OBJECT ORIENTED PROGRAMMING THROUGH JAVA

UNIT I

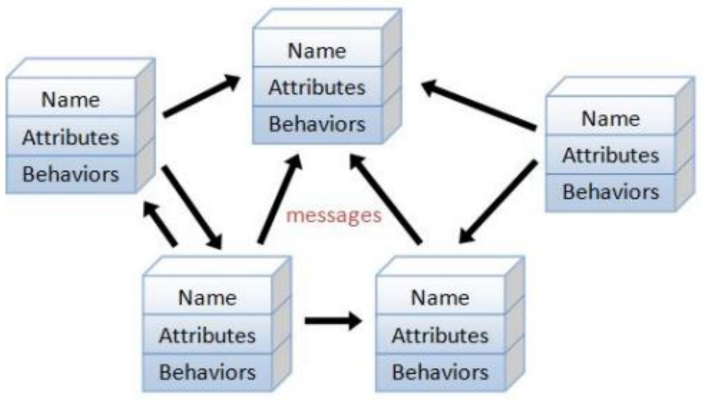
Thinking Object-Oriented:

Object-Oriented Programming was introduced by Alan Kay during 1960 – 1967. At that time it was not so popular. object oriented languages really came to the attention of the computing public-at-large in the 1980's. But nowadays the software is becoming more popular and the size of the software has increased.

Object-oriented programming (OOP) is a programming paradigm that uses "Objects "and their interactions to design applications.

The object oriented Programming Language is based upon the concept of "objects", which contains data as attributes in methods.

It simplifies the software development and maintenance.



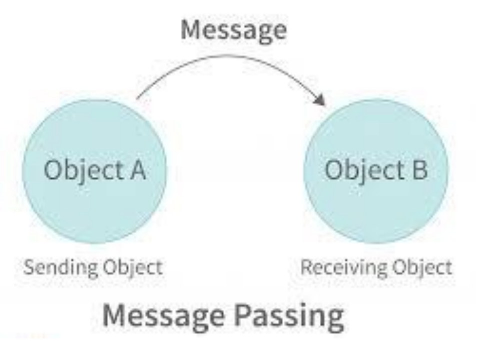
An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

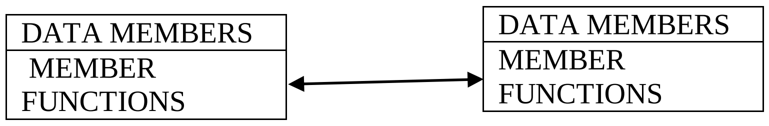
Object:

Objects are the basic run time entities in an object oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

Object
Instance of
Class
Beginnersbook.com

An object typically has:
1. **A state:** The properties or attributes of an object at a particular time.
2. **Behavior:** Methods represent the behavior of an object. Methods also define how the objects communicate.
3. **Identity:** Identifies the object by giving it a unique name.





1.Objects may communicate with each other through functions.
2. Data is hidden and cannot be accessed by External function
3. Bottom Up Approach

Data Member:

Variables declared within a class preceded by a data type which define the state of an object are called data members

Member Function:

Functions which define the behavior of an object of that class are called member functions.

Example:

```
class human
{
 private: string name;      // data member
 int age;                  // data member
 public: void run()        // member function
{ };
 void eat()                // member function
{ };
};
```

For Example – Consider a Television, It is an object. And the properties related to it are. It's With and Height, color, type (Smart TV or CRT TV), etc. And the behavior can be – we can change the channel, Adjust volumes and settings, switch off, switch on, etc are lots of behavior are there. The operation on the behavior will affect the properties. Changing the channel will affect its property.

So, similar things happen in Object-Oriented programming also, that each object has some properties and behavior associated with it, that are defined in the class.

Why OOPs are Popular in Comparison to Other Types of Programming Principles?

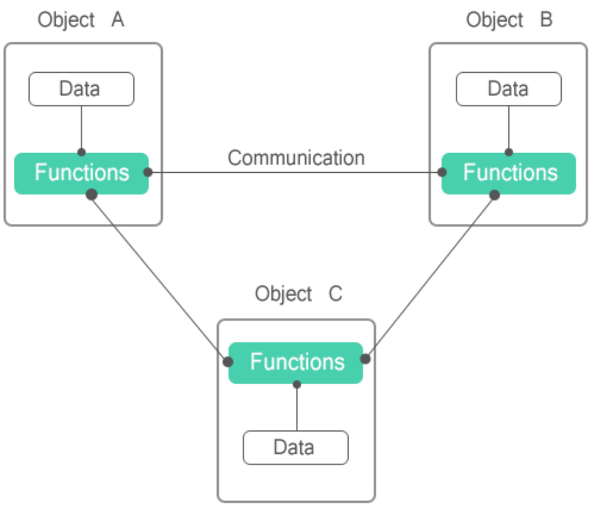
- Object-Oriented Programming Principles is the most popular amongst others because it relates to real-life objects.
- Every operation that is going to be functional is considered in terms of classes and objects. That provides a better programming style because you need not write code that needs to run anytime.
- Instead, you can create classes defining the functionality and call that function by creating an object of that.
- Languages that support Object-Oriented Programming are – C++, Java, JavaScript, C#, PHP, etc.

Components of Object-Oriented Programming

1. **Objects** – Object is the entity that makes the classes to be implemented into the program. It makes the features, properties, and behaviors of the class to be implemented. Example – A car is an object that has property color, model, brand name, fuel type, etc, and behaviors like, it runs. So these properties and behavior make an object (CAR).
2. **Classes –** A class can be stated as the blueprint of the object. This states what an object can do. It contains all the features, attributes, and behavior of what the model can do. We can say that class contains the definition of the object. Example – Car color, engine type, etc. And with the definition, we can create any number of objects.
3. **Methods** – Methods are the attributes of the class which are defined for the specified behavior of the object. It can also modify the state of the class. Example – Method to drive a car, It changes the state of the car from parking state to running state.
4. **Instances** – It is the members of the class who holds some values related to the objects of the class.

Object-Oriented Paradigm:

- The main necessity behind inventing object oriented approach is to remove the drawback encountered in the procedural approach.
- Object-oriented programming paradigm methods enable us to create a set of objects that work together to produce software that is better understandable and models their problem domains than produced using traditional techniques.
- Programs are divided into objects.
- Focus is on properties and functions rather than procedure.
- Data is hidden from external functions.
- Functions operate on the properties of an object.
- Objects may communicate with each other through a function called messaging.
- Follow the bottom-up approach in oop design



The major aspects of Object Oriented Programming (OOP) paradigm are as follows:

1. Reduced Maintenance :

The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated, the behaviors may be reused and incorporated into new behaviors.

2. Real-World Modeling :

Object-oriented system tend to model the real world in a more complete fashion than do traditional methods. Objects are organized into classes of objects, and objects are associated with behaviors. The model is based on objects, rather than on data and processing.

3. Improved Reliability and Flexibility :

Object-oriented system promise to be far more reliable than traditional systems, primarily because new behaviors can be "built" from existing objects. Because objects can be dynamically called and accessed, new objects may be created at any time. The new objects may inherit data attributes from one, or many other objects. Behaviors may be inherited from super-classes, and novel behaviors may be added without effecting existing systems functions.

4. High Code Re usability:

When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned. The new object will also inherit the data and behaviors from all super classes in which it participates.

STRUCTURED ORIENTED (OR) PROCEDURAL PROGRAMMING LANGUAGE:

- Programming in the high-level languages such as COBOL, FORTRAN, C, etc. is known as procedure-oriented programming.
- Procedure-oriented programming basically contains group of instructions known as function. There are multiple functions into the program.

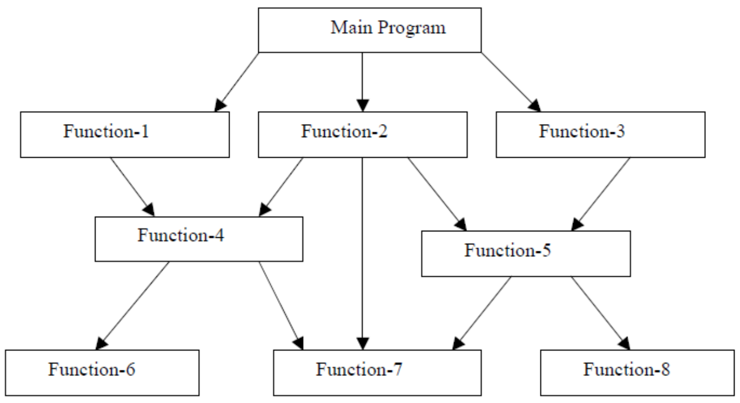
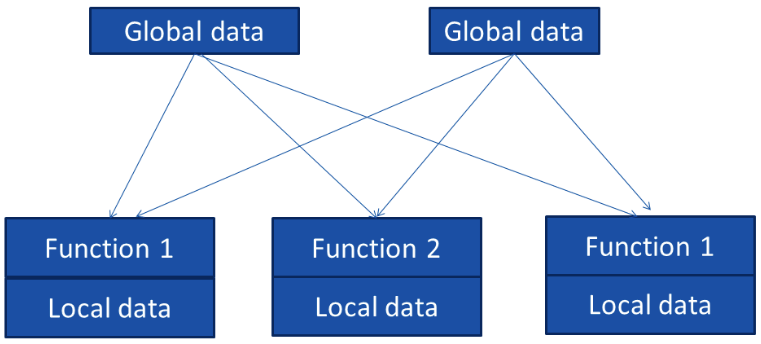


Fig. 1.2 Typical structure of procedural oriented programs

- In a multi-function program we use global variable to communicate between two functions.
- Global variable can be use by any function at any time while local variables are only used within the function.
- But it creates problem in large program because we can't determine which global variables (data) are used by which function.
- Also global variables are accessed by all the function so any function can change its value at any time so all the function will be affected.



Characteristics of procedure-oriented programming language:

1. It emphasis on algorithm.
2. Large programs are divided into smaller programs known as functions.
3. Function can communicate by global variable.
4. Data move freely from one function to another function.
5. Functions change the value of data at any time from any place. (Functions transform data from one form to another.)
6. It uses top-down programming approach.

What is Top-Down Approach?

Top Down Approach is an Approach to Design Algorithms in which a bigger problem is broken down into smaller Parts. Every part of the code is developed separately. Its having less interactions and communication between the modules in this approach.

What is Bottom Up Approach?

Bottom Up Approach is one in which the smaller problems are solved and then these solved problems are integrated to find the solution to a bigger problems. It requires a significant amount of Communication among different modules.

PROCEDURAL PROGRAMMING	OBJECT-ORIENTED PROGRAMMING
Object oriented programming can be described as a programming model which is based upon the concept of objects.	Procedural programming can be described as a programming model which is derived from structured programming, based upon the concept of calling procedures.
Follows a top-down approach	Follows a bottom-up approach
Instruction Oriented	Data Oriented
The abstraction is at procedure (function) level.	The abstraction is at object (class) level.
The sequence of events in a large program is divided into functions.	Entire program is divided into objects.
Real world is represented by 'procedures' operating on data.	Real world is represented by objects and the operations that can be performed on these objects.
Data and functions are separate.	Data and functions are encapsulated into a single unit.
Limited and difficult code reusability.	Versatile and easy code reusability.
Code is difficult to modify, extend and maintain.	Code is easy to modify, extend and maintain.
There is no access specifier in procedural programming. There is no access specifier in procedural programming. There is no access specifier in procedural programming.	Object-oriented programming has access specifiers like private, public, protected,etc.
Procedural programming does not have any proper way of hiding data so it is less secure.	Object-oriented programming provides data hiding so it is more secure.
In procedural programming, overloading is not possible.	Overloading is possible in object-oriented programming.
examples of Procedural Programming languages are C, COBOL, Pascal.	examples of Object Oriented languages are C++, Java, C#.

Advantages of OOPs

1. Re-usability

It means reusing some facilities rather than building them again and again. This is done with the use of a class.

2. Data Redundancy

This is a condition created at the place of data storage where the same piece of data is held in two separate places. So the data redundancy is one of the greatest advantages of OOP.

3. Code Maintenance

This feature is more of a necessity for any programming languages; it helps users from doing re-work in many ways. It is always easy and time-saving to maintain and modify the existing codes by incorporating new changes into them.

4. Security

With the use of data hiding and abstraction mechanism, we are filtering out limited data to exposure, which means we are maintaining security and providing necessary data to view.

5. Easy troubleshooting

lets witness some common issues or problems any developers face in their work.

6. Polymorphism Flexibility

Let’s see a scenario to better explain this behavior.

7. Problems solving

The broken components can be reused in solutions to different other problems

A WAY OF VIEWING THE WORLD:

a) To illustrate the major ideas in object-oriented programming, let us consider how we might go about handling a real-world situation and then ask how we could make the computer more closely model the techniques employed.

Suppose I wish to send owers to a friend who lives in a city many miles away. Let me call my friend Sally. Because of the distance, there is no possibility of my picking the owers and carrying them to her door myself. Nevertheless, sending her the owers is an easy enough task; I merely go down to my local orist (who happens to be named Flora), tell her the variety and quantity of owers I wish to send and give her Sally's address, and I can be assured the owers will be delivered expediently and automatically.

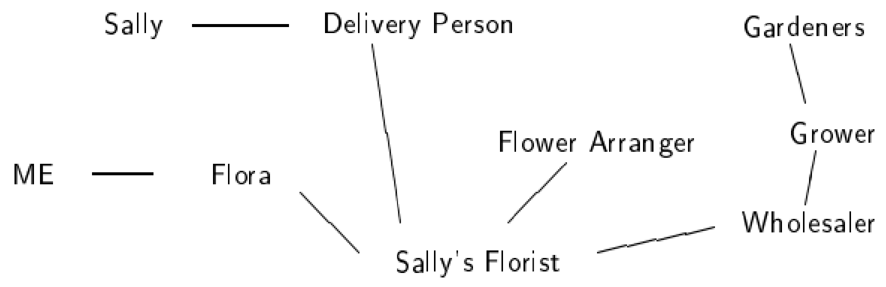


Figure 1.1: The community of agents helping me

Agents and Communities:

let me emphasize that the mechanism I used to solve my problem was to find an appropriate agent (namely, Flora) and to pass to her a message containing my request.

It is the responsibility of Flora to satisfy my request. There is some method{some algorithm or set of operations{used by Flora to do this. I do not need to know the particular method she will use to satisfy my request; indeed, often I do not want to know the details. This information is usually hidden from my inspection.

If I investigated however, I might discover that Flora delivers a slightly different message to another florist in my friend's city. That florist, in turn, perhaps has a subordinate who makes the flower arrangement. The florist then passes the flowers, along with yet another message, to a delivery person, and so on. Earlier, the florist in Sally's city had obtained her flowers from a flower wholesaler who, in turn, had interactions with the flower growers, each of whom had to manage a team of gardeners.

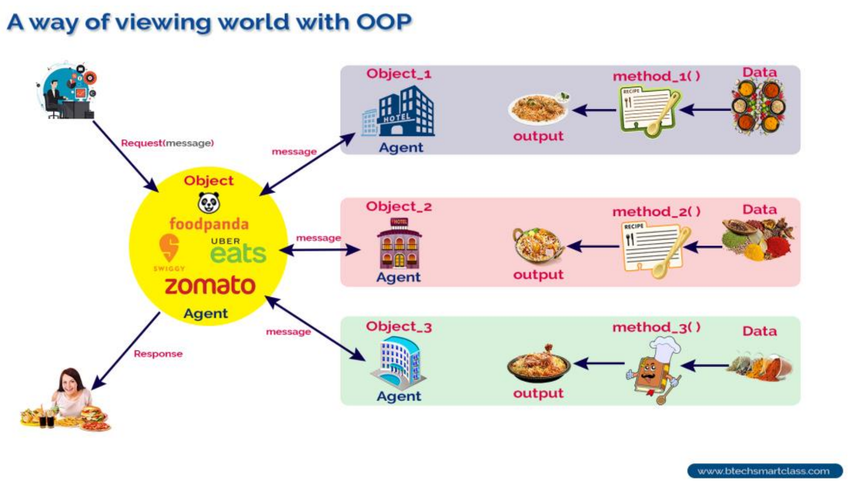
So, our first observation of object oriented problem solving is that the solution to my problem required the help of many other individuals. . Without their help, my problem could not be easily solved. We phrase this in a general fashion as the following:

An object oriented program is structured as a community of interacting agents, called objects. Each object has a role to play. Each object provides a service, or performs an action, that is used by other members of the community.

b) A way of viewing the world is an idea to illustrate the object-oriented programming concept with an example of a real-world situation.

Let us consider a situation, I am at my office and I wish to get food to my family members who are at my home from a hotel. Because of the distance from my office to home, there is no possibility of getting food from a hotel myself. So, how do we solve the issue?

To solve the problem, let me call zomato (an agent in food delevery community), tell them the variety and quantity of food and the hotel name from which I wish to delever the food to my family members. Look at the following image.



Agents and Communities:

To solve my food delivery problem, I used a solution by finding an appropriate agent (Zomato) and pass a message containing my request. It is the responsibility of the agent (Zomato) to satisfy my request. Here, the agent uses some method to do this. I do not need to know the method that the agent has used to solve my request. This is usually hidden from me.

So, in object-oriented programming, problem-solving is the solution to our problem which requires the help of many individuals in the community. We may describe agents and communities as follows.

An object-oriented program is structured as a community of interacting agents, called objects. Where each object provides a service (data and methods) that is used by other members of the community.

In our example, the online food delivery system is a community in which the agents are zomato and set of hotels. Each hotel provides a variety of services that can be used by other members like zomato, myself, and my family in the community.

Messages and Methods:

To solve my problem, I started with a request to the agent, which led to still more requests among the members of the community until my request has done. Here, the members of a community interact with one another by making requests until the problem has satisfied.

In object-oriented programming, every action is initiated by passing a message to an agent (object), which is responsible for the action. The receiver is the object to whom the message was sent. In response to the message, the receiver performs some method to carry out the request. Every message may include any additional information as arguments.

Responsibilities:

A fundamental concept in object-oriented programming is to describe behavior in terms of responsibilities. My request for action indicates only the desired outcome.

By discussing a problem in terms of responsibilities we increase the level of abstraction. This permits greater independence between objects, a critical factor in solving complex problems. The entire collection of responsibilities associated with an object is often described by the term protocol.

A traditional program often operates by acting on data structures, for example changing fields in an array or record. In contrast, an object oriented program requests data structures to perform a service.

Classes and Instances

In object-oriented programming, all objects are instances of a class. The method invoked by an object in response to a message is decided by the class. All the objects of a class use the same method in response to a similar message.

In our example, the zomato a class and all the hotels are sub-classes of it. For every request (message), the class creates an instance of it and uses a suitable method to solve the problem.

Classes Hierarchies

A graphical representation is often used to illustrate the relationships among the classes (objects) of a community. This graphical representation shows classes listed in a hierarchical tree-like structure. In this more abstract class listed near the top of the tree, and more specific classes in the middle of the tree, and the individuals listed near the bottom.

In object-oriented programming, classes can be organized into a hierarchical inheritance structure. A child class inherits properties from the parent class that higher in the tree.

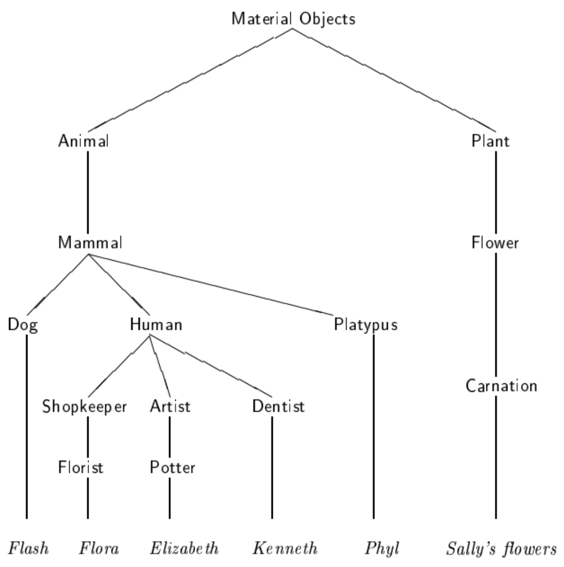


Figure 1.3: – A class hierarchy for various material objects.

This technique shows classes listed in a hierarchical tree-like structure, with more abstract classes (such as Material Object or Animal) listed near the top of the tree, and more specific classes, and finally individuals, are listed near the bottom.

Classes can be organized into a hierarchical inheritance structure. A child class (or subclass) will inherit attributes from a parent class higher in the tree. An abstract parent class is a class (such as Mammal) for which there are no direct instances; it is used only to create subclasses.

Method Binding, Overriding, and Exceptions:

In the class hierarchy, both parent and child classes may have the same method which implemented individually. Here, the implementation of the parent is overridden by the child. Or a class may provide multiple definitions to a single method to work with different arguments (overloading).

The search for the method to invoke in response to a request (message) begins with the class of this receiver. If no suitable method is found, the search is performed in the parent class of it. The search continues up the parent class chain until either a suitable method is found or the parent class chain is exhausted. If a suitable method is found, the method is executed. Otherwise, an error message is issued.

Summary of Object Oriented Concepts

- ✓ Everything is an object.
- ✓ An object is a runtime entity in an object oriented programming
- ✓ Computation is performed by objects communicating with each other.
- ✓ Objects communicate by sending and receiving messages.
- ✓ A message is a request for action bundled with whatever arguments may be necessary to complete the task.
- ✓ Each object has its own memory, which consists of other objects.
- ✓ Every object is an instance of a class.
- ✓ A class simply represents a grouping of similar objects, such as integers or lists.
- ✓ The class is the repository for behavior associated with an object.
- ✓ That is, all objects that are instances of the same class can perform the same actions.
- ✓ Classes are organized into a singly rooted tree structure, called the inheritance hierarchy.

- ✓ Memory and behavior associated with instances of a class are automatically available to any class associated with a descendant in this tree structure.

BASIC CONCEPTS OF OBJECTS ORIENTED PROGRAMMING:

1. Objects
2. Classes
3. Data abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism
7. Overloading
8. Dynamic binding
9. Message passing

OBJECTS:

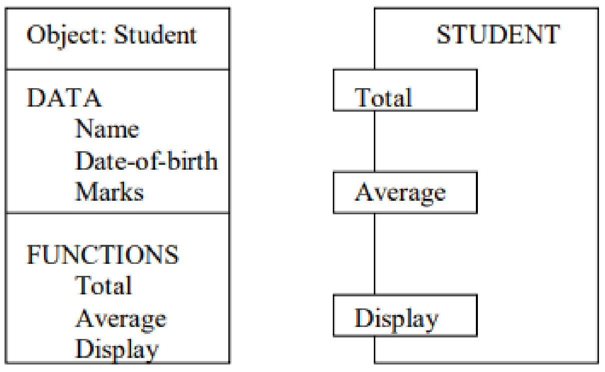
Objects are the basic run-time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program must handle.

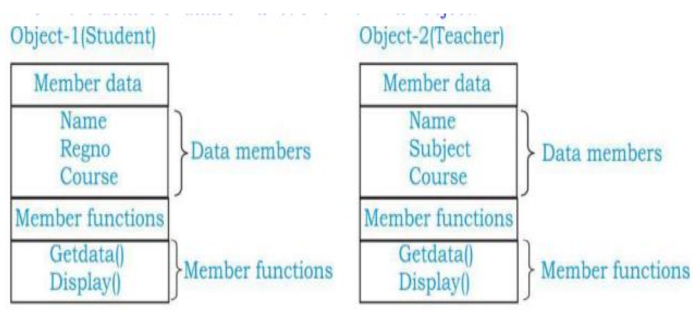
The fundamental idea behind object oriented approach is to combine both data and function into a single unit and these units are called objects.

The term objects means a combination of data and program that represent some real word entity.

For example: consider an example named kumar. kumar is 25 years old and his salary is 25000. The Kumar may be represented in a computer program as an object. The data part of the object would be (name: Kumar, age: 25, salary: 25000)

The program part of the object may be collection of programs (retrive of data, change age, change of salary). In general even any user –defined type-such as employee may be used. In the Amit object the name, age and salary are called attributes of the object.

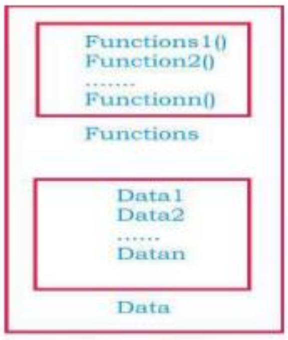




CLASS:

A class can be considered as a blue print of object. A group of objects that share common properties for data part and some program part are collectively called as class.

Class is a collection of objects of similar type. Objects are variables of the type of class. Once a class has been defined, we can create any number of objects belonging to that class.



DATA ABSTRACTION :

Abstraction is the process of hiding the internal details of an application from the outer world. Classes use the concept of abstraction and are defined as size, width and cost and functions to operate on the attributes.

Types of Abstraction

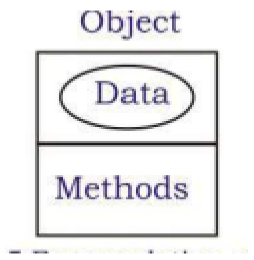
There are two types of abstraction.

1. Data Abstraction
2. Process Abstraction

DATA ENCAPSULATION :

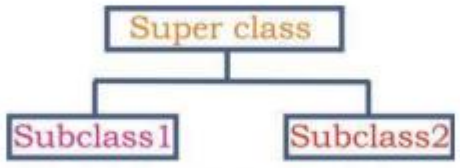
The wrapping up of data and function into a single unit (called class) is known as encapsulation.

Data encapsulation combines data and functions into a single unit called class. Data encapsulation will prevent direct access to data. The data can be accessed only through methods (function) present inside the class. The data cannot be modified by an external nonmember function of a class. Data encapsulation enables data hiding or information hiding.



INHERITENCE:

Inheritance is a process of creating a new class from the base class. Inheritance is the process by which objects of one class acquire the properties of another class. In the concept of inheritance provides the idea of reusability.



POLYMORPHISIM:

The word “polymorphism” means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A real-life example of polymorphism is a person who at the same time can have different characteristics.

A man at the same time is a father, a husband, and an employee. So the same person exhibits different behavior in different situations. This is called polymorphism. Polymorphism is considered one of the important features of Object-Oriented Programming.

Types of Polymorphism

- Compile-time Polymorphism
- Runtime Polymorphism

OVERLOADING:

Overloading allows objects to have different meaning depending uponcontext. There are 2 types of overloading namely

1. Operator overloading
2. Function overloading

When an existing operator operates on new data type, it is called operator overloading. Function overloading means two or more functions have same name ,but differ in the number of arguments or data type of arguments. Therefore it is said that (function name) is overloaded. Function overloading therefore is the process of defining same function name to carry out similar types of activities with various data items.

DYNAMIC BINDING:

Binding is the process of connecting one program to another. Dynamic binding means code associated with a procedure call is known only at the time of program execution routine.

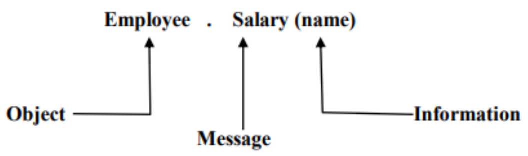
MESSAGE PASSING:

In OOP, processing is done by sending messages to objects. A message for an object is request for execution of procedure.

Message Passing in terms of computers is communication between processes.

An object oriented program consists of a set of objects that communicate with each other.

A message for an object is a request for execution of a procedure and therefore will invoke a function (procedure) in the receiving object that generates the desired result. Message passing involves specifying the name of the object, the name of the function (message) and information to be sent.



Applications of object oriented programming:

1. Real – Time systems.
2. Simulation and modeling
3. Object oriented databases.
4. Hypertext, hypermedia and expertext.
5. Al and expert systems.
6. Neural networks and parallel programming.
7. Decision support and office automation systems.
8. CIM / CAM / CAD system.

What is coping with complexity in oops?

Coping with complexity in Object-Oriented Programming (OOP) refers to the ability of the programming paradigm to manage and handle the intricacies and intricacies of large and complex software systems. OOP provides several features and principles that help developers manage complexity and build scalable and maintainable software. The size and complexity of a small program is small and simple. Whereas, the size and complexity of a large application program is large and hard. The complexity in dealing with the problems to build a large application depends on the 'composition' and 'abstraction' mechanisms.

Here are some key aspects of coping with complexity in OOP:

1. Encapsulation:

Encapsulation is the bundling of data and the methods that operate on that data into a single unit, known as a class. It hides the internal details of the object and exposes only what is necessary. This reduces complexity by providing a clear interface and allowing changes to be made within the class without affecting the external code.

2. Abstraction:

Abstraction involves simplifying complex systems by modeling classes based on the essential properties and behaviors they share. It allows developers to focus on the relevant details while ignoring unnecessary complexities. Abstraction also enables the creation of abstract classes and interfaces, which define common behaviors without specifying the implementation details.

3. Inheritance:

Inheritance allows a class to inherit properties and behaviors from another class. This promotes code reuse and helps manage complexity by creating a hierarchy of classes. Common functionalities can be defined in a base class, and more specific classes can inherit from it, inheriting and extending its features.

4. Polymorphism:

Polymorphism allows objects to be treated as instances of their base class rather than their specific class type. This simplifies code by enabling the use of a common interface for different types of objects. Polymorphism can take the form of method overloading or method overriding.

5. Modularity:

OOP promotes modularity by breaking down a complex system into smaller, manageable modules or classes. Each class is responsible for a specific aspect of the system, making it easier to understand, test, and maintain.

6. Message Passing:

Objects in OOP communicate by sending messages to each other. This message passing mechanism helps in reducing complexity by encapsulating the interactions between objects. Objects only need to know the interface of the objects they interact with, not their internal details.

7. Encapsulation of State and Behavior:

OOP encapsulates both state (data) and behavior (methods) within objects. This encapsulation helps in managing complexity by keeping related data and methods together, making it easier to understand and modify the behavior of objects.

OVERVIEW OF JAVA:

Java is a programming language created by James Gosling from Sun Microsystems in 1991. Java allows to write a program and run it on multiple operating systems.

Java is a class-based object-oriented simple programming language. It is a general-purpose, high-level programming language that helps programmers and developers to write a code once and run it anywhere.

Java is considered both a compiled and interpreted language.

Java divided into three categories, they are

- J2SE (Java 2 Standard Edition)- J2SE is used for developing client side applications.
- J2EE (Java 2 Enterprise Edition)- is used for developing server side applications.
- J2ME (Java 2 Micro or Mobile Edition)- is used for developing mobile or wireless application by making use of a predefined protocol called WAP

Java Terminology

1. Java Compiler: The JDK includes the Java compiler (javac), which converts Java source code (.java files) into byte code (.class files) that can be executed by the Java Virtual Machine (JVM).

2. Java Virtual Machine(JVM):

VM (Java Virtual Machine): The JVM is a virtual machine that executes Java bytecode. It provides an environment for running Java programs and translates the bytecode into machine code that can be understood by the underlying hardware.

3. Java Development Kit (JDK):

The Java Development Kit (JDK) is a software development environment used for developing Java applications. It includes a set of tools, libraries, and compilers that are necessary for writing, compiling, and running Java programs.

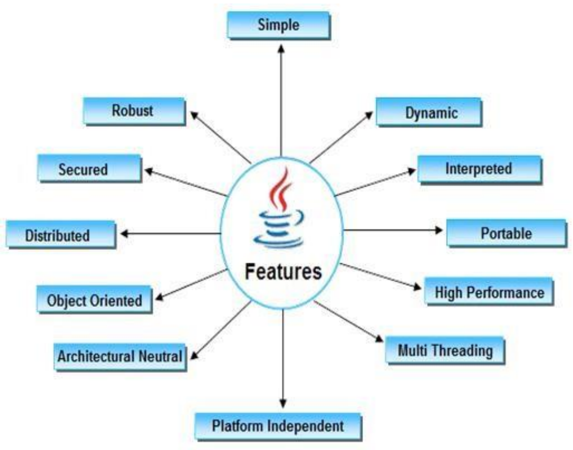
4. Java Runtime Environment (JRE):

The JRE is a subset of the JDK and includes the JVM, libraries, and other files necessary for running Java applications. It does not contain the development tools (such as the compiler) included in the JDK.

Java Buzzwords or Features of Java:

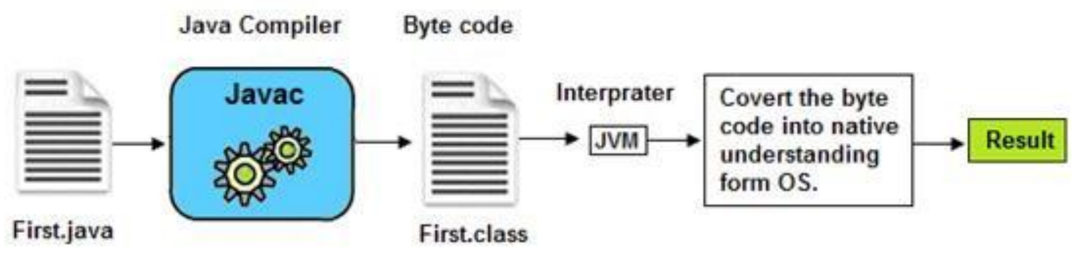
The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java buzzwords.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed



1. Simple:

- It is simple because of the following factors:
- It has Rich set of API (application protocol interface).
- It has Garbage Collector which is always used to collect un-Referenced (unused) Memory location for improving performance of a Java program.
- It contains user friendly syntax for developing any applications.

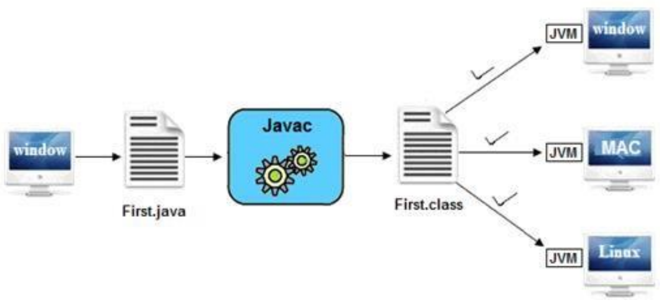


2. Object-Oriented:

Java is a fully object-oriented programming language. It supports key concepts of object-oriented programming, including encapsulation, inheritance, and polymorphism. Object-oriented programming enables modular and reusable code, enhances code organization, and promotes code maintainability.

3. Platform independent:

A program or technology is said to be platform independent if and only if which can run on all available operating systems with respect to its development and compilation.



4.Secured:

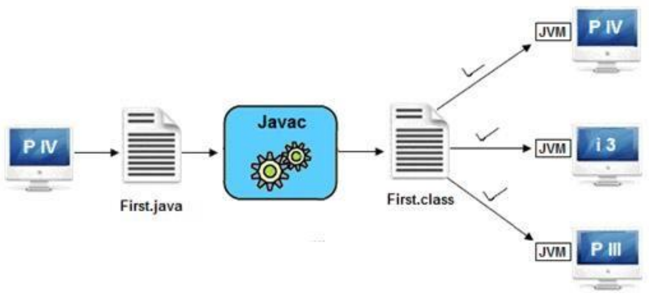
It is more secured language compare to other language; in this language all code is converted into byte code after compilation which is not readable by human.

5.Robust:

Simply means of Robust is strong. It is robust or strong Programming Language because of its capability to handle Run-time Error, automatic garbage collection, lack of pointer concept, Exception Handling. All these points make it robust Language.

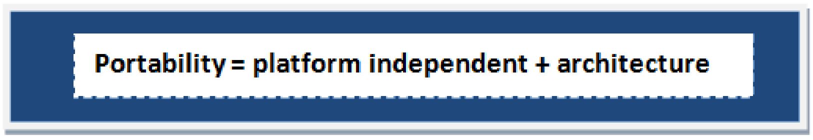
6. Architecture neutral:

Architecture represents processor. A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering there architecture and vendor (providers) irrespective to its development and compilation.



7. Portable:

If any language supports platform independent and architectural neutral feature known as portable. The languages like C, CPP, and Pascal are treated as non-portable language. It is a portable language.


Portability = platform independent + architecture

8. Dynamic:

It supports Dynamic memory allocation, due to this memory wastage is reduced and performance of application is improved. The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation. To allocate memory space dynamically we use an operator called 'new'. 'new' operator is known as dynamic memory allocation operator.

9. Interpreted:

It is one of the highly interpreted programming languages.

10. High Performance:

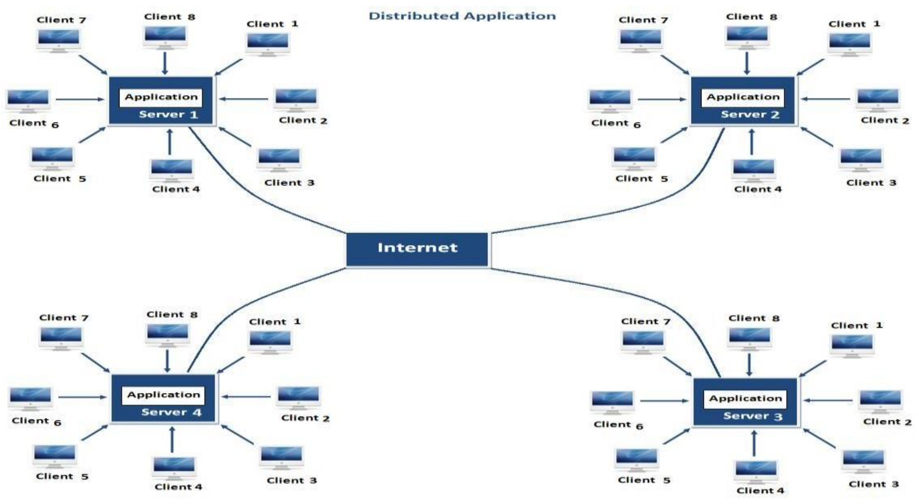
- It has high performance because of following reasons;
- This language uses Byte code which is faster than ordinary pointer code so Performance of this language is high.
- Garbage collector, collect the unused memory space and improve the performance of application.
- It has no pointers so that using this language we can develop an application very easily.
- It support multithreading, because of this time consuming process can be reduced to execute the program.

11. Multithreaded:

A flow of control is known as thread. When any Language executes multiple threads at a time that language is known as multithreaded Language. It is multithreaded Language.

12. Distributed:

Using this language we can create distributed application. RMI and EJB are used for creating distributed applications. In distributed application multiple client system are depends on multiple server systems so that even problem occurred in one server will never be reflected on any client system.



APPLICATIONS USING JAVA:

There are mainly 4 type of applications that can be created using java

a. Standalone Application:

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

b. Web Application:

An application that runs on the server side and creates dynamic page, is called web application. Currently, Servlet, JSP, struts, JSF etc. technologies are used for creating web applications in java.

c. Enterprise Application:

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

d. Mobile Application:

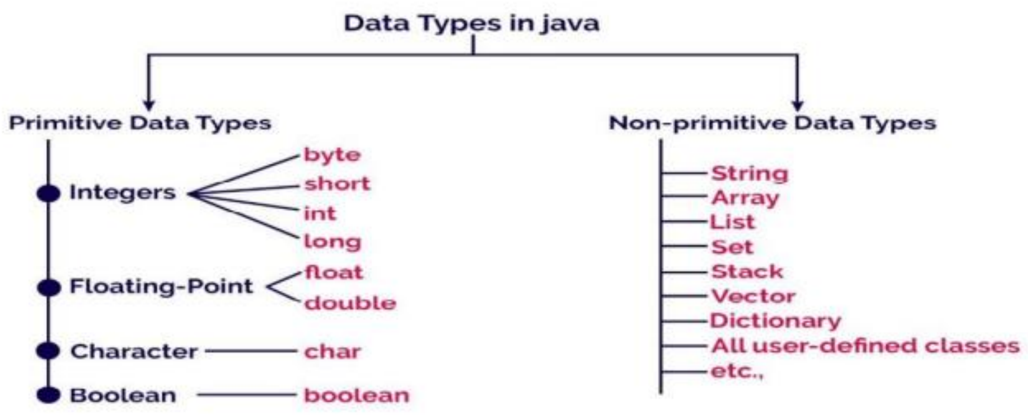
An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

Data Types in Java:

Data type is a special keyword used to allocate sufficient memory space for the data, in other words Data type is used for representing the data in main memory (RAM) of the computer.

Java programming language has a rich set of data types. The data type is a category of data stored in variables. In java, data types are classified into two types and they are as follows.

- Primitive Data Types
- Non-primitive Data Types



Primitive Data Types:

The primitive types are also commonly referred to as simple types.

The primitive data types include Boolean, char, byte, short, int, long, float and double.

These can be put in four groups:

1. Integers: This group includes byte, short, int, and long, which are for whole-valued signed numbers.

2. Floating-point numbers: This group includes float and double, which represent numbers with fractional precision.

3. Characters: This group includes char, which represents symbols in a character set, like letters and numbers.

4. Boolean: This group includes Boolean, which is a special type for representing true/false values.

Type	*Size(bytes)*	*Range*	*Default*
byte	1	-128 .. 127	0
short	2	-32768 to32767	0
Int	4	-2147483648 to 2147483647	0
long	8	-9223372036854775808 to 9223372036854775807	0
float	4	3.4 e38 to 1.4 e-45	0.0
double	8	1.e-308 to 4.9e-324	0.0
boolean	JVM Specific	true or false	FALSE
char	2	0 ..65535	\u0000

Int Data Type:

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example:

int a = 100000, int b = -200000

Float Data Type:

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example:

float f1 = 234.5f

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example:

char letterA = 'A'

Boolean Data Type:

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example:

Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example:

byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example:

short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example:

int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1) (inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example:

long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example:

float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example:

double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example:

char letterA = 'A’

Writing and Execution of simple Program

```
public class HelloWorld

{

public static void main(String[] args)

{

System.out.println("Hello world!");

}

}
```

Structure of the Java Program

Example

```
import java.lang.*;        //              package detials

class Demo                 //              class Classname

{

int x=10;                  //              Datamembers

void display()             //              user defined methods;

{

System.out.println("java program structure");     //

}

public static void main(string args[])            //       block of statements;

{

Demo obj=new Demo();

System.out.println("x value is"+obj.x);

Obj.display();

} }
```

A package is a collection of classes, interfaces and sub-packages. A sub package contains collection of classes, interfaces and sub-sub packages etc. java.lang.*; package is imported by default and this package is known as default package.

Class is keyword used for developing user defined data type and every java program must start with a concept of class.

"ClassName" represent a java valid variable name treated as a name of the class each and every class name in java is treated as user-defined data type.

Data member represents either instance or static they will be selected based on the name of the class.

User-defined methods represents either instance or static they are meant for performing the operations either once or each and every time.

Block of statements represents set of executable statements which are in term calling user-defined methods are containing business-logic.

The file naming conversion in the java programming is that which-ever class is containing main() method, that class name must be given as a file name with an extension .java.

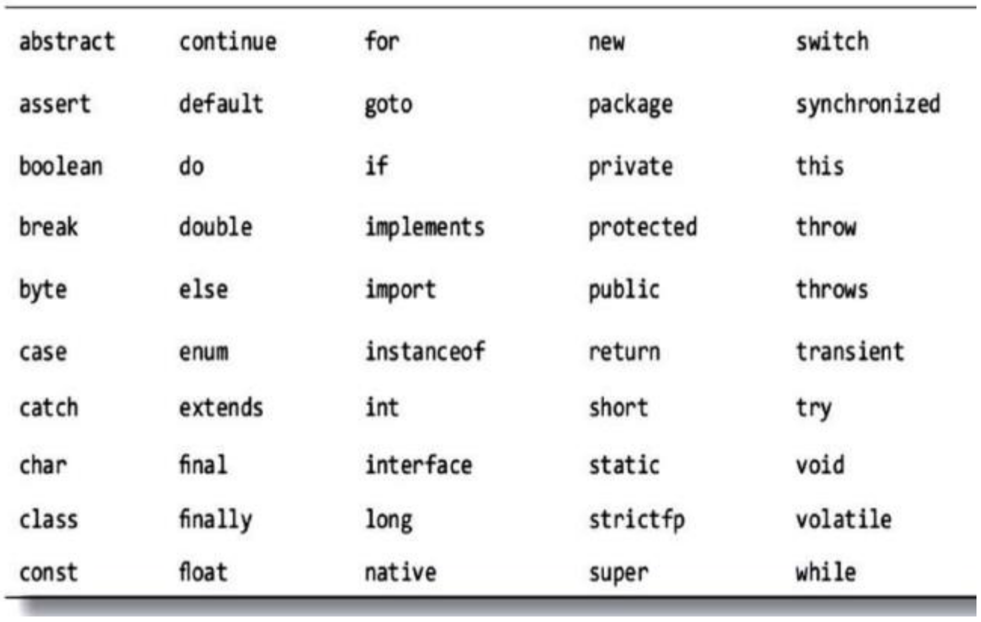
LEXICAL ISSUES (JAVA TOKENS)

A java 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 and separated by delimiters. The delimiters are not part of the tokens. A token is the smallest element of a program that is meaningful to the compiler. The compiler breaks lines into chunks of text called tokens.

- Keywords
- Identifiers or names
- Literals
- Separators
- Operators

KEYWORDS:
These are special words defined in Java and represent a set of instructions.
The keywords represent groups of instructions for the compiler.
• These are special tokens that have a predefined meaning and their use is restricted.
• keywords cannot be used as names of variables, methods, classes, or packages.
• These are written in the lower case.
• Keywords of Java Language are as follows:



abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

IDENTIFIERS OR NAMES:

- Identifier is the name of variables, methods, classes etc. Rules for framing Names or Identifiers.
- It should be a single word which contains alphabets a to z or A to Z, digits 0 to 9, underscore (_).
- It should not contain white spaces and special symbols.
- It should not be a keyword of Java.

- It should not be Boolean literal, that is, true or false.
- It should not be null literal.
- It should not start with a digit but it can start with an underscore.
- It can comprise one or more unicode characters which are characters as well as digits.

Conventions for Writing Names:

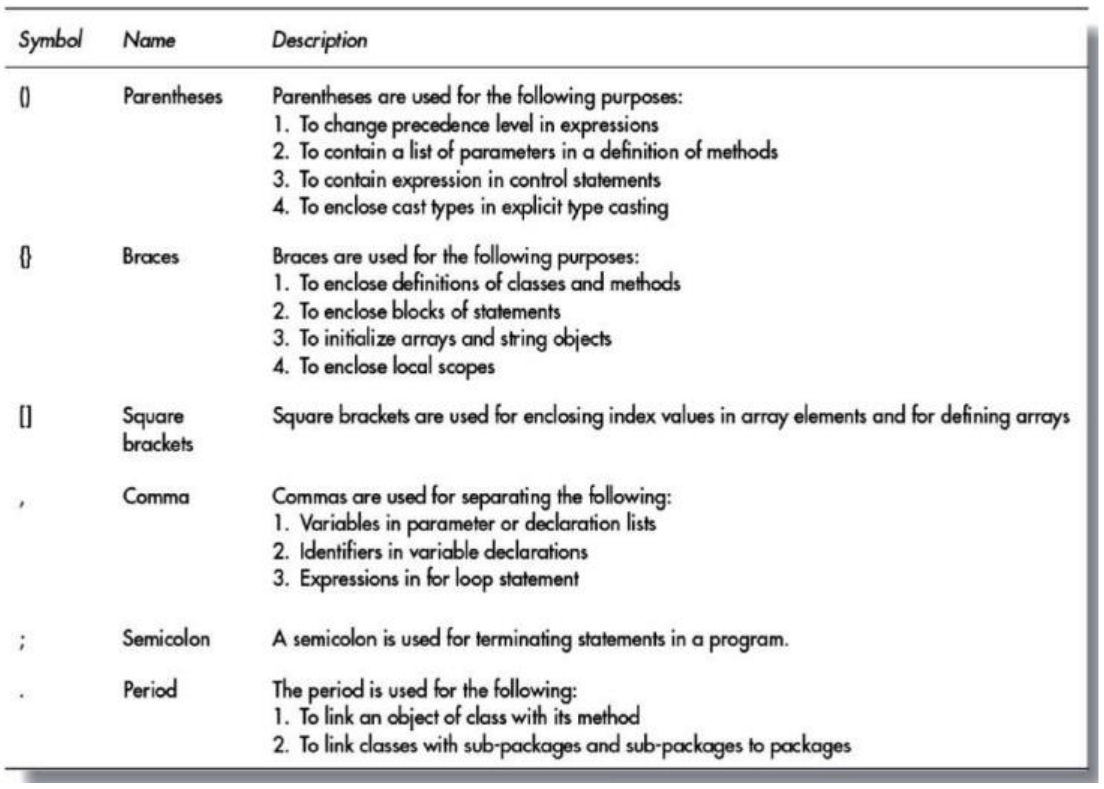
Names of packages are completely in lower-case letters such as mypackage, java.lang.
• Names of classes and interfaces start with an upper-case letter.
• Names of methods start with a lower-case character.
• Names of variables should start with a lower-case character.

LITERALS:

These are values represented by a set of character, digits, etc.
• A literal represents a value which may be of primitive type, String type, or null type.
• The value may be a number (either whole or decimal point number) or a sequence of characters which is called String literal, Boolean type, etc.
• A literal is a constant value.

SEPARATORS:

These include comma, semicolon, period(.), Parenthesis (), Square brackets [], etc



Symbol	*Name*	*Description*
()	Parentheses	Parentheses are used for the following purposes: 1. To change precedence level in expressions 2. To contain a list of parameters in a definition of methods 3. To contain expression in control statements 4. To enclose cast types in explicit type casting
{}	Braces	Braces are used for the following purposes: 1. To enclose definitions of classes and methods 2. To enclose blocks of statements 3. To initialize arrays and string objects 4. To enclose local scopes
[]	Square brackets	Square brackets are used for enclosing index values in array elements and for defining arrays
,	Comma	Commas are used for separating the following: 1. Variables in parameter or declaration lists 2. Identifiers in variable declarations 3. Expressions in for loop statement
;	Semicolon	A semicolon is used for terminating statements in a program.
.	Period	The period is used for the following: 1. To link an object of class with its method 2. To link classes with sub-packages and sub-packages to packages

OPERATORS: Operators are mostly represented by symbols such as +, -, *, etc

Types of Operators:

Arithmetic Operators:

Operator	Description
+	Addition or Unary plus
-	Subtraction or Unary minus
*	Multiplication
/	Division
%	Modulus

Relational Operators:

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

Logical Operators:

Operator	Description
&&	Greater than
||	Greater than or equal to
!	Less than

Assignment Operators:

Operator	Description
+=	Add and assign to
-=	Subtract and assign to
*=	Multiply and assign to
/=	Divide and assign to
%=	Modulus and assign to

Increment / Decrement Operators:

Operator	Description
++	Increment by one
--	Decrement by one

Bitwise Operators:

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise compliment
>>	Shift Right
<<	Shift Left
>>>	Shift right with Zero fill

Conditional Operators:

Operator	Description
?:	Used to construct Conditional expression

VARIABLES:

It is an identifier that denotes a storage location used to store a data value. A variable provides us with named storage that our programs can manipulate.

A variable name may consists of alphabets, digits or underscore.

As mentioned earlier, variable names may consist of alphabets, digits, the underscore (_) and dollar characters, subject to the following conditions:

- ✓ It should not be a keyword.
- ✓ White space is not allowed.

- ✓ Variable names can be of any length.
- ✓ They must not begin with a digit.

Declaration of Variables:

In Java, variables are the names of storage locations. After designing suitable variable names, we must declare them to the compiler.

A variable must be declared before it is used in the program.

A variable can be used to store a value of any data type.

The general form of declaration of a variable is:

```
type variable1, variable2, ……………….., variableN;
```

Variables are separated by commas. A declaration statement must end with a semicolon. Some valid declarations are:

Example:

int a, b, c; // Declares three ints, a, b, and c.

int a = 10, b = 10; // Example of initialization

byte B = 22; // initializes a byte type variable B.

double pi = 3.14159; // declares and assigns a value of PI.

char a = 'a'; // the char variable a is initialized with value 'a'

There are three kinds of variables in Java:

- ➢ Local variables
- ➢ Instance variables
- ➢ Class/static variables

Example:

```
class A
{
int data=50;            //instance variable
static int m=100;       //static variable
void method( )
{
int n=90;               //local variable
}
}                       // end of class
```

Local Variable:

A variable declared inside the body of the method is called local variable.

Access modifiers cannot be used for local variables.

There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

Example:

Here, age is a local variable. This is defined inside pupAge() method and its scope is limited to this method only.

```
public class Test

{

public void pupAge( )

{

int age = 0;

age = age + 7;

System.out.println("Puppy age is : " + age);

}

public static void main(String args[])

{

Test test = new Test();

test.pupAge();

}
```

Output:

```
Puppy age is: 7
```

Instance Variable:

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

Instance variables are created when an object is created with the use of the keyword 'new' and dest royed when the object is destroyed.

Instance variables can be declared in class level before or after use.

Instance variables have default values. For numbers the default value is 0

Instance variables can be accessed directly by calling the variable name inside the class.

Class/Static variable:

Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.

Static variables are stored in static memory.

Static variables are created when the program starts and destroyed when the program stops.

Static variables can be accessed by calling with the class name . ClassName.VariableName.

SCOPE AND LIFETIME OF VARIABLES

The scope of a variable refers to the areas or the sections of the program in which the variable can be accessed, and the lifetime of a variable indicates how long the variable stays alive in the memory.

The scope and lifetime of a variable is *"how and where the variable is defined."*

JAVA ARRAYS

An array is a collection of similar data types with a contiguous memory location. It is used to store multiple values in a single variable instead of declaring separate variables for each value. It is also known as static data structure because size of an array must be specified at the time of its declaration.

Arrays are indexed, meaning each array element has a specific position (also called index) that is used to access it. Array starts from zero index and goes to n-1 where n is length of the array.

There are two types of arrays in Java.

- One-Dimensional Arrays.
- Multi-Dimensional Arrays.

One-Dimensional Arrays:

Single dimensional array use single index to store elements.

Syntax:

```
dataType[] arr;

dataType []arr;

dataType arr[];
```

Example:

```
int[] intArray; // Declare an integer array
```

Example on Single Dimensional Array:

```
class Testarray1

{

public static void main(String args[])

{

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

for(int i=0;i<a.length;i++)

System.out.println(a[i]);

}}
```

Output : 33

```
        3
        4
        5
```

Multi-Dimensional Arrays

In Java, a multidimensional array is an array of arrays. It is a data structure that allows you to store multiple arrays in a single array. The most common types of multidimensional arrays are two-dimensional (2D) and three-dimensional (3D) arrays, but Java also supports arrays with more than three dimensions. These types of arrays are also known as jagged arrays.

Syntax

```
<data-type>[][] <variable-name>

<data-type> <variable-name>[][]

 int[] myArray = {1, 2, 3, 4, 5};
```

Examples on Two Dimensional array

Example 1: program to declare,create,initialize and access the elements of 2-D array

```
class array2d
{
public static void main(String args[])
 {
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
for(int i=0;i<3;i++)
{
for(int j=0;j<3;J++)
{
System.out.print(arr[i][j]+" ");
}
System.out.println();
}}}
```

```
Output:1 2 3
        2 4 5
        4 4 5
```

Jagged Array:

Jagged array is an array that has different numbers of columns elements. In java, a jagged array means to have a multi-dimensional array with uneven size of columns in it.

```
int[ ][ ] arr = new int[3][ ];

arr[0] = new int[3];

arr[1] = new int[4];

arr[2] = new int[5];
```

Rules for creating an Array

- At the time of array creation compulsory we should specify the size otherwise we will get compile time error.
- It is legal to have an array with size zero in java.
- The allowed data types to specify array size are byte, short, char, int. By mistake if we are using any other type we will get compile time error.
- The maximum allowed array size in java is maximum value of int size [2147483647].

Example 2 : program for Addition of two matrix

```
class twodarrayaddition

{

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
}}}
```

Output : 2 6 8

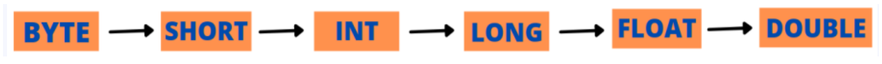
```
        6 8 10
```

TYPE CONVERSION

Type conversion is a process in which the data type is automatically converted into another data type.

The compiler does this automatic conversion at compile time. This type of type conversion is also called Widening Type Conversion/ Implicit Conversion/ Casting Down.

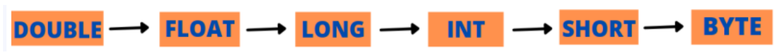
For type conversion to take place, the destination data type must be larger than the source type. In short, the below flow chart has to be followed.



In this case, as the lower data types with smaller sizes are converted into higher ones with a larger size, there is no chance of data loss. This makes it safe for usage.

TYPE CASTING

Type casting is a process in which the programmer manually converts one data type into another data type. For this the casting operator (), the parenthesis is used. Type casting is also called Narrowing Type Casting/ Explicit Conversion/ Casting Up.



This case, as the higher data types with a larger size are converted into lower ones with a smaller size, there is a chance of data loss.

Java Expressions

An expression is a series of variables, operators, and method calls that evaluates to a single value.

An expression is a collection of operators and operands that represents a specific value.

An operator is a symbol that performs tasks like arithmetic operations, logical operations, and conditional operations, etc.
Operands are the values on which the operators perform the task. Here operand can be a direct value or variable or address of memory location.

In the java programming language, expressions are divided into THREE types. They are as follows.

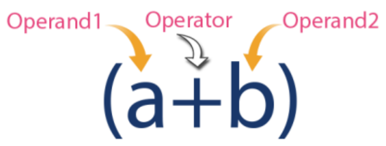
Infix Expression

Postfix Expression

Prefix Expression

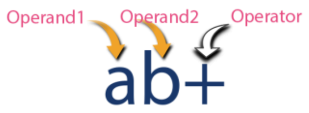
Infix Expression

The expression in which the operator is used between operands is called infix expression.



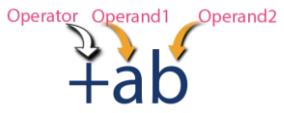
Postfix Expression

The expression in which the operator is used after operands is called postfix expression.



Prefix Expression

The expression in which the operator is used before operands is called a prefix expression.



OPERATORS

Java supports a rich set of operators. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions.

Java operators can be classified into a number of related categories as below:

- Arithmetic Operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and decrement operators
- Conditional operators
- Bitwise operators
- Special operators

Arithmetic Operators:

Arithmetic operators are used to construct mathematical expressions as in algebra. Java provides all the basic arithmetic operators. These can on any built-in numeric data type of Java.

Operator	*Meaning*
+	Addition or unary plus
–	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division (Remainder)

Arithmetic operators are used as shown below:

a – b	a + b
a * b	a / b
a % b	– a * b

Here **a** and **b** may be variables or constants and are known as operands.

Relational Operators

We often compare two quantities, and depending on their relation, take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators.

a < b or x < 20

Operator	*Meaning*
<	is less than
<=	is less than or equal to
>	Is greater than
>=	Is greater than or equal to
= =	Is equal to
!=	is not equal to

Logical Operators

The logical operators && and | | are used when we want to form compound conditions by combining two or more relations. An example is:

a > b && x == 10

An expression of this kind which combines two or more relational expressions is termed as a logical expression or a compound relational expression.

Operator	*Meaning*
&&	logical AND
| |	logical OR
!	logical NOT

Assignment Operators:

Assignment operators are used to assign the value of an expression to a variable. We have seen the usual assignment operator, '='. In addition, Java has a set of 'shorthand' assignment operators which are used in the form

```
v op= exp;
```

where v is a variable, exp is an expression and op is a Java binary operatory. The operator op = is known as the shorthand assignment operator.

Increment and Decrement Operators:

Java has two very useful operators not generally found in many other languages. These are the increment and decrement operators.

++ and —

Conditional Operator

The character pair ? : is a ternary operatory available in Java. This operator is used to construct conditional expressions of the form

```
exp1 ? exp2 : exp3
```

where exp1, exp2, and exp3 are expressions.

The operator ? : works as follows: exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of the conditional expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the conditional expression. None that only one of the expressions (either exp2 or exp3) is evaluated. For example, consider the following statements:

a = 10; b = 15; x = (a > b) ? a : b;

In this example, x will be assigned the value of b. This can be achieved using the if….else statement as follows:

if(a > b)

x = a;

else

x = b;

Bitwise Operators:

Java has a distinction of supporting special operators known as bitwise operators for manipulation of data at values of bit level.

These operators are used for testing the bits, or shifting them to the right or left. Bitwise operators may not be applied to float or double.

Operator	*Meaning*
&	bitwise AND
!	bitwise OR
^	Bitwise exclusive OR
~	one's complement
<<	shift left
>>	shift right
>>>	shift right with zero fill

Special Operators(Miscellaneous Operators)

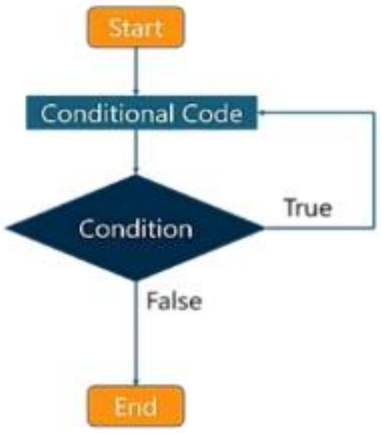
Java supports some special operators of interest such as instanceof operatory and member selection operator (.).

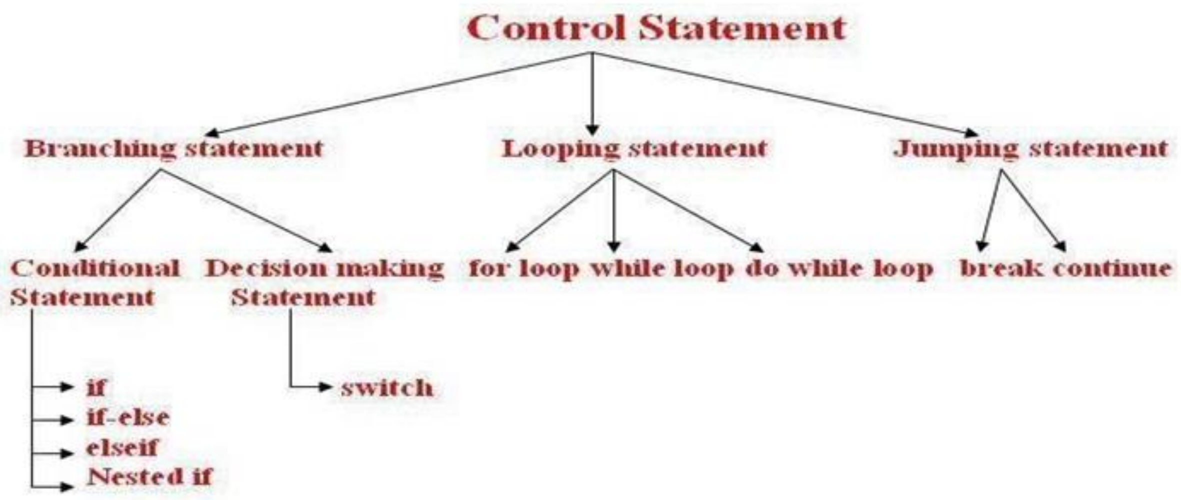
The instanceof is an object operator and returns true of the object on the left-hand side is an instance of the class given on the right-hand side. This operator allows us to determine whether the object belongs to a particular class or not.

Example: person instanceof student

CONTROL STATEMENTS

A programming language uses control statements to control the flow of execution of a program. In Java programming, we can control the flow of execution of a program based on some conditions. Java control statements can be put into the following three categories:





SELECTION STATEMENTS(CONDITIONAL STATEMENT):

The Java selection statements allow your program to choose a different path of execution based on a certain condition. Java selection statements provide different type of statements such as:

1. if statement.

2. if-else statement.

3. if-else-if ladder statement.

4. nested if statement.

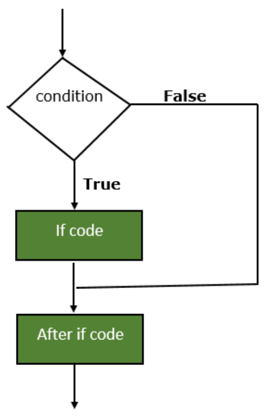
if Statement:

if statement in Java is a simple decision-making statement. Java if statement is used to decide whether a certain statement or block of the statement will be executed or not. If a certain condition is true, then the block of statement is executed, otherwise not.

Syntax:

```
if(condition)

{

 Statement(s);

}
```

Flowchart:



Example

```
public class IfStatementExample

 {

public static void main(String args[])

{

int num=70;

if( num < 100 )

{

System.out.println("number is less than 100");

}

}}
```

Output:

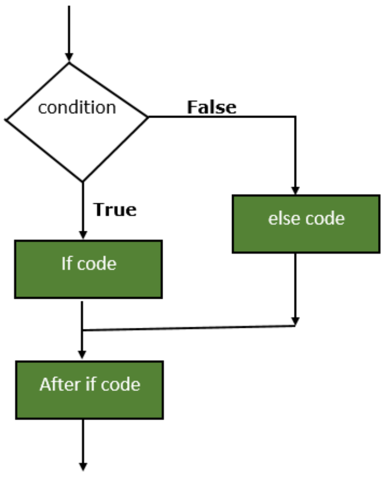
number is less than 100

if-else Statement:

if statement is used to decide whether a certain statement or block of the statement will be executed or not. If a certain condition is true, then the block of statement is executed, otherwise not. But, if you want to do something else if the condition is false then we should use else statement. We can use else statement with if statement to execute the block of code when the condition is false.

Syntax:

```
if(condition)

{

//code for if statement

}

else{

//code for else statement.

}
```



Example of if-else statement

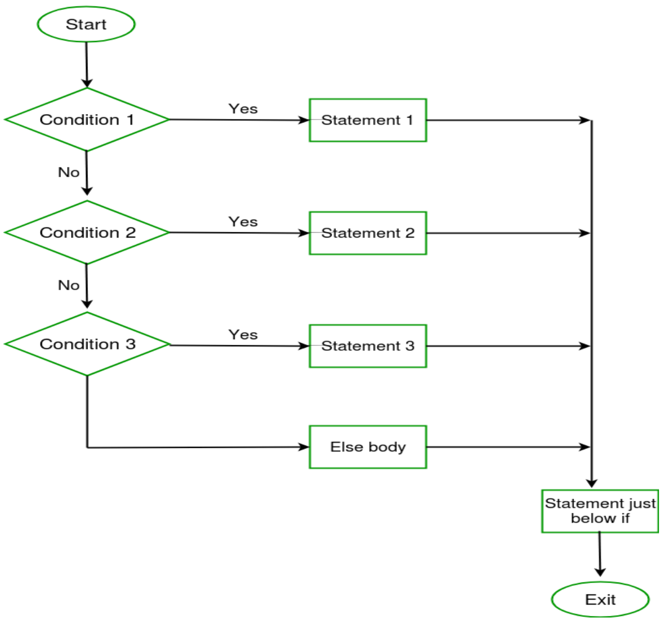
```
public class IfElseExample

{

public static void main(String args[])

{

int num=120;

if( num < 50 )

{

System.out.println("num is less than 50");
}
else
{
System.out.println("num is greater than or equal 50");
}}}
```

Output:

```
        num is greater than or equal 50
```

if-else-if Statement:

if-else-if statement is used when we need to check multiple conditions. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, If none of the conditions is true, then the final else statement will be executed. The final else acts as a default condition; that is, if all other conditional tests fail, then the last else statement is performed.



Syntax

```
if(condition1)
{
//statement1;
}
else if(condition2)
{
//statement2;
}
else if(condition3)
{
//statement3;
}
.........
```

```
.........
.........
Else
{
//default statement;
}
```

Example:

```
public class IfElseIfExample
 {
public static void main(String args[])
{
int num=1234;
if(num <100&&num>=1)
{
System.out.println("Its a two digit number");
}
else if(num <1000&&num>=100)
{
System.out.println("Its a three digit number");
}
else if(num <10000&&num>=1000)
{
System.out.println("Its a four digit number");
}
else if(num <100000&&num>=10000)
 {
System.out.println("Its a five digit number");
}
Else
{
System.out.println("number is not between 1 & 99999");
}
}
}
```

Output:

```
Its a four digit number
```

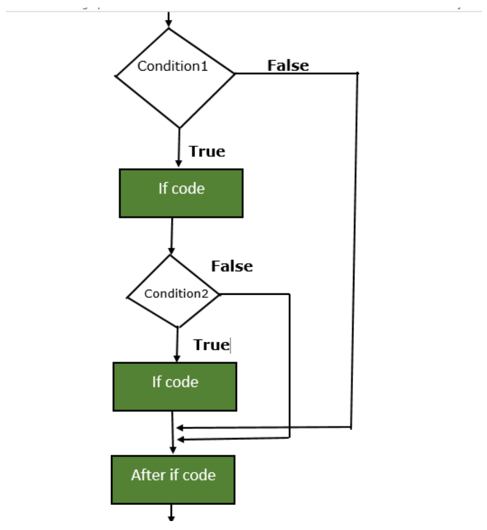
Nested if statement:

When there is an if statement inside another if statement then it is called the nested if statement.

Syntax:

```
if(condition1){

//code to be executed for condition1

}

if(condition2){

//code to be executed for condition2
```

}



Example

```
public class NestedIfExample
{
public static void main(String args[])
{
int num=70;
if( num < 100 )
{
System.out.println("number is less than 100");
if(num > 50)
{
System.out.println("number is greater than 50");
}}}}
```

Decision Making Statement:(Switch Case statement)

Switch case statement is used when we have number of options (or choices) and we may need to perform a different task for each choice.

A switch statement can work with byte, short, char, and int primitive data types. It also works with enumerated types and the String class.

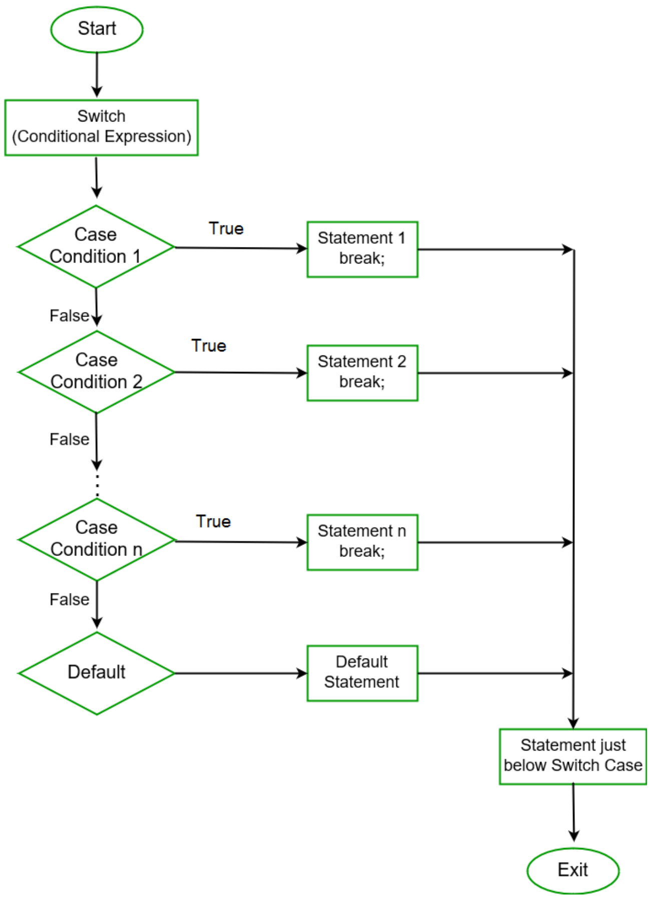
The value of the expression is compared with each of the literal values in the case statements.

If a match is found, the code sequence following that case statement is executed. If none of the constants matches the value of the expression, then the default statement is executed.
The default statement is optional.

The **break** statement is used inside the **switch** to terminate a statement sequence.

Syntax

```
switch(expression)
{
 case value1:
  //statement sequence
 break;
 case value2:
  //statement sequence
 break;
.......
.......
.......
 case valueN:
  //statement sequence
 break;
 default:
  //default statement sequence
}
```



Example

```
class WeekDays
{
    public static void main(String s[])

{

int day = 2;

 switch(day)
{
case 1:
                System.out.println("Monday");
                break;
 case 2:
                System.out.println("Tuesday");
                break;
  case 3:
                System.out.println("Wednesday");
                break;
   case 4:
                System.out.println("Thursday");
                break;
  case 5:
                System.out.println("Friday");
                break;
   case 6:
                System.out.println("Saturday");
                break;
    case 7:
                System.out.println("Sunday");
                          break;
default:
                System.out.println("Invalid");
                          break;
}}}
```

Iteration statement (Looping Statement):

Java Iteration statements allow program execution to repeat one or more statements. java Iteration statements provide different type of statements such as:

1. For loop.
2. While loop.
3. Do While loop.

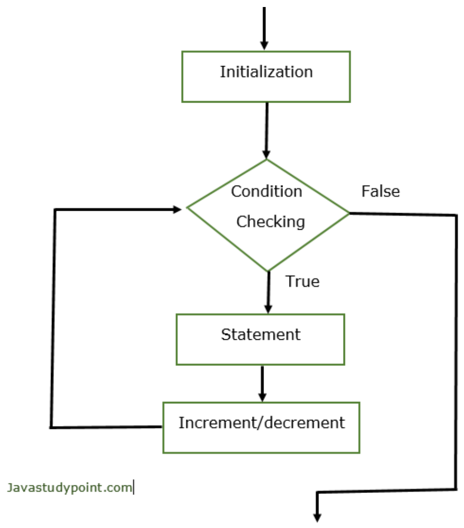
1. For loop.
Loops are used to execute a set of statements repeatedly until a particular condition is satisfied.
The for loop in java is used to iterate a part of the program several times.
The Java for loop consumes initialization, condition, and increment/decrement in one line.
Syntax

```
for(initialization; condition; incr/decr){

//body of the loop.

}
```

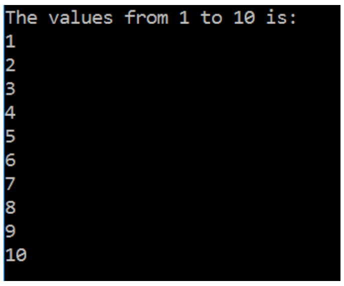
initialization: The initialization expression initializes the loop;

condition: In this step, we can check the condition. If the condition is true, then the body of the loop is executed otherwise not.

increment/decrement: In this step, we can increment or decrement the value of a variable.



```
class ForLoop
{
        public static void main(String args[]){
                System.out.println("The values from 1 to 10 is: ");

                // initialization; condition; incr/decr.
                for(int i = 1; i<=10; i++)
{
                        System.out.println(i);
                }
        }
}
```



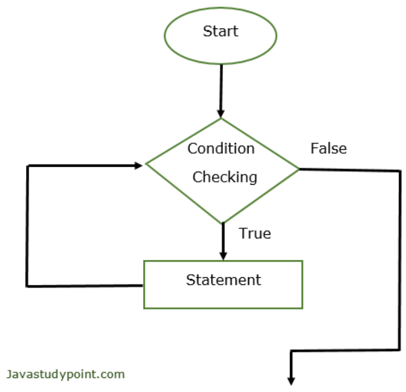
```
The values from 1 to 10 is:
1
2
3
4
5
6
7
8
9
10
```

While Loop:

The while loop in java is the most fundamental looping statement. Java while loop repeats a statement or a block while a particular condition is true. You can use a while loop if the number of iteration is not fixed.

Syntax

```
while(boolean condition)
{
//body of the loop.
}
```



```
class WhileLoopExample
{
public static void main(String args[])
{
int i=10;
while(i>1)
{
System.out.println(i);
i--;
}}}
```

Output:

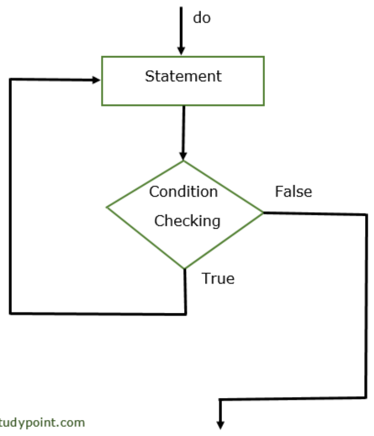
```
10
9
8
7
6
5
4
3
```

2

Do While loop:

do-while loop is similar to while loop. In while loop, condition is evaluated before the execution of loop's body but in do-while loop condition is evaluated after the execution of loop's body.

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



```
class DoWhileLoopExample
 {
 public static void main(String args[])
{
 int i=10;
do
{
 System.out.println(i);
 i--;
 }
while(i>1);
 } }
```

Output:

```
10
 9
 8
 7
 6
 5
 4
 3
 2
```

Jump Statements:

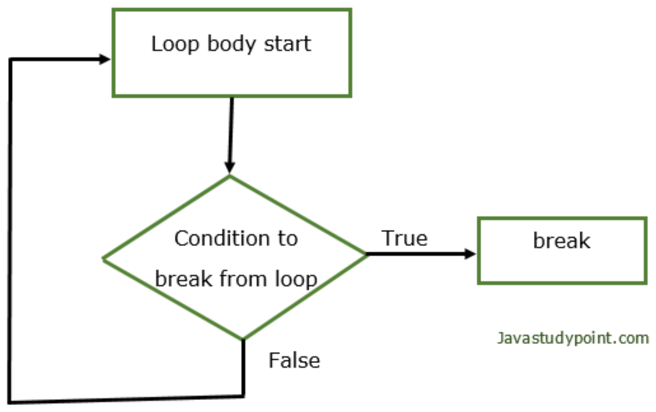
Java Jump statements transfer the control to other parts of the program. java Jump statements provide different type of statements such as:

1. break statement.

2. continue statement

break statement

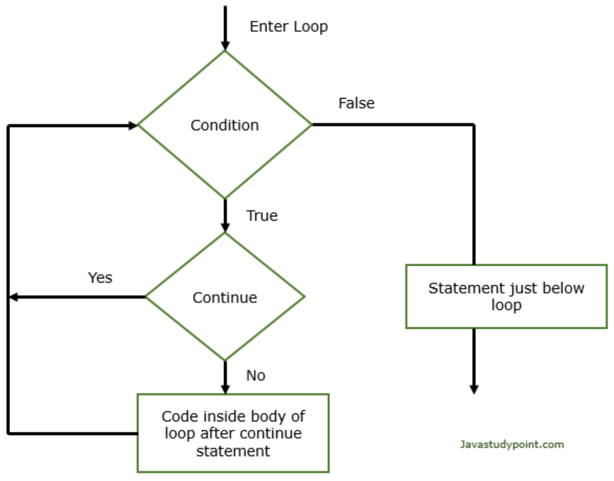
Java break statement can be used to terminate the loop. It can be used inside a loop. We can use a break statement in all types of loops such as: while loop, do-while loop, and for loop. When a break statement is encountered inside a loop, the loop is terminated and the program control goes to the next statement following the loop.



```
public class BreakExample3
{
public static void main(String args[])
{
int num=2;
switch (num)
{
case 1:
System.out.println("Case 1 ");
break;
case 2:
System.out.println("Case 2 ");

break;
case 3:
System.out.println("Case 3 ");
break;
default:
System.out.println("Default ");
}}}
```

continue statement:

Continue statement is mostly used inside loops. Whenever it is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop’s body for the current iteration.


Enter Loop
Condition
False
True
Yes
Continue
No
Statement just below loop
Code inside body of loop after continue statement
Javastudypoint.com

```
public class ContinueExample
{
public static void main(String args[])
{
for (int j=0; j<=6; j++)
{
if (j==4)
{
continue;
}
System.out.print(j+" ");
}}}
```

Output: 0 1 2 3 5 6

Introducing Classes:

Class fundamentals

How to create classes by using the following basics:

- The parts of a class definition
- Declaring and using instance variables
- Defining and using methods
- Creating Java applications, including the main() method and how to pass arguments to a Java program from a command line

What is a class in Java?

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity.

A class in Java can contain:

- ✓ Fields
- ✓ Methods
- ✓ Constructors
- ✓ Blocks
- ✓ Nested class and interface

Syntax to declare a class:

```
Class
 {
        field;
        method;
}
```

Declaration of Object and its initialization

To create an object of the class we use new keyword as shown below syntax

```
        classname objectname=new classname( );
```

3 Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable

2. By method

3. By constructor

Object and Class Example: Initialization through reference:

```
class Student

{

int id;

String name;

}

class TestStudent2

{

public static void main(String args[])

{

Student s1=new Student();

s1.id=101;

s1.name="Sonoo";

System.out.println(s1.id+" "+s1.name);//printing members with a white space

} }
```

Output: 101 Sonoo

Object and Class Example: Initialization through method

```
class Student
{
int rollno;
String name;
void insertRecord(int r, String n)
{
rollno=r;
name=n;
}
void displayInformation(){System.out.println(rollno+" "+name);
}
}
class TestStudent4{
public static void main(String args[])
{
Student s1=new Student();
Student s2=new Student();
s1.insertRecord(111,"Karan");
s2.insertRecord(222,"Aryan");
s1.displayInformation();
s2.displayInformation();
}}
```

Output:

```
 111 Karan
222 Aryan
```

Object and Class Example: Initialization through a constructor

```
class Employee
{
int id;
String name;
float salary;
void insert(int i, String n, float s)
{
id=i;
name=n;
salary=s;
}
void display()
{
System.out.println(id+" "+name+" "+salary);
}
}
public class TestEmployee {
public static void main(String[] args) {
Employee e1=new Employee();
Employee e2=new Employee();
Employee e3=new Employee();
e1.insert(101,"ajeet",45000);
e2.insert(102,"irfan",25000);
e3.insert(103,"nakul",55000);
e1.display();
e2.display();
e3.display();
}}
```

Output:

```
101 ajeet 45000.0
 102 irfan 25000.0
103 nakul 55000.0
```

DIFFERENCE BETWEEN OBJECT AND CLASS

OBJECT	CLASS
Object is an instance of a class.	Class is a blueprint or template from which objects are created.
Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a group of similar objects.

Object is a physical entity.	Class is a logical entity.
Object is created through new keyword e.g. Student s1=new Student();	Class is declared using class keyword e.g. class Student{}
Object is created many times as per requirement.	Class is declared once.
Object allocates memory when it is created.	Class doesn't allocated memory when it is created.

Method in Java

A method is a block of statements. A method is like a function which is used to expose the behavior of an object. Every method is used to perform a specific task. a method defined as a behavior of an object.

Every method in java must be declared inside a class.

Every method declaration has the following characteristics.

- **returnType -** Specifies the data type of a return value.
- **name -** Specifies a unique name to identify it.
- **parameters -** The data values it may accept or recieve.
- **{ } -** Defienes the block belongs to the method.

Advantage of Method

- ✓ Code Reusability
- ✓ Code Optimization

Creating a method

A method is created inside the class and it may be created with any access specifier. However, specifying access specifier is optional.

Syntax:

```
class <ClassName>

{

   <accessSpecifier> <returnType> <methodName>( parameters )

{

       ...

       block of statements;

       ...

   }}
```

//Java Program to demonstrate the working of a banking-system

//where we deposit and withdraw amount from our account. //Creating an Account class which has deposit() and withdraw() methods

```
class Account

{

int acc_no;
```

```
String name;
float amount;
//Method to initialize object
void insert(int a,String n,float amt){
acc_no=a;
name=n;
amount=amt;
}
//deposit method
void deposit(float amt){
amount=amount+amt;
System.out.println(amt+" deposited");
}
//withdraw method
void withdraw(float amt){
if(amount<amt){
System.out.println("Insufcient Balance");
}else{
amount=amount-amt;
System.out.println(amt+" withdrawn");
} }
//method to check the balance of the account
void checkBalance(){System.out.println("Balance is: "+amount);}
//method to display the values of an object
void display(){System.out.println(acc_no+" "+name+" "+amount);}
}
//Creating a test class to deposit and withdraw amount
class TestAccount{
```

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

Account a1=new Account();

a1.insert(832345,"Ankit",1000);

a1.display();

a1.checkBalance();

a1.deposit(40000);

a1.checkBalance();

a1.withdraw(15000);

a1.checkBalance();

}}
```

Output:

```
832345 Ankit 1000.0

Balance is: 1000.0

 40000.0 deposited

Balance is: 41000.0

15000.0 withdrawn

 Balance is: 26000.0
```

String Handling

A string is a sequence of characters surrounded by double quotations. In a java programming language, a string is the object of a built-in class. The string values are organized as an array of a character data type.

The String class defined in the package java.lang package. The String class implements Serializable, Comparable, and CharSequence interfaces.

Creating String object in java

we can use the following two ways to create a string object.

- Using string literal

Using new keyword

String literal:

In java, Strings can be created like this: Assigning a String literal to a String instance:

String str1 = "Welcome";

String str2 = "Welcome";

Using New Keyword

String str1 = new String("Welcome");

String str2 = new String("Welcome");

STRING HANDLING METHODS:

METHOD	**DESCRIPTION**	**RETURN VALUE**
compareTo(String)	Compares two strings	Int
concat(String)	Concatenates the object string with argument string.	String
toLowerCase()	Converts a string to lower case letters	String
toUpperCase()	Converts a string to upper case letters	String
replace(String, String)	Replaces the first string with second string	String

JAVA ACCESS MODIFIERS

In Java, the access specifiers (also known as access modifiers) used to restrict the scope or accessibility of a class, constructor, variable, method or data member of class and interface. There are four access specifiers, and their list is below.

- ✓ default (or) no modifier
- ✓ public
- ✓ protected
- ✓ private

The **public** members can be accessed everywhere.

The **private** members can be accessed only inside the same class.

The **protected** members are accessible to every child class (same package or other packages).

The **default** members are accessible within the same package but not outside the package.

Example:

```
class ParentClass
{
        int a = 10;
        public int b = 20;
        protected int c = 30;
        private int d = 40;
                void showData()
{
                System.out.println("Inside ParentClass");
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                System.out.println("d = " + d);
        }
}
class ChildClass extends ParentClass{
        void accessData() {
                System.out.println("Inside ChildClass");
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                //System.out.println("d = " + d); // private member can't be accessed
        }
        }
public class AccessModifiersExample {

        public static void main(String[] args) {

                ChildClass obj = new ChildClass();
                obj.showData();
                obj.accessData();

        }}
```

this keyword in java

Here is given the 6 usage of java this keyword

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method(implicitly)
3. this() can be used to invoke current class constructor
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

```
class Student
{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee)
{
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display( )
{
System.out.println(rollno+" "+name+" "+fee);
}
}
class TestThis2
{
public static void main(String args[])
{
Student s1=new Student(111,"ankit",5000f);
Student s2=newStudent(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
111 ankit 5000
112 sumit 6000
```

CONSTRUCTORS IN JAVA

- Constructors are used to initialize the state of an object when it is created.
- Constructors are invoked while creating objects, usually after the new keyword. Every time an Object is created using the new() keyword, at least one constructor is called.
- It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.

```
public class MyClass
{
//This is the constructor
MyClass()
{
} .. }
```

Rules for creating Java constructor:

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Default Constructor:

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

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

Example:

```
//Java Program to create and call a default constructor
class Bike1
{
//creating a default constructor
Bike1()
{
System.out.println("Bike is created");
}
//main method
public static void main(String args[])
{
//calling a default constructor
Bike1 b=new Bike1();
} }
```

Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide diferent values to distinct objects.

Example

```
class Student4
{
int id;
String name;
Student4(int i,String n)                    //creating a parameterized constructor
{
id = i;
name = n;
}
void display()
{
System.out.println(id+" "+name);
}
public static void main(String args[])               //creating objects and passing values
{
Student4 s1 = new Student4(111,"Karan");
Student4 s2 = new Student4(222,"Aryan");
s1.display();                  //calling method to display the values of object
s2.display();
}}
```

CONSTRUCTOR OVERLOADING

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task.

Example

```
class Student5
{
int id;
String name;
int age;
//creating two arg constructor
Student5(int i,String n)
{
id = i;
name = n;
}
//creating three arg constructor
```

COPY CONSTRUCTOR

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

//Java program to initialize the values from one object to another object.

```
class Student6
{
int id;
String name;
```
//constructor to initialize integer and string
```
Student6(int i,String n)
{
id = i;
name = n;
}
```

//constructor to initialize another object
```
Student6(Student6 s)
{
id = s.id;
name =s.name;
}
void display( )
{
System.out.println(id+" "+name);
}
public static void main(String args[])
{
Student6 s1 = new Student6(111,"Karan");
Student6 s2 = new Student6(s1);
s1.display();
s2.display();
}}
```
Output:
```
 111 Karan
111 Karan
```

DIFFERENCE BETWEEN CONSTRUCTOR AND METHOD IN JAVA

CONSTRUCTOR	METHOD
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

ABSTRACT CLASSES

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). It needs to be extended and its method implemented.

Syntax:

```
abstract class classname

{

 }
```

ABSTRACT METHOD

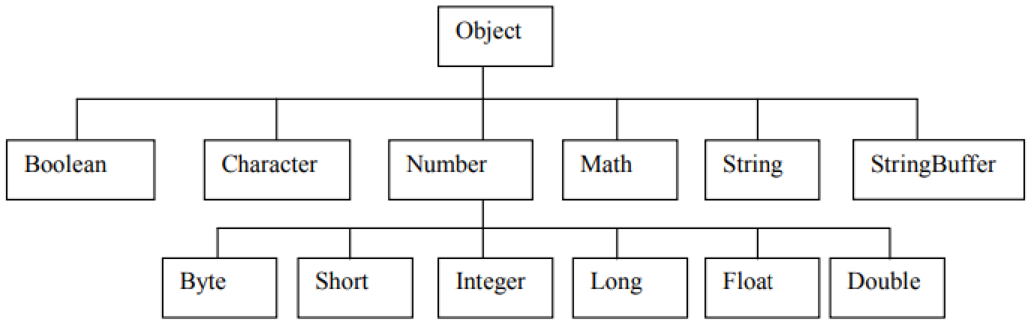
A method that is declared as abstract and does not have implementation is known as abstract method. The method body will be defined by its subclass.

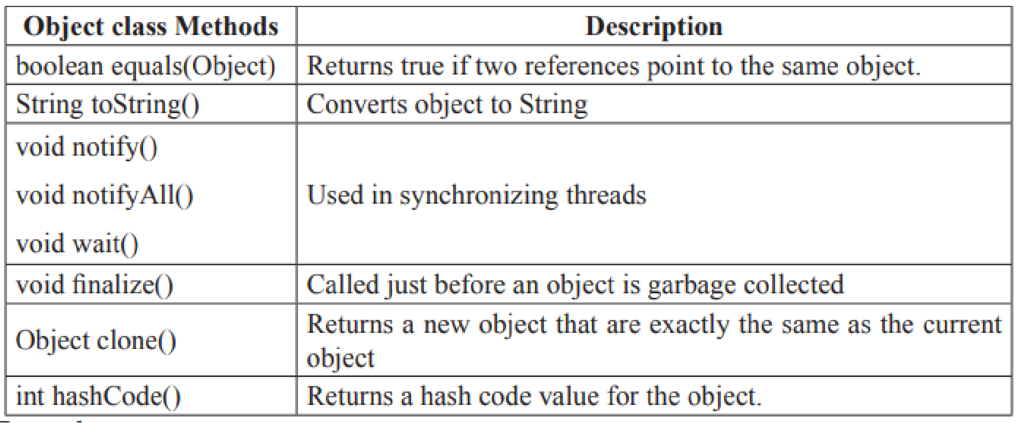
Syntax:

```
        abstract returntype functionname ();                    //No definition
```

THE OBJECT CLASS

The Object class is the parent class of all the classes in java by default (directly or indirectly). The java.lang.Object class is the root of the class hierarchy. Some of the Object class are Boolean, Math, Number, String etc.





Object class Methods	Description
boolean equals(Object)	Returns true if two references point to the same object.
String toString()	Converts object to String
void notify() void notifyAll() void wait()	Used in synchronizing threads
void finalize()	Called just before an object is garbage collected
Object clone()	Returns a new object that are exactly the same as the current object
int hashCode()	Returns a hash code value for the object.

GARBAGE COLLECTION

The Garbage Collection (GC) feature in the Java Virtual Machine (JVM). Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

Java garbage collection is an automatic process. The programmer does not need to explicitly mark objects to be deleted.

Here are some key points about Java Garbage Collection:

1. Automatic Memory Management:

Java developers don't need to explicitly free memory as in languages like C or C++. The garbage collector takes care of deallocating memory.

2. Heap Memory:

The memory allocated for Java objects is managed in the heap. The heap is divided into two main areas: the Young Generation and the Old Generation.

3. Generational Garbage Collection:

Java's garbage collector uses a generational garbage collection algorithm. Objects are initially created in the Young Generation. The objects that survive multiple garbage collection cycles in the Young Generation are eventually moved to the Old Generation.

4. Garbage Collector Implementations:

Java Virtual Machines may use different garbage collector implementations, such as Serial, Parallel, CMS (Concurrent Mark-Sweep), G1 (Garbage First), and more.

The choice of garbage collector depends on factors like application requirements, available hardware, and JVM version.

5. Tuning Garbage Collection:

Java provides options to tune garbage collection to meet specific application requirements. This can include setting parameters related to the heap size, garbage collector algorithm, and other options.

6. Monitoring and Analysis:

Various tools, like VisualVM, JConsole, and others, allow developers to monitor and analyze the behavior of the garbage collector and overall memory usage.

7. Manual Memory Management:

While developers don't explicitly manage memory, they can still influence garbage collection behavior by writing efficient code, avoiding memory leaks, and tuning JVM parameters.

8. Memory Leaks:

Despite garbage collection, memory leaks can occur if references to objects are unintentionally maintained, preventing them from being garbage collected. Developers need to be mindful of managing references appropriately.

Why is Java Garbage Collection Important?

Java Garbage Collection is essential for several reasons:

1. **Automation and simplification:** Automatic garbage collection in Java takes the burden off developers. In contrast, languages like C or C++ require explicit memory allocation and deallocation, which can be error-prone and lead to crashes if not handled properly.
2. **Increased Developer Productivity:** With automatic memory management, developers can focus on writing application logic, leading to faster development cycles.
3. **Preventing OutOfMemoryError:** By automatically tracking and removing unused objects, garbage collection prevents memory-related errors like OutOfMemoryError errors.
4. **Eliminates dangling pointer bugs:** These are bugs that occur when a piece of memory is freed while there are still pointers to it, and one of those pointers is dereferenced. By then the memory may have been reassigned to another use with unpredictable results.
5. **Eliminates Double free bugs:** These happen when the program tries to free a region of memory that has already been freed and perhaps already been allocated again.

POLYMORPHISM

Polymorphism in Java is a concept by which we can perform a single action in 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:

1. Compile-time polymorphism or Ad hoc polymorphism
2. Runtime polymorphism. Or Pure polymorphism

Compile time Polymorphism (or Static polymorphism)

Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile time polymorphism.

Method Overloading:

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

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) foe two parameters, and b(int,int,int) for three parameters then it may be difficult foe you as well as other programmers to understand the behavior of the method because its name differs.

Example:

```
class SimpleCalculator
{
int add(int a, int b)
{
return a+b;
}
int add(int a, int b, int c)
{
return a+b+c;
}}
public class Demo
{
public static void main(String args[])
{
SimpleCalculator obj = new SimpleCalculator();
System.out.println(obj.add(10, 20));
System.out.println(obj.add(10, 20, 30));
}}
```

Output:

```
30
60
```

Runtime Polymorphism (or Dynamic 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.

Method Overriding:

If subclass (child class) has the same method as declaeed in the paeent class, it is known as method overriding in Java.

Example

```
//Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle
{
//defining a method
void run()
{
System.out.println("Vehicle is running");
}}
//Creating a child class
class Bike2 extends Vehicle
{
//defining the same method as in the parent class
void run()
{
System.out.println("Bike is running safely");
}
public static void main(String args[])
{
Bike2 obj = new Bike2();                    //creating object
obj.run();                                  //calling method
}}
```

Output: Bike is running safely

DIFFERENCE BETWEEN METHOD OVERLOADING AND METHOD OVERRIDING

METHOD OVERLOADING	METHOD OVERRIDING
Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
Method overloading helps to increase the readability of the program.	Method overriding is used to grant the specific implementation of the method which is already provided by its parent class or super class.
It occurs within the class.	It is performed in two classes with inheritance relationships.
Method overloading may or may not require inheritance.	Method overriding always needs inheritance.
Static binding is being used for overloaded methods.	Dynamic binding is being used for overriding methods.
Private and final methods can be overloaded.	Private and final methods can’t be overridden.

STATIC VS DYNAMIC BINDING

STATIC BINDING	DYNAMIC BINDING
It happens at the compile time.	It happens at the run time.
It is also called early binding.	It is also called late binding.
It can be achieved during normal function calls, function overloading and operator overloading.	It is achieved with the use of virtual functions.
Execution becomes faster than dynamic binding because the function call gets resolved before run time.	As the function call is resolved at run time, sometimes it leads to slower code execution.
It provides less flexibility compared to the dynamic binding.	It provides more flexibility as different types of objects at runtime can be handled by a single function call making the source code more readable.

PARAMETER PASSING METHODS

Parameter passing refers to the process of providing input values to a function or method when it is called during program execution. In programming, parameters are variables or values that are used to pass information into a function or method.

1. Pass by value
2. Pass by reference

Pass by value:

In pass-by-value, the actual value of the parameter is passed to the function or method. This means that changes made to the parameter inside the function or method do not affect the original value in the calling code.

This method is commonly used for primitive data types like integers, floats, and booleans.

Example:

```
class Swapper
{
	int a;
	int b;
	Swapper(int x, int y)
	{
		a = x;
		b = y;
	}
	void swap(int x, int y)
	{
		int temp;
		temp = x;
		x = y;
		y = temp;
	}
}
class SwapDemo
{
	public static void main(String[] args)
	{
	Swapper obj = new Swapper(10, 20);
	System.out.println("Before swapping value of a is "+obj.a+" value of b is "+obj.b);
	obj.swap(obj.a, obj.b);
	System.out.println("After swapping value of a is "+obj.a+" value of b is "+obj.b);
	}
}
```

Output:

```
	Before swapping value of a is 10 value of b is 20
	After swapping value of a is 10 value of b is 20
```

Pass-by-Reference:

In pass-by-reference, a reference or memory address of the parameter is passed to the function or method. This allows the function or method to access and modify the actual data in the calling code.

This method is commonly used for objects and complex data structures.

```
class Swapper

{

int a;

int b;

Swapper(int x, int y)

{

	a = x;

	b = y;
```

```
        }
        void swap(Swapper ref)
        {
        int temp;
        temp = ref.a;
        ref.a = ref.b;
        ref.b = temp;
        }
}
class SwapDemo
{
public static void main(String[] args)
        {
        Swapper obj = new Swapper(10, 20);
        System.out.println("Before swapping value of a is "+obj.a+" value of b is "+obj.b);
        obj.swap(obj);
        System.out.println("After swapping value of a is "+obj.a+" value of b is "+obj.b);
        }}
```

Output:

```
        Before swapping value of a is 10 value of b is 20
        After swapping value of a is 20 value of b is 10
```

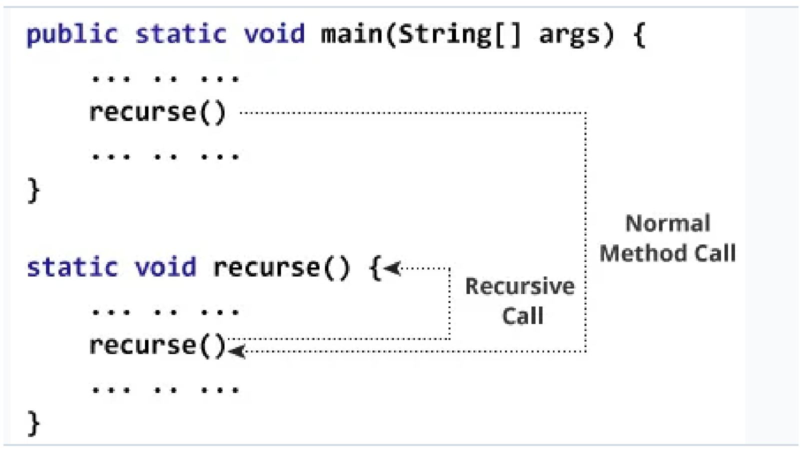
RECURSION:

What Is Recursion In Java?

Recursion is a process by which a function or a method calls itself again and again. This function that is called again and again either directly or indirectly is called the “recursive function”.

Syntax:

```
returntype methodname()
{                                   //code to be executed
methodname();           //calling same method
}
```



```
public static void main(String[] args) {
    ... .. ...
    recurse()
    ... .. ...
}

static void recurse() {
    ... .. ...
    recurse()
    ... .. ...
}
```

In the above example, we have called the recurse() method from inside the main method. (normal method call). And, inside the recurse() method, we are again calling the same recurse method. This is a recursive call.

Example

```
public class RecursionExample
 {
 static int factorial(int n){
          if (n == 1)
            return 1;
          else
            return(n * factorial(n-1));
    }

public static void main(String[] args) {
System.out.println("Factorial of 5 is: "+factorial(5));
}
}
```

Output:

```
Factorial of 5 is: 120
```

Example (Fibonacci Series)

```
public class RecursionExample
{
    static int n1=0,n2=1,n3=0;
     static void printFibo(int count)
{
        if(count>0){
             n3 = n1 + n2;
             n1 = n2;
             n2 = n3;
             System.out.print(" "+n3);
             printFibo(count-1);
         }     }
public static void main(String[] args)
{
    int count=15;
      System.out.print(n1+" "+n2);             //printing 0 and 1
      printFibo(count-2);                //n-2 because 2 numbers are already printed
}  }
```

Java Nested and Inner Class

The Java language allows you to define a class within another class.

```
class OuterClass
{                   // ...
   class NestedClass
{           // ...
   }}
```

Types of Nested Classes:

- Static Nested Class (Static Inner Class)
- Inner Class (Non-static Nested Class)
- Anonymous Inner Class

Static Nested Class (Static Inner Class)

- Declared with the static keyword.
- Can access static members of the outer class.
- Does not have access to the instance variables or methods of the outer class without an instance of the outer class.

```
public class OuterClass
{
   static class StaticNestedClass
{
      // Static nested class members
   }}
```

Inner Class (Non-static Nested Class):

- Associated with an instance of the outer class.
- Can access both static and instance members of the outer class.
- Has a reference to an instance of the outer class.

```
public class OuterClass {

   class InnerClass {

      // Inner class members

   }}
```

Anonymous Inner Class:

A type of local inner class without a name.

Often used for one-time use, such as implementing an interface or extending a class.

```
public class OuterClass
{
   interface MyInterface
{
      void myMethod();
   }
 void myMethod()
 {
      MyInterface myInterface = new MyInterface()
{
         public void myMethod()
{
            // Implementation
        }
    };   }}
```

EXPLORING STRING CLASS:
Java String Class Methods

String is a sequence of characters. For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'. We use double quotes to represent a string. The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.

String Operations

Method	Description
char charAt(int index)	It returns char value for the particular index
int length()	It returns string length
static String format(String format, Object... args)	It returns a formatted string.
static String format(Locale l, String format, Object... args)	It returns formatted string with given locale.
String substring(int beginIndex)	It returns substring for given begin index.
String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
boolean equals(Object another)	It checks the equality of string with the given object.
boolean isEmpty()	It checks if string is empty.
String concat(String str)	It concatenates the specified string.
String replace(char old, char new)	It replaces all occurrences of the specified char value.
String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
String[] split(String regex)	It returns a split string matching regex.
String[] split(String regex, int limit)	It returns a split string matching regex and limit.
String intern()	It returns an interned string.
int indexOf(int ch)	It returns the specified char value index.
int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
int indexOf(String substring)	It returns the specified substring index.
int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.

String toLowerCase()	It returns a string in lowercase.
String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
String toUpperCase()	It returns a string in uppercase.
String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
String trim()	It removes beginning and ending spaces of this string.
static String valueOf(int value)	It converts given type into string. It is an overloaded method.

charAt() method

String charAt() function returns the character located at the specified index.

```
public class Demo
 {
   public static void main(String[] args) {
      String str = "studytonight";
      System.out.println(str.charAt(2));
   }}
```

Output: u

NOTE: Index of a String starts from 0, hence str.charAt(2) means third character of the String str.

equalsIgnoreCase() method

String equalsIgnoreCase() determines the equality of two Strings, ignoring their case (upper or lower case doesn't matter with this method).

```
public class Demo
{
   public static void main(String[] args)
{
      String str = "java";
      System.out.println(str.equalsIgnoreCase("JAVA"));
   }}
```

Output:

```
True
```

concat() method:

We can join two strings in Java using the concat() method. The concat() method joins the second string to the first string and assigns it to the joinedString variable.
We can also join two strings using the + operator in Java. To learn more, visit Java String concat().

```
class Main
{
  public static void main(String[] args)
 {
   // create first string
   String first = "Java ";
   System.out.println("First String: " + first);
   // create second
   String second = "Programming";
   System.out.println("Second String: " + second);
   // join two strings
   String joinedString = first.concat(second);
   System.out.println("Joined String: " + joinedString);
  }
}
```

Output

```
First String: Java
Second String: Programming
Joined String: Java Programming
```

Equals() Method:

we can make comparisons between two strings using the equals() method.

```
class Main
 {
  public static void main(String[] args) {
    // create 3 strings
    String first = "java programming";
    String second = "java programming";
    String third = "python programming";

    // compare first and second strings
    boolean result1 = first.equals(second);
    System.out.println("Strings first and second are equal: " + result1);
    // compare first and third strings
    boolean result2 = first.equals(third);
    System.out.println("Strings first and third are equal: " + result2);
  }}
```

length() method:

To find the length of a string, we use the length() method of the String.

```
public class Demo {
 public static void main(String[] args) {
 String str = "Count me";
 System.out.println(str.length());
    }}
```

replace() method

String replace() method replaces occurances of character with a specified new character.

```
public class Demo {
    public static void main(String[] args) {
        String str = "Change me";
        System.out.println(str.replace('m','M'));
    }}
```

OUTPUT:

```
 Change Me
```

substring() method

String substring() method returns a part of the string. substring() method has two override methods.

1. public String substring(int begin);
2. public String substring(int begin, int end);

```
public class Demo {
    public static void main(String[] args) {
        String str = "0123456789";
        System.out.println(str.substring(4));
        System.out.println(str.substring(4,7));
    }}
```

OUTPUT:

```
        Abcdef
```

toUpperCase() method

This method returns string with all lowercase character changed to uppercase.

```
public class Demo {
  public static void main(String[] args) {
     String str = "abcdef";
     System.out.println(str.toUpperCase());
  }}
```

OUTPUT:

```
ABCDEF
```

valueOf() method

String class uses overloaded version of valueOf() method for all primitive data types and for type Object.

NOTE: valueOf() function is used to convert primitive data types into Strings.

```
public class Demo {
  public static void main(String[] args) {
      int num = 35;
      String s1 = String.valueOf(num);    //converting int to String
      System.out.println(s1);
      System.out.println("type of num is: "+s1.getClass().getName());
  }}
```

OUTPUT:

```
        35
type of num is: java.lang.String
```

toString() method

String toString() method returns the string representation of an object. It is declared in the Object class, hence can be overridden by any java class. (Object class is super class of all java classes).

```
public class Car {
  public static void main(String args[])
  {
     Car c = new Car();
     System.out.println(c);
  }
  public String toString()
  {
     return "This is my car object";
  }}
```

OUTPUT:

```
This is my car object
```

trim() method

This method returns a string from which any leading and trailing whitespaces has been removed.

```
public class Demo {
  public static void main(String[] args) {
     String str = "   hello  ";
     System.out.println(str.trim());
  }}
```

OUTPUT:

```
Hello
```

contains()Method

String contains() method is used to check the sequence of characters in the given string. It returns true if a sequence of string is found else it returns false.

```
public class Demo {
  public static void main(String[] args) {
      String a = "Hello welcome to studytonight.com";
```

```
        boolean b = a.contains("studytonight.com");
        System.out.println(b);
        System.out.println(a.contains("javatpoint"));
    }}
```

OUTPUT:

```
true
false
```

endsWith() Method

String endsWith() method is used to check whether the string ends with the given suffix or not. It returns true when suffix matches the string else it returns false.

```
public class Demo {
    public static void main(String[] args) {
        String a="Hello welcome to studytonight.com";
        System.out.println(a.endsWith("m"));
        System.out.println(a.endsWith("com"));
    }}
```

OUTPUT:

```
true
true
```

format() Method

String format() is a string method. It is used to the format of the given string.

```
public class Demo {
    public static void main(String[] args) {
        String a1 = String.format("%d", 125);
        String a2 = String.format("%s", "studytonight");
        String a3 = String.format("%f", 125.00);
        String a4 = String.format("%x", 125);
        String a5 = String.format("%c", 'a');
        System.out.println("Integer Value: "+a1);
        System.out.println("String Value: "+a2);
        System.out.println("Float Value: "+a3);
        System.out.println("Hexadecimal Value: "+a4);
        System.out.println("Char Value: "+a5);
    }}
```

OUTPUT:

```
Integer Value: 125
String Value: studytonight
Float Value: 125.000000
Hexadecimal Value: 7d
Char Value: a
```

getChars() Method

String getChars() method is used to copy the content of the string into a char array.

```
public class Demo {
    public static void main(String[] args) {
        String a= new String("Hello Welcome to studytonight.com");
        char[] ch = new char[16];
        try
        {
            a.getChars(6, 12, ch, 0);
            System.out.println(ch);
        }
        catch(Exception ex)
        {
            System.out.println(ex);
        }  }}
```

OUTPUT:

```
Welcom
```

isEmpty() Method

String isEmpty() method is used to check whether the string is empty or not. It returns true when length string is zero else it returns false.

```
public class IsEmptyDemo1
{
    public static void main(String args[])
    {
        String a="";
        String b="studytonight";
        System.out.println(a.isEmpty());
        System.out.println(b.isEmpty());
    }}
```

OUTPUT:

```
true
false
```

join() Method

String join() method is used to join strings with the given delimiter. The given delimiter is copied with each element

```
public class JoinDemo1
{
    public static void main(String[] args)
    {
    String s = String.join("*","Welcome to studytonight.com");
    System.out.println(s);
        String date1 = String.join("/","23","01","2020");
System.out.println("Date: "+date1);
        String time1 = String.join(":", "2","39","10");
System.out.println("Time: "+time1);
    }  }
```

OUTPUT:

```
Welcome to studytonight.com
Date: 23/01/2020
Time: 2:39:10
```

StartsWith() Method

String startsWith() is a string method in java. It is used to check whether the given string starts with given prefix or not. It returns true when prefix matches the string else it returns false.

```
public class Demo {
    public static void main(String[] args) {
          String str = "studytonight";
          System.out.println(str.startsWith("s"));
          System.out.println(str.startsWith("t"));
          System.out.println(str.startsWith("study",1));
    }}
```

OUTPUT:

```
true
false
false
```