoked only once.
- public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
- public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
- public void destroy(): is used to destroy the Applet. It is invoked only once.

**java.awt.Component class**

The Component class provides 1 life cycle method of applet.

- public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

There are two ways to run an applet

- By html file.
- By appletViewer tool (for testing purpose).

**Example of Applet by html file**:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g)
{
g.drawString("welcome",150,150);
}
}
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

File : applet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

**Simple example of Applet by appletviewer tool:**

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g)
{
g.drawString("welcome to applet",150,150);
} }
/* <applet code="First.class" width="300" height="300"> </applet> */
```

**Displaying Graphics in Applet**

java.awt.Graphics class provides many methods for graphics programming.

## Commonly used methods of Graphics class:

- public abstract void drawString(String str, int x, int y): is used to draw the specified string.
- public void drawRect(int x, int y, int width, int height): draws a rectangle with the specified width and height.
- public abstract void fillRect(int x, int y, int width, int height): is used to fill rectangle with the default color and specified width and height.
- public abstract void drawOval(int x, int y, int width, int height): is used to draw oval with the specified width and height.
- public abstract void fillOval(int x, int y, int width, int height): is used to fill oval with the default color and specified width and height.
- public abstract void drawLine(int x1, int y1, int x2, int y2): is used to draw line between the points(x1, y1) and (x2, y2).
- public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.
- public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used draw a circular or elliptical arc.
- public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used to fill a circular or elliptical arc.
- public abstract void setColor(Color c): is used to set the graphics current color to the specified color.
- public abstract void setFont(Font font): is used to set the graphics current font to the specified font.

## Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo extends Applet
{
public void paint(Graphics g)
{
        g.setColor(Color.red);
        g.drawString("Welcome",50, 50);
        g.drawLine(20,30,20,300);
        g.drawRect(70,100,30,30);
        g.fillRect(170,100,30,30);
```

```
        g.drawOval(70,200,30,30);
        g.setColor(Color.pink);
        g.fillOval(170,200,30,30);
        g.drawArc(90,150,30,30,30,270);
        g.fillArc(270,150,30,30,0,180);
 }}
```

File : applet.html

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

**Displaying Image in Applet**

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

**Syntax of drawImage() method:**

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

**Get the object of Image:**

The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

```
public Image getImage(URL u, String image)
{
}
```

Other required methods of Applet class to display image:

- public URL getDocumentBase(): is used to return the URL of the document in which applet is embedded.
- public URL getCodeBase(): is used to return the base URL.

**Example of displaying image in applet:**

```
import java.awt.*;
import java.applet.*;
public class DisplayImage extends Applet
 {
  Image picture;
```

```
 public void init( )
{
   picture = getImage(getDocumentBase( ),"so.jpg");
 }
 public void paint(Graphics g)
{
   g.drawImage(picture, 30,30, this);
 }}
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

File : applet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

## EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

**Example of EventHandling in applet**:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener
{
Button b;
TextField tf;
public void init( )
{
tf=new TextField();
tf.setBounds(30,40,150,20);
 b=new Button("Click");
```

```
b.setBounds(80,150,60,50);
add(b);add(tf);
b.addActionListener(this);
setLayout(null);
}
 public void actionPerformed(ActionEvent e)
{
  tf.setText("Welcome");
 } }
```

In the above example, we have created all the controls in init() method because it is invoked only once.

File : applet.html

```
<html>
<body>
<applet code="EventApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

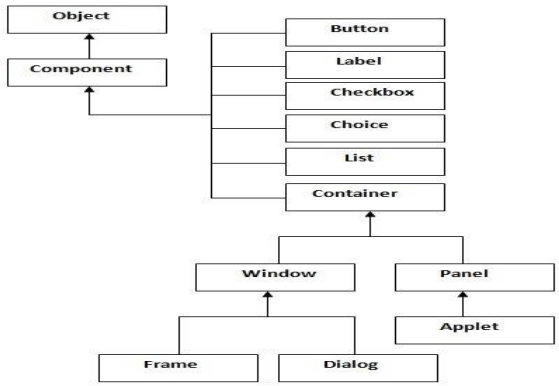
## AWT And Event Handling

Java AWT (Abstract Windowing Toolkit) is an API to develop GUI or window-based application in java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components uses the resources of system.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

**Java AWT Hierarchy**

The hierarchy of Java AWT classes are given below.



**Container**

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

**Window**

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

**Panel**

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

**Frame**

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

**Useful Methods of Component class**

| Method | Description |
|---|---|
| public void add(Component c) | inserts a component on this component. |
| public void setSize(int width,int height) | sets the size (width and height) of the component. |
| public void setLayout(LayoutManager m) | defines the layout manager for the component. |
| public void setVisible(boolean status) | changes the visibility of the component, by default false. |

**There are two ways to create a frame in AWT.**

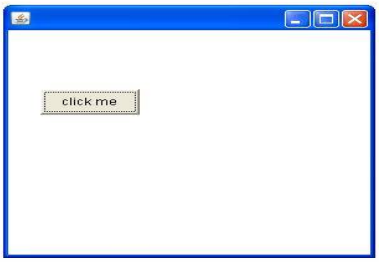
- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

Simple example of AWT by inheritance

```
import java.awt.*;
class First extends Frame
{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
```

```
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[])
{
First f=new First();
}
}
```

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.




**Simple example of AWT by association**

```
import java.awt.*;
class First2
{
First2( )
{
Frame f=new Frame( );
Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[ ])
{
First2 f=new First2();
}
}
```

## Java Event Handling

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

**Event classes and Listener interfaces:**

| Event Classes | Listener Interfaces |
|---|---|
| ActionEvent | ActionListener |
| MouseEvent | MouseListener and MouseMotionListener |
| MouseWheelEvent | MouseWheelListener |
| KeyEvent | KeyListener |
| ItemEvent | ItemListener |
| TextEvent | TextListener |
| AdjustmentEvent | AdjustmentListener |
| WindowEvent | WindowListener |
| ComponentEvent | ComponentListener |
| ContainerEvent | ContainerListener |
| FocusEvent | FocusListener |

**Following steps are required to perform event handling:**

- Implement the Listener interface and overrides its methods
- Register the component with the Listener

For registering the component with the Listener, many classes provide the registration methods. For example:

Button

```
public void addActionListener(ActionListener a)
{
}
```

MenuItem

```
public void addActionListener(ActionListener a)
{
}
```

TextField

```
public void addActionListener(ActionListener a)
{
}
public void addTextListener(TextListener a)
{                }
```

TextArea

```
public void addTextListener(TextListener a)
{
}
```

Checkbox

```
public void addItemListener(ItemListener a)
{
}
```

Choice

```
public void addItemListener(ItemListener a)
{
}
```

List

```
public void addActionListener(ActionListener a)
{
}
public void addItemListener(ItemListener a)
{
}
```

## EventHandling Codes:

We can put the event handling code into one of the following places:

- Same class
- Other class
- Annonymous class

## Example of event handling within class:

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener
{
```

```
TextField tf;
AEvent( )
{
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
b.addActionListener(this);
add(b);
add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
tf.setText("Welcome");
}
public static void main(String args[ ])
{
new AEvent();
}
}
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.


**2) Example of event handling by Outer class:**

```
import java.awt.*;
import java.awt.event.*;
```

```
class AEvent2 extends Frame
{
TextField tf;
AEvent2( )
{
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
Outer o=new Outer(this);
b.addActionListener(o);//passing outer class instance
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[ ])
{
new AEvent2();
} }
import java.awt.event.*;
class Outer implements ActionListener
{
AEvent2 obj;
Outer(AEvent2 obj)
{
this.obj=obj;
}
public void actionPerformed(ActionEvent e)
{
obj.tf.setText("welcome");
}
}
```

**3) Example of event handling by Annonymous class:**

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame
```

```
{
TextField tf;
AEvent3( )
{
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);
b.addActionListener(new ActionListener( )
{
public void actionPerformed( )
{
tf.setText("hello");
}
};
add(b);
add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[ ])
{
new AEvent3();
} }
```

## Review & Self Assessment Question

1. What is Applet?
2. Define life cycle of Applet?
3. Describe hierarchy of Applet?
4. What is AWT?
5. What is Java Event Handling?
6. What is Java AWT hierarchy.

## Further Readings

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

# UNIT-10 SWING

## Contents

- Introduction to swing
- Difference between awt and swing
- Java foundation classes
- Hierarchy of java swing class
- Review & self assessment Question
- Further Readings

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

like AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

**Difference between AWT and Swing**

There are many differences between java awt and swing that are given below.

| No. | Java AWT | Java Swing |
|---|---|---|
| 1) | AWT components are platform-dependent. | Java swing components are platform-independent. |
| 2) | AWT components are heavyweight. | Swing components are lightweight. |
| 3) | AWT doesn't support pluggable look and feel. | Swing supports pluggable look and feel. |
| 4) | AWT provides less components than Swing. | Swing provides more powerful componentssuch as tables, lists, scrollpanes,colorchooser, tabbedpane etc. |
| 5) | AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between | Swing follows MVC. |

| | | |
|---|---|---|
| | model and view. | |

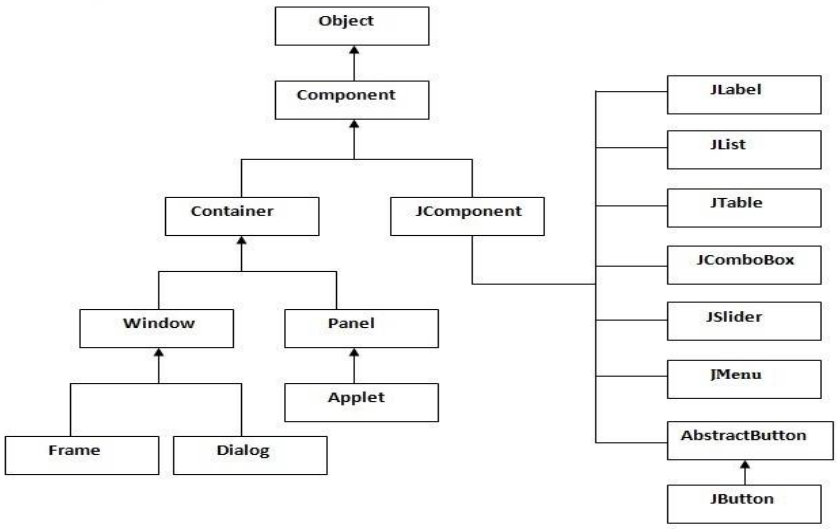
## JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Notes

# Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.




Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

| Method | Description |
|---|---|
| public void add(Component c) | add a component on another component. |
| public void setSize(int width,int height) | sets size of the component. |
| public void setLayout(LayoutManager m) | sets the layout manager for the component. |
| public void setVisible(boolean b) | sets the visibility of the component. It is by default false. |

**There are two ways to create a frame:**

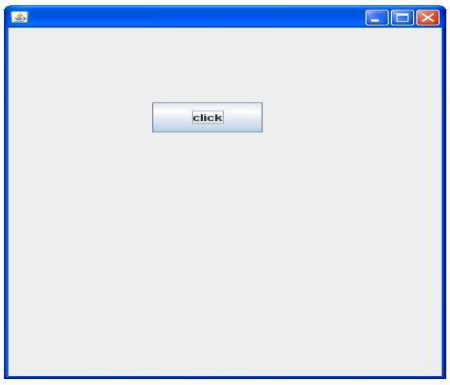
- **By creating the object of Frame class (association)**
- **By extending Frame class (inheritance)**

We can write the code of swing inside the main(), constructor or any other method.

**Simple Java Swing Example**

IN simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

```
import javax.swing.*;
public class FirstSwingExample
{
public static void main(String[ ] args)
{
JFrame f=new JFrame( );//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```




**Example of Swing by Association inside constructor**

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

```
import javax.swing.*;
public class Simple
```

```
{
JFrame f;
Simple( )
{
f=new JFrame();//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
public static void main(String[ ] args)
{
new Simple();
}
}
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

## Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
import javax.swing.*;
public class Simple2 extends JFrame
{//inheriting JFrame
JFrame f;
Simple2( )
{
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);
add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[ ] args)
{  new Simple2();  }}
```

## Review & Self Assessment Question

1. What is swing in java .
2. What is Java Foundation classes?
3. What are the difference between Java AWT and Java Swing ?
4. Define hierarchy of java swing classes.
5. What is panel in swing ?

## Further Readings

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

## Bibliography

Programming in Java by Herbert Schildt

Programming in Java by Kathy Sierra and Bert Bates

Programming in Java by Harimohan Pandey

Programming in Java by C Xavier

Programming in Java by O S Swift