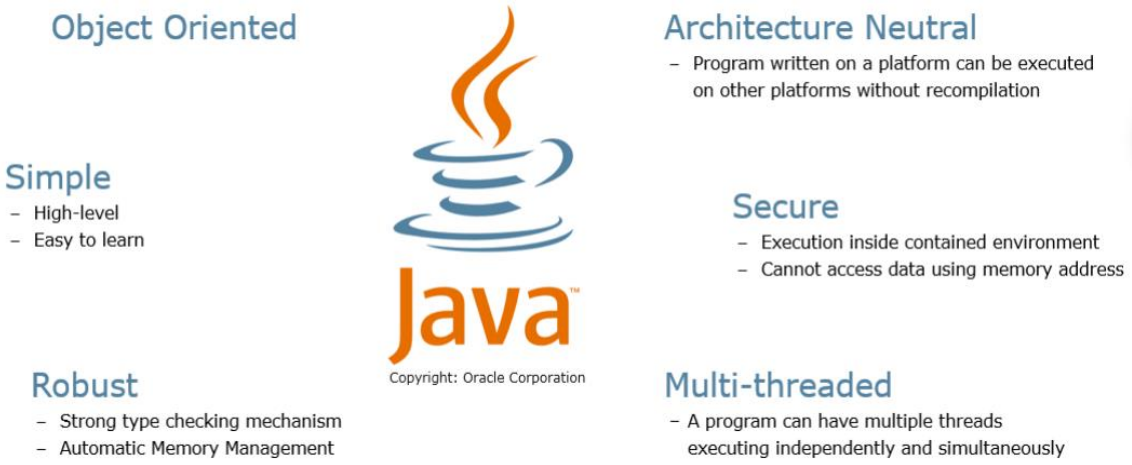
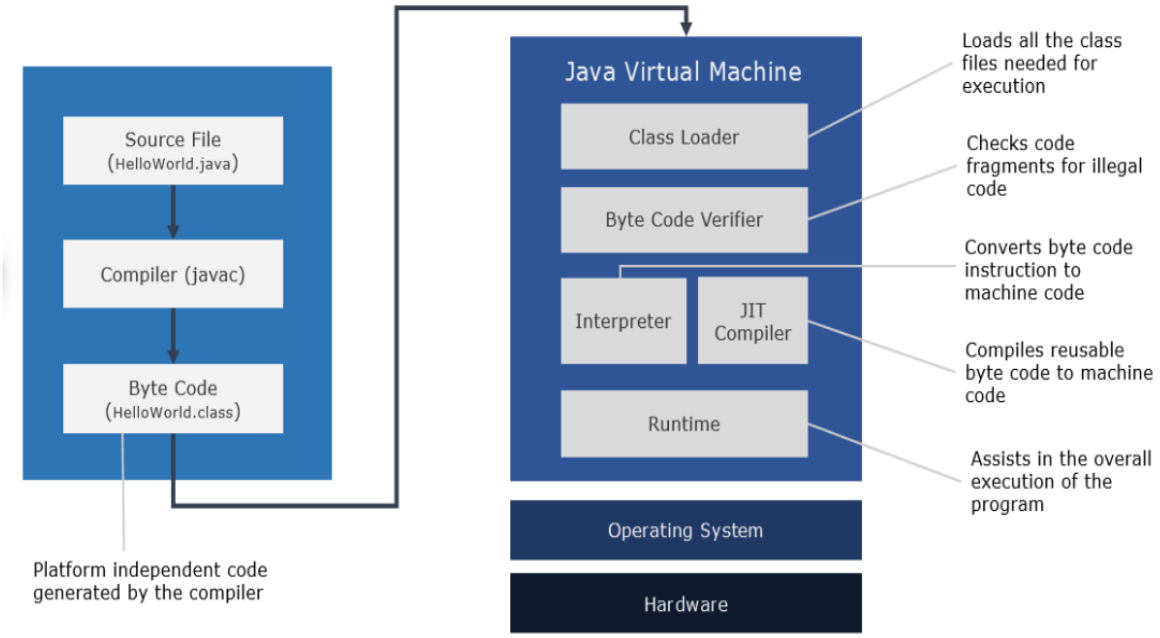
- Java is a programming language and a platform.
- Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by Sun Microsystems in the year 1995.
- James Gosling is known as the father of Java.
- Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

FEATURES OF JAVA

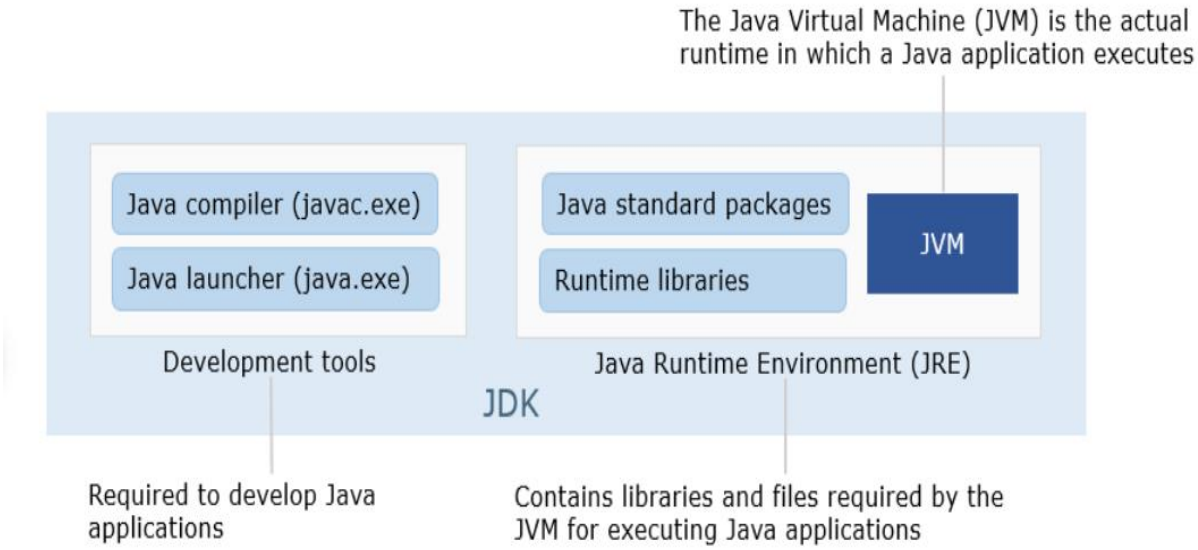
1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic



- In Java, the program (**source code**) written by the programmer gets compiled and converted into **byte code** (compiled code) by the Java compiler.
- All the byte codes required for the program are then given to the interpreter. The interpreter reads the byte code line by line, converts it to binary form, also called as machine language or binary language, and then executes it.
- The Java source code is saved in a file with .java extension. When we compile a Java program (.java file), .class files (byte code) with the same class names present in .java file are generated by the Java compiler (javac). These .class files go through various steps when we run the program as shown in the below diagram.



- The development, compilation and execution of Java programs is taken care by JDK which contains 2 main components: Development tools and JRE.



- The development tools consist of Java compiler and Java launcher.

- Java compiler (**javac.exe**) - It is the primary Java compiler. The compiler accepts Java source code and produces Java bytecode conforming to the Java Virtual Machine Specification (JVMS).
- Java launcher (**java.exe**) - It helps in launching a Java application during execution.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

First Java Program

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

Output: - Hello Java

Parameters used in First Java Program

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

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.

- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

Valid Java main() method signature

1. **public static void** main(String[] args)
2. **public static void** main(String []args)
3. **public static void** main(String args[])
4. **public static void** main(String... args)
5. **static public void** main(String[] args)
6. **public static final void** main(String[] args)
7. **final public static void** main(String[] args)
8. **final strictfp public static void** main(String[] args)

Java Variables

- A variable is a named memory location which holds some value.
- The value stored in a variable can vary during the execution of the program.

There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

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

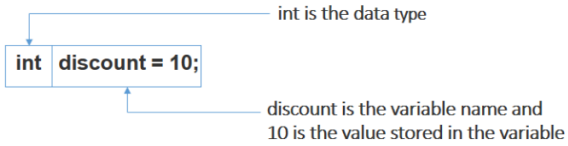
It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

You can declare variables in Java as follows:

E.g. -



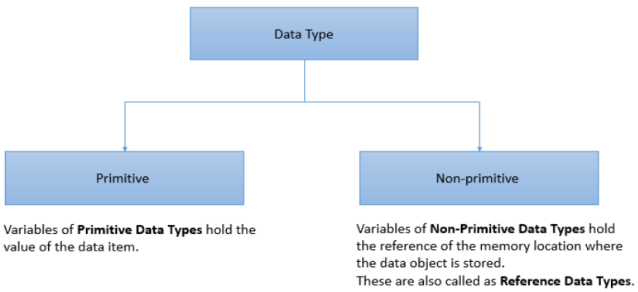
IDENTIFIER: -

Similarly, in Java, an **identifier** is the name given to a variable, method or class to uniquely identify it.

In Java, following rules apply to the identifier name:

- It can contain alphanumeric characters([a-z], [A-Z], [0-9]), dollar sign ($), underscore (_)
- It should not start with a digit ([0-9])
- It should not have spaces.
- It should not be a Java keyword.
- It is case-sensitive.
- It has no length restrictions.

DATA TYPES: -



Primitive data types are the basic data types defined in Java. There are 8 primitive data types as shown in the below table.

Data type	Description	Default	Size	Range	Example
boolean	stores true or false	false	1 bit	true or false	boolean v = true;
byte	stores whole numbers and is used for small values	0	8 bits	-2^7 to 2^7-1	byte a = 10;
char	stores a single Unicode character	\u0000	16 bits	'\u0000' to '\uffff'	char c = 'A';
short	stores whole numbers and is used for smaller values	0	16 bits	-2^{15} to $2^{15}-1$	short s = 10;
int	stores whole numbers and is the preferred data type for numeric values	0	32 bits	-2^{31} to $2^{31}-1$	int x = 1000;
long	stores whole numbers and is used for large values that int cannot store	0	64 bits	-2^{63} to $2^{63}-1$	long a = 100000L;
float	stores fractional numbers with 6-7 decimal digits	0.0	32 bits	1.4e-045 to 3.4e+038	float f = 23.5f;
double	stores fractional numbers with up to 15 decimal digits	0.0	64 bits	4.9e-324 to 1.8e+308	double d = 32.5;

Here are some of the important points about primitive data types:

1. Numeric and Boolean (true, false) values are written without quotes. E.g. int score = 85; boolean isQualified = true;
2. The character value must be written in single quotes while assigning it to a character variable. E.g. char gender = 'M';
3. A long value is assigned to the variable, suffixed with L (uppercase letter or lower case letter L can be used). E.g. long salary = 500000L;
4. A float value must be suffixed with F or f while assigning to the variable. E.g. float average = 78.6f;

Non-primitive data types include classes, arrays, interfaces and strings etc.

CODING STANDARDS: -

In Java, variable names should be nouns starting with lowercase letter. If it contains multiple words, then every inner word must start with capital letter. This type of casing is called camel casing.

Some examples of variables are given below:

- mobileNumber
- name
- unitPrice
- paymentMode
- age

Operators in Java

Operators are the symbols used to perform specific operations.

The operators are categorized as:

- Unary
- Arithmetic
- Relational
- Logical
- Ternary
- Assignment
- Bitwise

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc.

Object-Oriented Programming is a methodology to design a program using classes and objects.

It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. This can be physical or logical.
- An Object can also be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance of a class.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

Class

Collection of objects is called class.

A class is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. Class doesn't consume any space.

A class in Java can contain:

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

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability.

Polymorphism

If *one task is performed in different ways*, it is known as polymorphism.

- In Java, we use method overloading and method overriding to achieve polymorphism.

Abstraction

Hiding internal details and showing functionality is known as abstraction.

For example, phone call, we don't know the internal processing.

- In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Identifiers Type	Naming Rules	Examples

Class	It should start with the uppercase letter.
	It should be a noun such as Color, Button, System, Thread, etc.
	Use appropriate words, instead of acronyms.
		public class **Employee**
		{
		//code snippet
		}

Interface	It should start with the uppercase letter.
	It should be an adjective such as Runnable, Remote, ActionListener.
	Use appropriate words, instead of acronyms.
		interface **Printable**
		{
		//code snippet
		}

Method	It should start with lowercase letter.
	It should be a verb such as main(), print(), println().
	If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
		class Employee
		{
		// method
		void **draw()**
		{
		//code snippet
		}
		}

Variable	It should start with a lowercase letter such as id, name.
	It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore).
	If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
	Avoid using one-character variables such as x, y, z.
		class Employee
		{
		// variable
		int **id**;
		//code snippet
		}

Package	It should be a lowercase letter such as java, lang.
	If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.
		//package
		package **com.javatpoint;**
		class Employee
		{
		//code snippet
		}

Constant	It should be in uppercase letters such as RED, YELLOW.
	If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY.
	It may contain digits but not as the first letter.
		class Employee
		{
		//constant
		static final int **MIN_AGE** = 18;
		//code snippet
		}

Java Naming Convention

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

But, it is not forced to follow. So, it is known as convention not a rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.

Method in Java

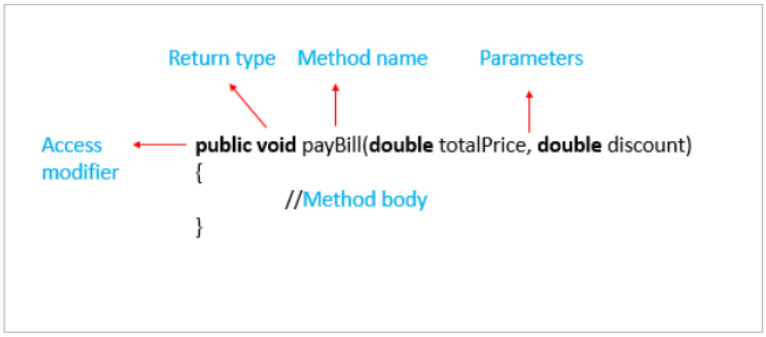
In Java, a method is like a function which is used to expose the behavior of an object.

- In Java, every method must be a part of some class.

Advantage of Method

- Code Reusability
- Code Optimization

Method syntax:



Access modifier – defines the access type of the method. You will learn more about this later in the course.

Return type – the data type of the value returned by the method or void if nothing is returned

Method name – name of the method

Parameters – comma separated input values passed to the method, should be () if no parameters are passed

Method body – the code that defines the functionality of the method

The datatype of the return value must match the return type mentioned in the method header. If the method does not return any value, **void** should be mentioned as the return type in the method header.

A method can accept data as arguments or parameters. The arguments passed while making the method call are known as **actual parameters** and the arguments present in the method header are known as **formal parameters**.

Whenever a value of a primitive data type is passed, the values are copied from the actual parameters to the formal parameters. This kind of parameter passing is known as **pass by value**. In pass by value, both the actual and formal parameters point to different memory locations and the values are copied in both the memory locations.

You can see that the values are changed only inside the method. This is because any changes made inside the method will be reflected only in the memory locations of the formal arguments and not in the memory locations of the actual arguments.

When an object is passed as a parameter, the formal and the actual parameters both refer to the same object and hence the same memory location. Therefore, the changes made inside the method to the formal parameters are reflected in the actual parameters also. This kind of parameter passing is known as **pass by reference**.

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

main outside the class

In real time development, we create classes and use it from another class. It is a better approach. Let's see a simple example, where we are having main() method in another class.

3 Ways to initialize object

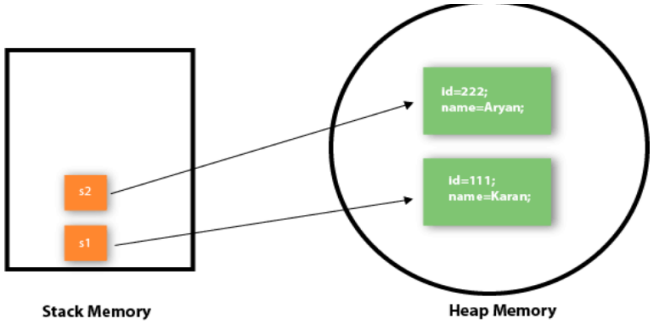
There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

Initializing an object means storing data into the object.

- We can also create multiple objects and store information in it through reference variable.

```
Student s1=new Student();
Student s2=new Student();
s1.insertRecord(111,"Karan");
s2.insertRecord(222,"Aryan");
```



As you can see in the above figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method
- By deserialization
- By factory method etc.

Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. If you have to use an object only once, an anonymous object is a good approach.

```
new Calculation();//anonymous object
```

Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

- Initialization of primitive variables:

1. `int a=10, b=20;`

- Initialization of reference variables:

1. `Rectangle r1=new Rectangle(), r2=new Rectangle();//creating two objects`

Constructors in Java

A **constructor** in Java is a special method that is used to initialize instance/class variables at the time of object creation.

Each time an object is created using the new() keyword, a constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating Java constructor

There are two rules defined for the 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.

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

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

- ✓ If you make any class constructor private, you cannot create the instance of that class from outside the class.

Types of Java constructors

There are two types of constructors in Java:

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

Java Default Constructor

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

The default constructor is used to initializes the default values to the class variables based on the data type.

You can also create **parameterless** constructor in a class. In this case, Java does not create a separate default constructor.

Java Parameterized Constructor

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

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

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.

They are differentiated by the compiler by the number of parameters in the list and their types.

Java Copy Constructor

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 static keyword

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

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

Java static variable

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

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- All instance data members will get memory each time when the object is created.
- The static variable gets memory only once in the class area at the time of class loading.
- Java static property is shared to all objects.
- It makes your program **memory efficient** (i.e., it saves memory).

Java static method

If you apply static keyword with any method, it is known as static method.

- *A static method belongs to the class rather than an instance of a class..*
- *A static method can be invoked without creating an instance of a class.*
- A static method can access static data member and can change the value of it.

Restrictions for the static method

There are two main restrictions for the static method. They are:

1. The static method cannot use/access non static data member or call non-static method directly.
2. this and super cannot be used in static context.

Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

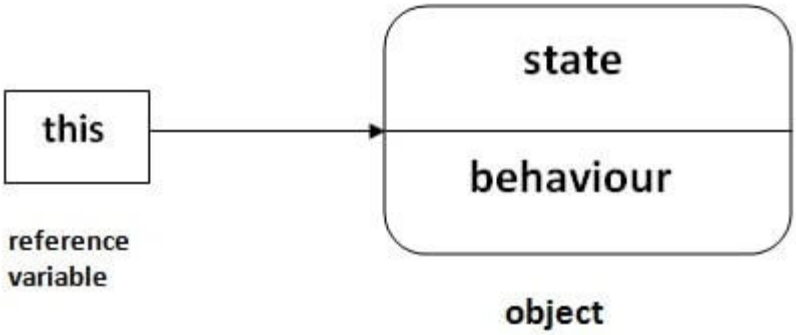
Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of class loading.
- If there are multiple static blocks, they will be executed in the order of their occurrence.

NOTE: - Static blocks and static methods cannot access non-static (instance) members directly since static methods do not belong to any object, so it is not possible to know which object's instance variables should be accessed.

this keyword in Java

In Java, this is a reference variable that refers to the current object.



Usage of Java this keyword

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 constructor (constructor chaining).
3. this can be used to invoke current class method (implicitly)
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.

> ➢ **It is better approach to use meaningful names for variables. So, we use same name for instance variables and parameters in real time, and always use this keyword.**

Real usage of this() constructor call

The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining.

> ***Rule: Call to this() must be the first statement in constructor.***

In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

Inheritance in Java

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

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

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

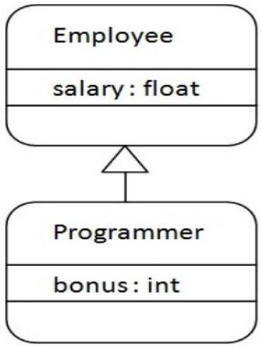
Super Class/Parent Class: Superclass is a class which is inherited by subclass. It is also called a base class or a parent class.

The syntax of Java Inheritance

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

✓ The meaning of "extends" is to increase the functionality.

Java Inheritance Example

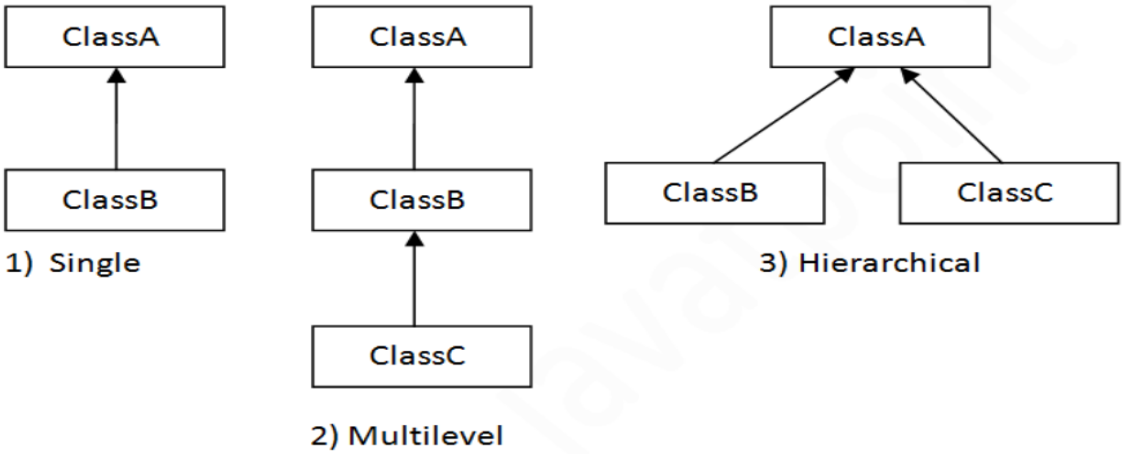


As displayed in the above figure, Programmer is the subclass and Employee is the superclass.

The relationship between the two classes is **Programmer IS-A Employee**.

Types of inheritance in java

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



- When a class inherits another class, it is known as a *single inheritance*
- When there is a chain of inheritance, it is known as *multilevel inheritance*.
- When two or more classes inherit a single class, it is known as *hierarchical inheritance.*
- When one class inherits multiple classes, it is known as multiple inheritance.

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

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class. Since, compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So, whether you have same method or different, there will be compile time error.

Aggregation in Java

- If a class has an entity reference, it is known as Aggregation.
- This kind of relationship exists between two classes when a reference variable of one class is a member variable in another class.
- Aggregation represents HAS-A relationship.

Why use Aggregation?

- For Code Reusability.

When use Aggregation?

- Code reuse is also best achieved by aggregation when there is no is-a relationship.
- Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Method Overloading in Java

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

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

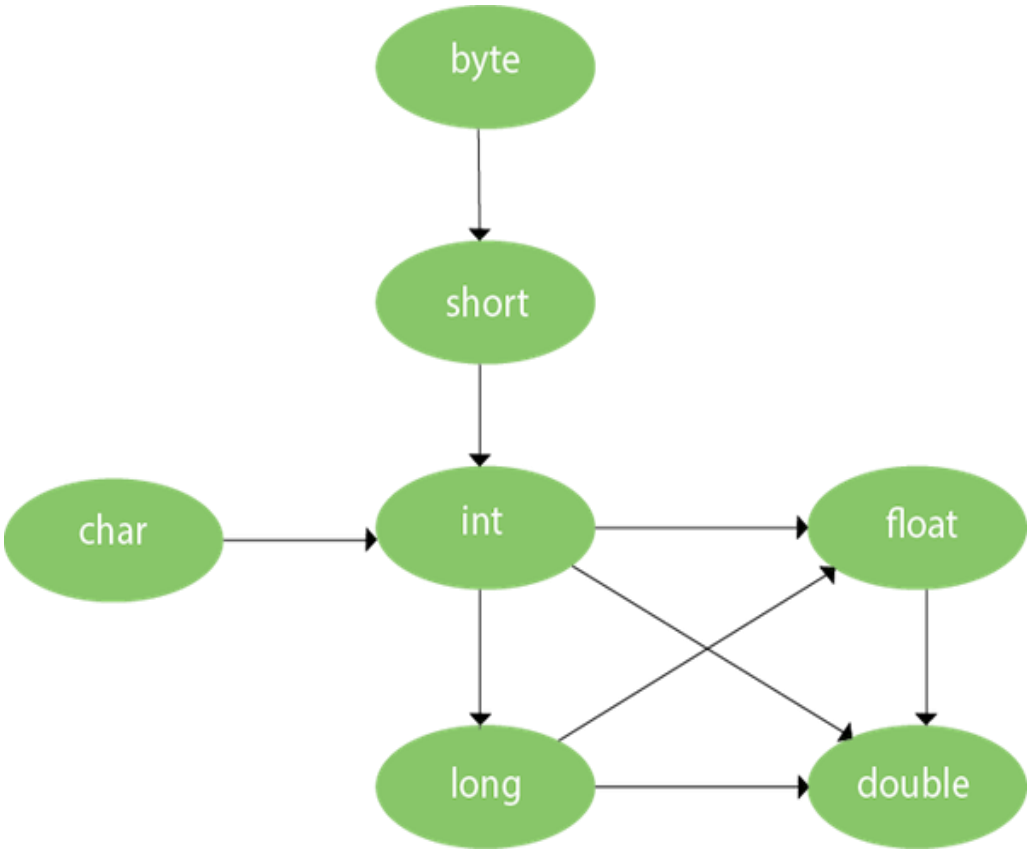
1. By changing number of parameters
2. By changing the data type of parameters

In Java, Method Overloading is not possible by changing the return type of the method only.

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

Method Overloading and Type Promotion



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on.

- One type is promoted to another implicitly if no matching datatype is found.
- If there are matching type arguments in the method, type promotion is not performed.
- If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

One type is not de-promoted implicitly for example double cannot be depromoted to any type implicitly.

Method Overriding in Java

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

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism.

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameters as in the parent class.
3. There must be an IS-A relationship (inheritance)

Java method overriding is mostly used in Runtime Polymorphism

Can we override static method?

No, a static method cannot be overridden.

Why can we not override static method?

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

Can we override java main method?

No, because the main is a static method.

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Covariant Return Type

The covariant return type specifies that the return type may vary in the same direction as the subclass.

Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type.

Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class instance(object).

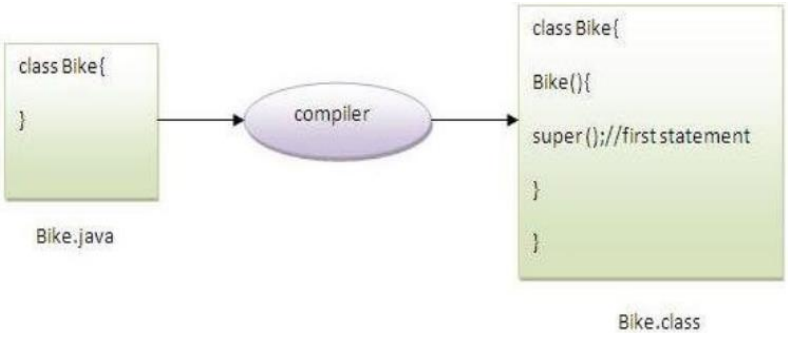
Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

Note: super() is added in each class constructor automatically by compiler if there is no super() or this().



As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

Instance initializer block

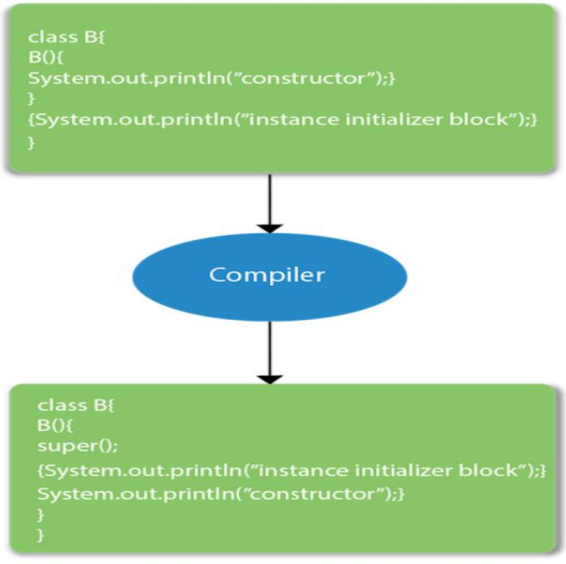
- **Instance Initializer block** is used to initialize the instance variable.
- It runs each time when object of the class is created.
- The initialization of the instance variable can be done directly but there can be extra operations performed while initializing the instance variable in the instance initializer block.
- The instance initializer block comes in the order in which they appear.

Why use instance initializer block?

Suppose we have to perform some operations while assigning value to instance variable.
e.g., a for loop to fill a complex array or error handling etc.

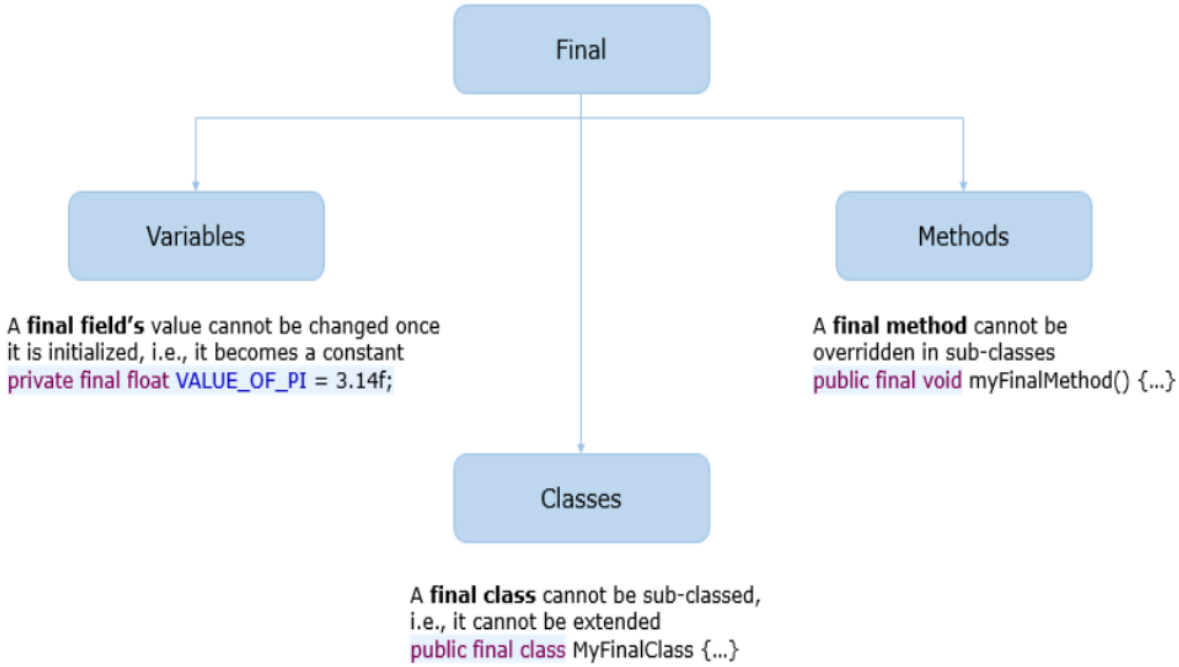
What is invoked first, instance initializer block or constructor?

It seems that instance initializer block is firstly invoked but NO. Instance intializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super(). So firstly, constructor is invoked.



Final Keyword In Java

The final keyword can be used with classes, variables and methods.



- We cannot change the value of final variable (It will be constant).

- A final variable that is not initialized at the time of declaration is known as blank final variable or uninitialized final variable. It can be initialized in the constructor only.
- A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized in the static block only.

Polymorphism in Java

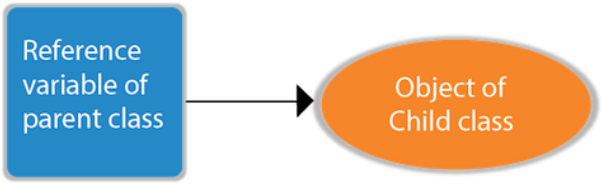
- If *one task is performed in different ways*, it is known as polymorphism.
- Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So, polymorphism means many forms.

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

Let's first understand the upcasting before Runtime Polymorphism.

Upcasting

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




```
class A{}
class B extends A{}

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

For upcasting, we can use the reference variable of class type or an interface type. For Example:

```
interface I{}
class A{}
class B extends A implements I{}
```

Here, the relationship of B class would be:

```
B IS-A A
B IS-A I
B IS-A Object
```

Since Object is the root class of all classes in Java, so we can write B IS-A Object.

Runtime Polymorphism in Java

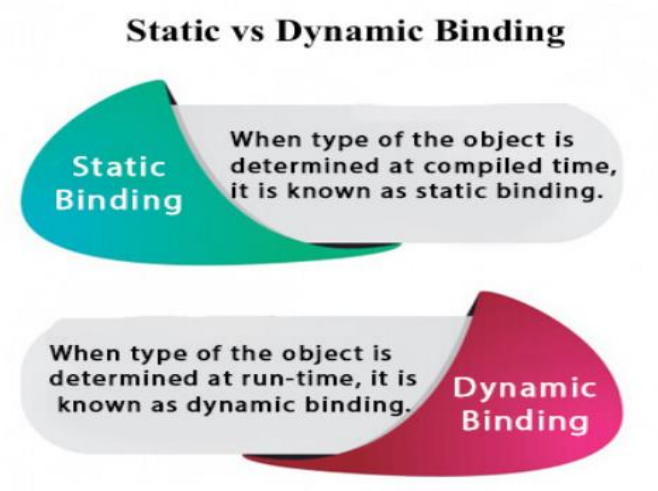
Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

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

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

There are two types of binding

1. Static Binding (also known as Early Binding).
2. Dynamic Binding (also known as Late Binding).



An object is an instance of particular java class, but it is also an instance of its superclass.

```
class Animal{}
class Dog extends Animal{
 public static void main(String args[]){
  Dog d1=new Dog();
 }
}
```

Here d1 is an instance of Dog class, but it is also an instance of Animal.

Abstraction in Java

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

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

A class which is declared as abstract is known as an **abstract class.**

Points to Remember

- If a class contains at least one abstract method, the class should be abstract.
- Abstract class cannot be instantiated.
- A class can be made abstract even without any abstract methods.
- It can have abstract and non-abstract methods (concrete methods).
- An abstract class can have a data member, abstract method, non-abstract, constructor, and static methods(main-method).
- It can have final methods which will force the subclass not to change the body of the method.
- Concrete (non-abstract) classes which extend an abstract class must implement all the abstract methods. Otherwise, they should be made abstract as well.

Example of abstract class; - abstract class A{}

Abstract Method in Java

A method which is declared as abstract and does not have any implementation is known as an abstract method.

Example of abstract method

```
abstract void printStatus();//abstract and   no method body ;method without body
```

Interface in Java

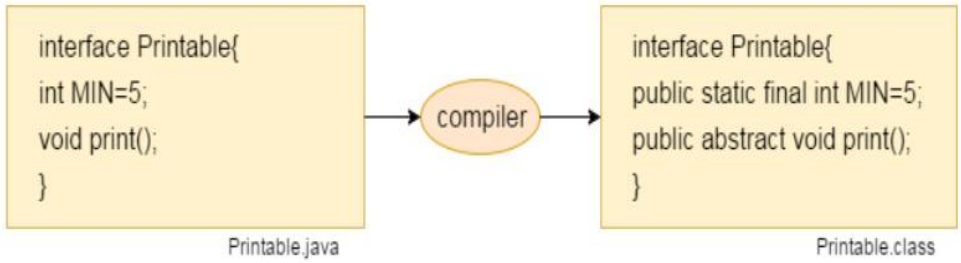
An **interface in Java** is a blueprint of a class.

- Java Interface also **represents the IS-A relationship**.
- Interface cannot be instantiated just like the abstract class.
- An interface is declared by using the interface keyword. It provides total abstraction.
- In an interface, all methods are implicitly public and abstract, and variables are implicitly public, static, and final.
- The class which implements the interface must implement all the abstract methods. Otherwise, it should be made abstract.
- **A class** can extend only one Java class and implement multiple Java interfaces.
- Since Java 8, we can have **default and static methods** in an interface.
- Since Java 9, we can have **private methods** in an interface.

Why use Java interface?

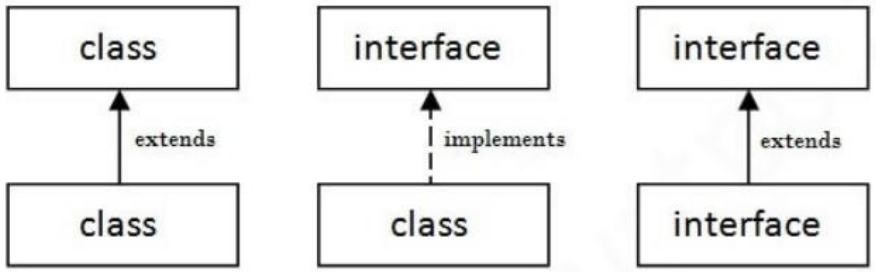
- It is used to achieve abstraction.
- It is used to multiple inheritance in Java.
- It is used to achieve loose coupling.

The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.



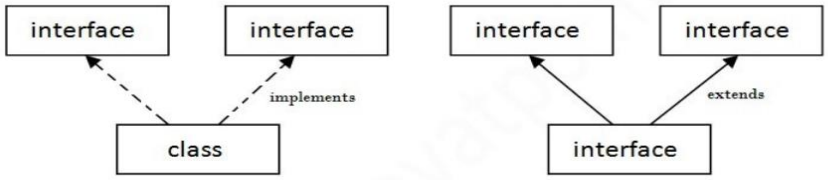
The relationship between classes and interfaces

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



Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

Java Package

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

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

There are two types of packages in java.

1. built-in package
2. user-defined package.

How to access package from another package?

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

- import package.*;

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

- import package.classname;

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

- fully qualified name.

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

  Note: If you import a package, subpackages will not be imported.

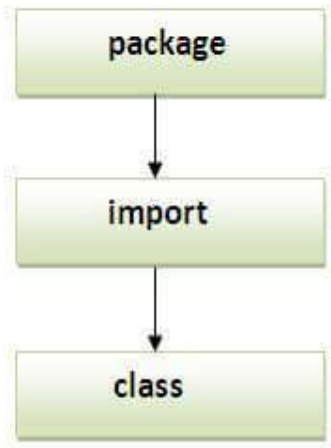
Advantage of Java Package

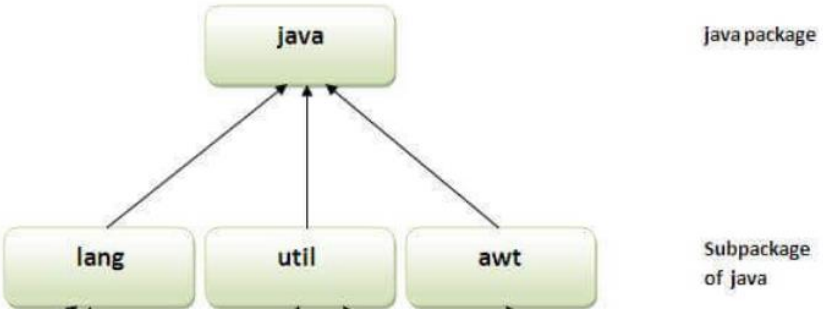
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

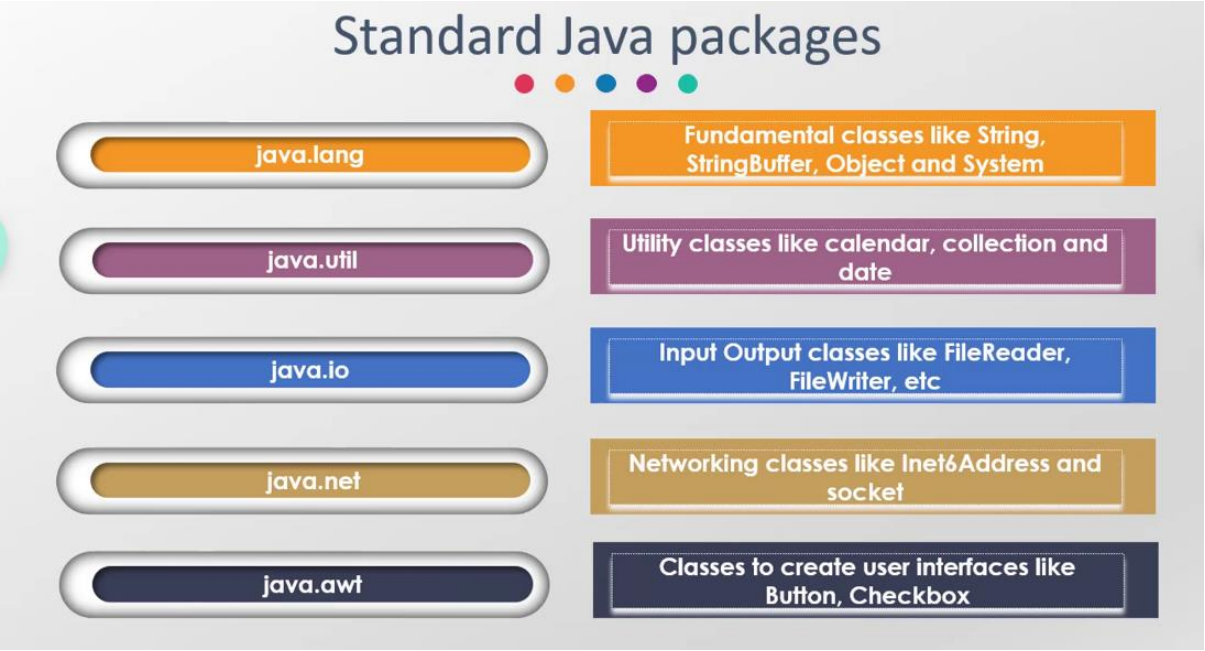
2) Java package provides access protection.

3) Java package removes naming collision.

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







Access Modifiers in Java

The access modifiers in Java specifies the accessibility of a field, method, constructor, or class.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside of the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside of the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside of the package through child class. If you do not make the child class, it cannot be accessed from outside of the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class.

Note: A class cannot be private or protected except nested class.

- ✓ If you make any class constructor private, you cannot create the instance of that class from outside the class.
- ✓ overridden method (i.e., declared in subclass) must not be more restrictive.

Encapsulation in Java

Encapsulation in Java is a *process of wrapping code and data together into a single unit*.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of a fully encapsulated class.

Advantage of Encapsulation in Java

By providing only a setter or getter method, you can make the class **read-only or write-only**.

It provides you the **control over the data**.

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

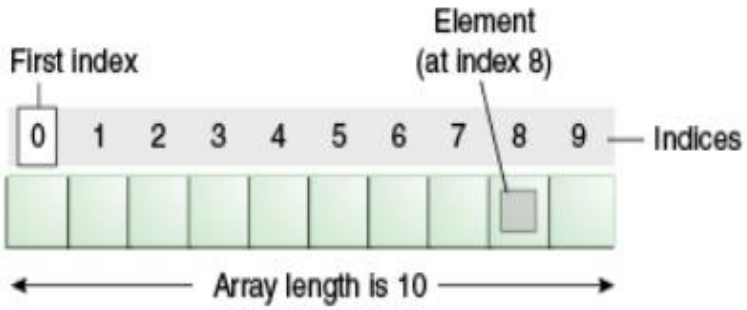
The encapsulated class is **easy to test**. So, it is better for unit testing.

The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

Java Array

An array is a collection of elements of same data type stored at contiguous memory locations.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



- **Size Limit:** We can store only the fixed number of elements in the array.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Syntax to Declare an Array in Java

1. dataType[] arr; (or) e.g.:- int[] value;
2. dataType []arr; (or) e.g.:- int []value;
3. dataType arr[]; e.g.:- int value[];

Instantiation

arr = new dataType[size];

value = new int[3];

DECLARATION AND INSTANTIATION

dataType[] arr= **new** datatype[size];

int[] value = new int[3];

DECLARATION AND INSTANTIATION INTIALIZATION

dataType[ ] arrayVarName = {elementsOfArraySeparatedByComma};

```
int[] value = {11, 13, 17, 19};  // declaration, instantiation and initialization
```

Initialization of array elements. Array indexing starts from 0.

arrayVarName[index] = element;

value[0] = 100;

value[1] = 200;

Reference type array.

Bank[] bank = {new Bank(), new Bank()};

Note: - The **length** attribute of an array is used to get its size

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

- Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method. E.g.: - *min*(**new int**[] {34,61,79,55})
- We can also return an array from the method in Java.

```
E.g:-  int[] value1;

       int []value2;

       int value3[];   //DECLARATION

       value3=new int[5];  //INSTANTIATION

       int[] value4=new int[5];  //DECLARATION AND INSTANTIATION

       value4[0]=10;

       value4[1]=34;   //Initialization of array elements

       value4[2]=67; //and so on

       value3=new int[]{10,76,98,76,42};    //to initialize already instantiated array

       int[] value5={10,76,98,76,42};  //// declaration, instantiation and initialization
```

Multidimensional Array in Java

Multi-dimensional array is an array of arrays, each element of the array holding the reference of other arrays.

Syntax to Declare Multidimensional Array in Java

1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];

Syntax for DECLARATION AND INSTANTIATION 2D array:
dataType[][] arrayRefVar = new dataType[rowsize][columnsize];

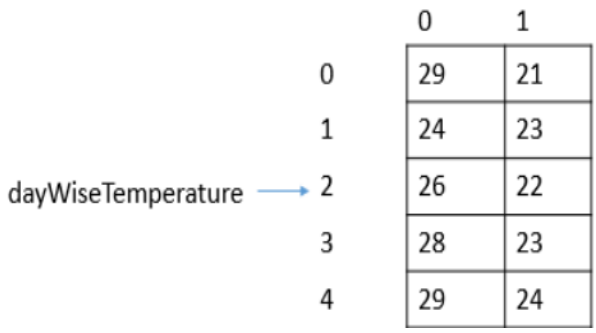
Note: The second dimension, i.e., the column size is optional.

//Another way of creating and initializing 2D array

int[][] dayWiseTemperature = new int[][] {{29,21},{24,23},{26,22},{28,23},{29,24}};

E.g.: -

```
int[][] a;

int[] []b;

int [][]c;

int []d[];

int e[][];   ////DECLARATION

e=new int[3][2];  //INSTANTIATION

int[][] f=new int[4][3];  //DECLARATION AND INSTANTIATION

int[][] g={{29,21},{24,23},{26,22}};  // declaration, instantiation and initialization

f=new int[][] {{29,21},{24,23},{26,22}};   ////to initialize already instantiated 2Darray

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

arr[0][0]=1;   arr[0][1]=2;

arr[0][2]=3;  arr[1][0]=4;  ////Initialization of array elements

arr[1][1]=5;     //and so on
```



Jagged Array in Java

- If you create odd number of columns in a 2D array, it is known as a jagged array.
- In other words, it is an array of arrays with different number of columns.

Wrapper classes in Java

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive.*

Use of Wrapper classes in Java

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc.

Let us see the different scenarios, where we need to use the wrapper classes.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

The eight classes of the *java.lang* package are known as wrapper classes in Java.

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing.

For example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Since Java 5, we don't need to use the valueOf() method of wrapper classes to convert the primitives into objects.

Unboxing

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing.

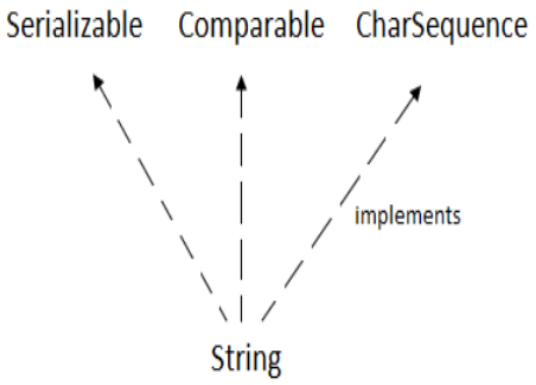
It is the reverse process of autoboxing.

Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the objects into primitives.

```
int a=10;

Integer b=Integer.valueOf(a);  //converting int into Integer explicitly

Integer c=a;                   //autoboxing, now compiler will write Integer.valueOf(a) internally


Integer j=675;

int k=j.intValue();  ///converting Integer to int explicitly

int i=j;       //unboxing, now compiler will write i.intValue() internally
```

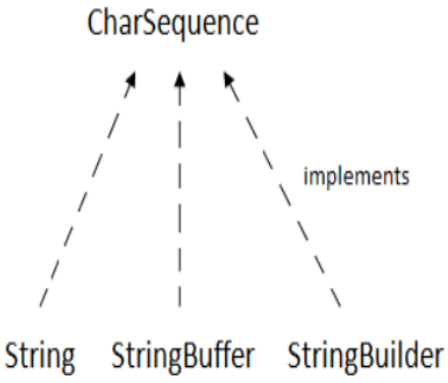
Java String

- ✓ String is a sequence of characters.
- ✓ But in Java, string is an object that represents a sequence of characters.
- ✓ An array of characters works same as Java string.
- ✓ The java.lang.String class is used to create a string object.
- ✓ **Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.
- ✓ The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in Java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

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

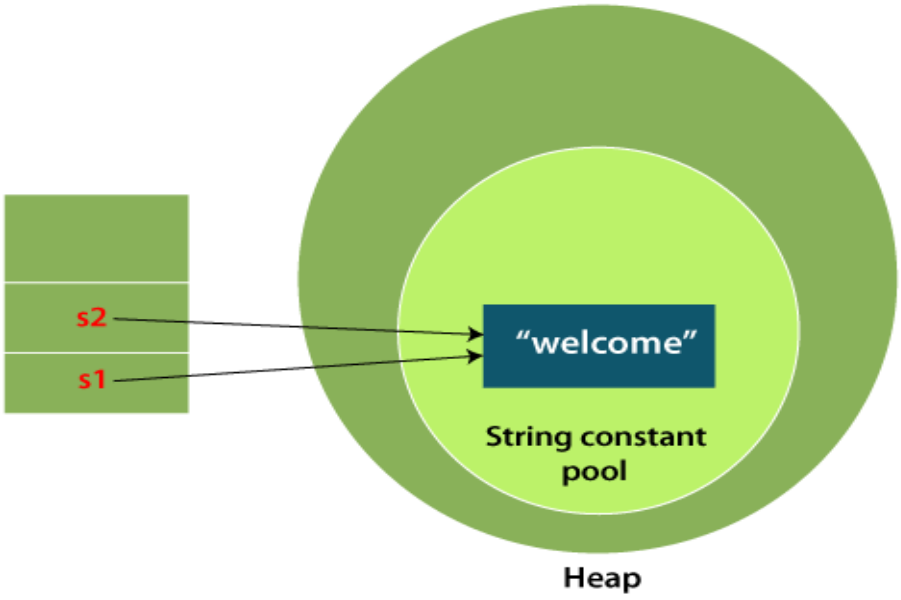
1. String s="welcome";

Every time you create a string literal, the JVM checks the String constant pool. (String constant pool in Java is a pool of Strings stored in Heap memory). If the string exists in the pool, then a reference of the existing literal is returned. If the string is not found, then a new instance is created and placed in the pool.

In the below diagram, since the value of s2 is also Welcome, new memory is not allocated for s2 and s2 points to the same memory location as s1 does.

In the below case, s1==s2 will be true.

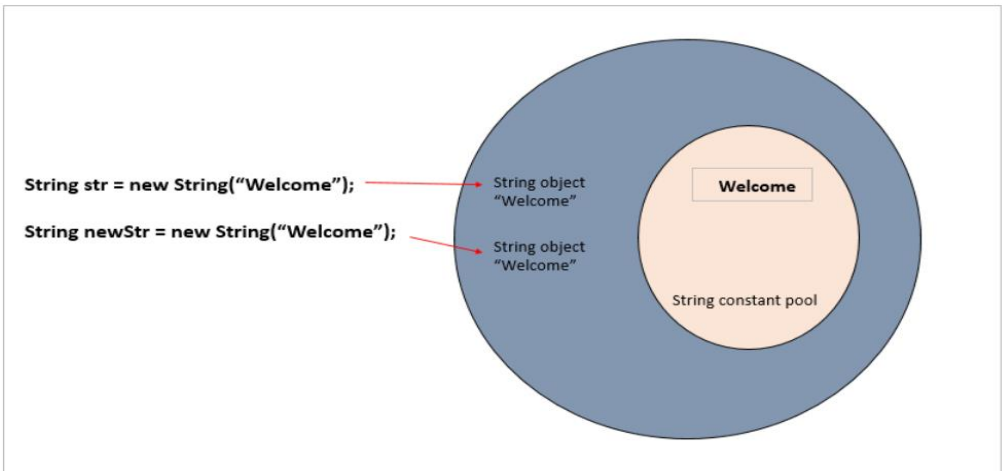
1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance



2) By new keyword

When you create a string using the new() keyword, JVM places the literal in the constant pool and also creates a new string object in heap memory. The reference variable refers to the object in the heap memory.

In the below case, str == newStr will be false.



Immutable String in Java

In Java, **String objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once String object is created its data can't be changed but a new String object is created.

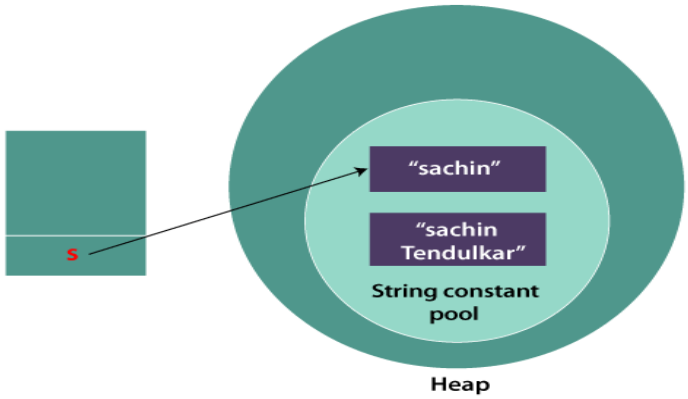
```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
   System.out.println(s);//will print Sachin because strings are immutable objects
 }
}
```

Test it Now

Output:

```
Sachin
```

Here Sachin is not changed but a new object is created with Sachin Tendulkar. That is why String is known as immutable.



As you can see in the above figure that two objects are created but ***s*** reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitly assign it to the reference variable, then it will refer to "Sachin Tendulkar" object.

```
class Testimmutablestring1{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s);
 }
}
```

Test it Now

Output:

```
Sachin Tendulkar
```

In such a case, s points to the "Sachin Tendulkar". Please notice that still Sachin object is not modified.

Java String compare

There are three ways to compare String in Java:

1. By Using equals() Method
2. By Using == Operator
3. By compareTo() Method

1) By Using equals() Method

The String class equals() method compares the original content of the string.

String class provides the following two methods:

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

2) By Using == operator

The == operator compares references not values

3) By Using compareTo() method

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

Suppose s1 and s2 are two String objects. If:

- **s1 == s2** : The method returns 0.
- **s1 > s2** : The method returns a positive value.
- **s1 < s2** : The method returns a negative value.

String Concatenation in Java

In Java, String concatenation forms a new String that is the combination of multiple strings. There are two ways to concatenate strings in Java:

1. By + (String concatenation) operator
2. By concat() method

Note: After a string literal, all the + will be treated as string concatenation operator.

Substring in Java

- ✓ A part of String is called **substring**. In other words, substring is a subset of another String.
- ✓ Java String class provides the built-in substring() method that extract a substring from the given string by using the index values passed as an argument.
- ✓ In case of substring() method startIndex is inclusive and endIndex is exclusive.

Note: Index starts from 0.

public String substring(int startIndex):

public String substring(int startIndex, int endIndex):

- **startIndex:** inclusive
- **endIndex:** exclusive

The String class provides many useful methods to perform various operations.

String Method	Description
int length()	returns number of characters in a string
String concat(String s)	concatenates or joins two strings and returns third string as the result
boolean equals(String s)	checks case sensitive equality of the string
boolean equalsIgnoreCase(String s)	checks case insensitive equality of the string
String toLowerCase()	returns a string that contains all the characters of the source string converted to lower case
String toUpperCase()	returns a string that contains all the characters of the source string converted to upper case
char charAt(int index)	returns a char value at the given index
String substring(int beginIndex, [endIndex])	returns substring from beginIndex to endIndex and if endIndex is not mentioned then returns substring from beginIndex to end of string
boolean contains(CharSequence s)	returns true if the character sequence is present in the string else returns false
String replace(char old, char new)	replaces all the occurrences of the specified character with the new character

What is a mutable String?

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

Java StringBuffer Class

Java StringBuffer class is used to create mutable String objects.

The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.

Important methods of StringBuffer class

Modifier and Type	Method
public synchronized StringBuffer	append(String s)
public synchronized StringBuffer	insert(int offset, String s)
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)
public synchronized StringBuffer	delete(int startIndex, int endIndex)
public synchronized StringBuffer	reverse()

Synchronized methods of String Buffer are append, insert, replace, reverse, delete(to memorize "airrd").

Java StringBuilder Class

Java StringBuilder class is used to create mutable (modifiable) String. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Important methods of StringBuilder class

Method
public StringBuilder append(String s)
public StringBuilder insert(int offset, String s)
public StringBuilder replace(int startIndex, int endIndex, String str)
public StringBuilder delete(int startIndex, int endIndex)
public StringBuilder reverse()

Difference between StringBuffer and StringBuilder

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable. There are many differences between StringBuffer and StringBuilder.

No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. I means two threads can call the methods of StringBuilde simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

How to create Immutable class?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable.

We can also create immutable class by creating final class that have final data members.

Java toString() Method:-

- toString() method returns the String representation of the object.

If you print any object, Java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output.

Exception:-

- In Java, exception is an event that disrupts the normal flow of the program.
- Simply, an object which is thrown at runtime.

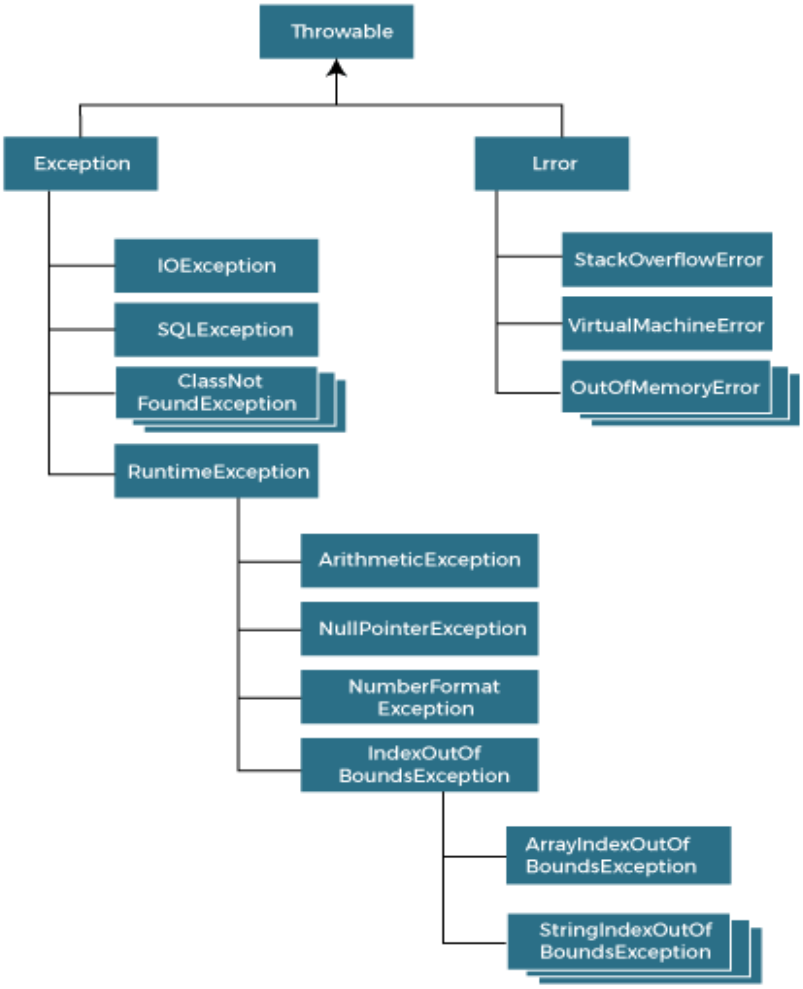
Exception Handling in Java

- ✓ **Exception Handling in Java** is a mechanism to handle runtime exceptions so that the normal flow of the program can be maintained.

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

Java Exception Hierarchy: -

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error.



There are two different types of exceptions in Java:

- Checked Exception
- Unchecked Exception

Checked Exception

All exceptions other than runtime exceptions are known as Checked Exceptions as the compiler checks them during compilation. Checked exceptions are checked at compile-time.

E.g. - SQLException, IOException, etc. are Checked Exceptions

Unchecked Exception

Unchecked exceptions are runtime exceptions. These exceptions are not checked at compile-time.

E.g. - ArithmeticException, NullPointerException, IndexOutOfBoundsException, etc. are Unchecked Exceptions

Note: - All the exceptions inherit the Exception class, which is a child of the Throwable class.

In Java, **java.lang.Exception** class is the superclass of all exception objects.

An exception occurs!

The output is the exception's **stack trace**. It tells the type of exception, message, method call stack, and exception's location. This helps to **debug** the code.

- The try block contains the code which may throw some exception.
- The try block is followed by single or multiple catch blocks or a finally block.
- A catch block is an exception handler that can handle the exception specified as its argument.

Best practice: In a catch block, specific exceptions are preferred rather than general exceptions.

- If an exception arises from the try block, the matching catch block gets executed. In the try block, the code after the line which caused the exception will not be executed.
- If a catch block handles the exception, then the execution continues from the code placed after the try-catch block.
- All catch blocks must be ordered from most specific to most general.
- A catch block that can handle objects of **Exception** class can catch all the exceptions. This block should always be the last catch block in the catch sequence.
- If an exception occurs in a method, there are two ways to handle it:

Handle the exception there itself or **allow it to propagate to be handled somewhere else.**

- If no exception is thrown from the try block, their subsequent catch blocks are ignored.
- If a similar exception handling logic is required for multiple exceptions, a **multi-catch** block can be used.

Java Nested try block.

In Java, using a try block inside another try block is permitted. It is called as nested try block.

When any try block does not have a catch block for a particular exception, then the catch block of the outer (parent) try block are checked for that exception, and if it matches, the catch block of outer try block is executed.

Java finally block

Java finally block is a block used to execute important code such as closing the connection, etc.

Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

The finally block follows the try-catch block.

> ***Note: If you don't handle the exception, before terminating the program, JVM executes finally block (if any).***

> ***Rule: For each try block there can be zero or more catch blocks, but only one finally block.***

> ***Note: The finally block will not be executed if the program exits***

Java throw keyword.

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

- We can throw either checked or unchecked exceptions in Java by throw keyword.
- It is mainly used to throw a custom exception.

By extending the Exception class, we can define a user defined exception class.

```
class BeginnerException extends Exception{

public BeginnerException(String s) {super(s);}

}
```

Java throws keyword.

The **Java throws keyword** is used to declare an exception.

Exception Handling in java is mainly used to handle the checked exceptions.

- If there occurs any unchecked exception such as NullPointerException, it is programmers' fault that he is not checking the code before it being used.
- Exception and its subclasses are Checked Exceptions.

Which exception should be declared?

Ans: Checked exception only, because:

- **unchecked exception:** under our control so we can correct our code.
- **error:** beyond our control.

Exception	Description
ClassNotFoundException	No definition for the class is found
IOException	I/O operation is interrupted or failed
NoSuchMethodException	Thrown when the method is not found
InterruptedException	If a thread is sleeping, waiting or otherwise occupied and it is interrupted then this Exception is thrown.
IllegalAccessException	If the method doesn't have access to the zero-argument constructor of class then this Exception is thrown

Rule: If we are calling a method that declares an exception, we must either caught or declare the exception.

The main() method will be aware and be prepared to handle the exception.

There are two cases:

1. **Case 1:** We have caught the exception i.e. we have handled the exception using try/catch block.
2. **Case 2:** We have declared the exception i.e. specified throws keyword with the method.

- In case we declare the exception, if exception does not occur, the code will be executed fine.
- In case we declare the exception and the exception occurs, it will be thrown at runtime because **throws** does not handle the exception.

- **If the superclass method does not declare an exception**
    1. If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
- **If the superclass method declares an exception**
    1. If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

Points to remember.

- An Exception is an unwanted event that interrupts the normal flow of the program.
- In Java, exception is basically of two types – Checked and Unchecked Exception.
- Exception can be handled with try, catch and finally block.
- A single try block can have any number of catch blocks.
- A single try block must be followed by at least one catch block or finally block.
- A generic catch block can handle all the exceptions.
- If no exception occurs in try block, then the catch blocks are completely ignored.
- Generic catch block must be the last catch block of try-catch.
- Finally block is optional.
- A finally block gets executed irrespective of whether an exception occurs or not.
- A finally block must be associated with a try block, you cannot use finally without a try block.
- An exception in the finally block behaves exactly like any other exception.
- Exception can be raised explicitly also by using throw keyword.
- If you are propagating the exception to the method call using throw, then method must be declared with throws keyword.
- Programmer can define their own exception, i.e., create User-defined exception.
- User-defined exception must extend Exception class.

Unary operators act upon only one operand and perform operations such as increment, decrement, negating an expression or inverting a boolean value.

Operator	Name	Description
++	post increment	increments the value after use
	pre increment	increments the value before use
--	post decrement	decrements the value after use
	pre decrement	decrements the value before use
~	bitwise complement	flips bits of the value
!	logical negation	Inverts the value of a boolean

Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication and division.

Operator	Description
+	additive operator (also used for String concatenation)
-	subtractive operator
*	multiplication operator
/	division operator
%	modulus operator

Relational operators are used to compare two values. The result of all the relational operations is either true or false.

Operator	Description
==	equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
!=	not equal to

\

Logical operators are used to combine two or more relational expressions or to negate the result of a relational expression.

Operator	Name	Description
&&	AND	result will be true only if both the expressions are true
||	OR	result will be true if any one of the expressions is true
!	NOT	result will be false if the expression is true and vice versa

Assume A and B to be two relational expressions. The below tables show the result for various logical operators based on the value of expressions, A and B.

A	B	A&&B	A||B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

A	!A
true	false
false	true

Ternary operator is used as a single line replacement for if-then-else statements and acts upon three operands.

Syntax:

<condition> ? <value if condition is true> : < value if condition is false>

Assignment operator is used to assign the value on the right hand side to the variable on the left hand side of the operator.

Some of the assignment operators are given below:

Operator	Description
=	assigns the value on the right to the variable on the left
+=	adds the current value of the variable on left to the value on the right and then assigns the result to the variable on the left
-=	subtracts the value of the variable on the right from the current value of the variable on left and then assigns the result to the variable on the left
*=	multiplies the current value of the variable on left to the value on the right and then assigns the result to the variable on the left
/=	divides the current value of the variable on left by the value on the right and then assigns the result to the variable on the left

BITWISE OPERATOR: -

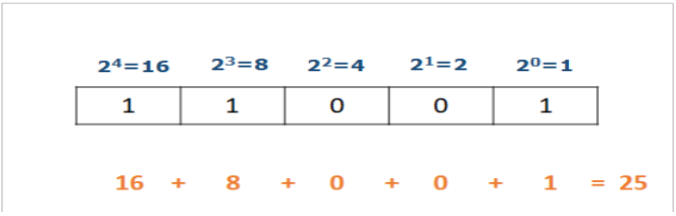
25 decimal = 11001 binary

You will now see how to convert the binary number back to decimal number.

The decimal number is equal to the sum of binary digits (d_n) times their power of 2 (2^n).

Lets take the example of 11001.

$11001 = 1*2^4+1*2^3+0*2^2+0*2^1+1*2^0=16+8+0+0+1=25$



- **Bitwise OR(|)**

  It returns bit by bit OR of the input values. If either of the bits is 1, then it gives 1, else it gives 0.

E.g. - The output of 10 | 5 is 15.

- **Bitwise AND(&)**

It returns bit by bit AND of the input values. If both the bits are 1, then it gives 1, else it gives 0.

E.g. - The output of 10 & 5 is 0.

```
System.out.println(10<<2);//10*2^2=10*4=40

System.out.println(10<<3);//10*2^3=10*8=80

System.out.println(20<<2);//20*2^2=20*4=80

System.out.println(15<<4);//15*2^4=15*16=240

System.out.println(10>>2);//10/2^2=10/4=2

System.out.println(20>>2);//20/2^2=20/4=5

System.out.println(20>>3);//20/2^3=20/8=2
```

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.The bitwise & operator always checks both conditions whether first condition is true or false.

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.The bitwise | operator always checks both conditions whether first condition is true or false.

Have a look at the evaluation of below expression based on precedence and associativity of operators:

```
5 + 4 * 9 % (3 + 1) / 6 - 1
  5 + 4 * 9 % 4 / 6 - 1
    5 + 36 % 4 / 6 - 1
      5 + 0 / 6 - 1
        5 + 0 - 1
          5 - 1
            4
```

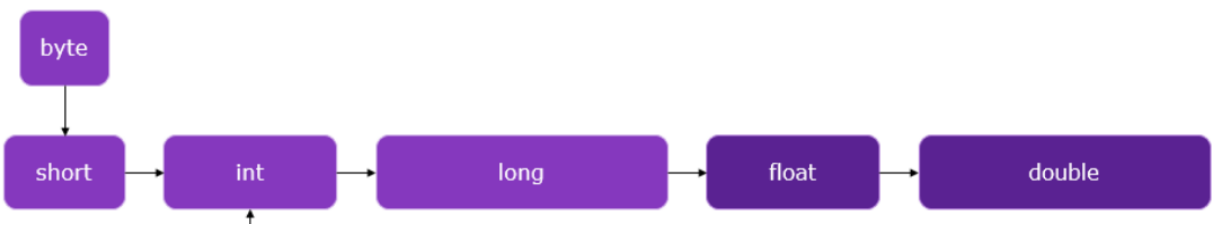
Operator	Symbols	Associativity
Post-unary operators	expression ++, expression --	Left to Right
Pre-unary operators	++ expression, -- expression	Right to Left
Other unary operators	-, !, ~, +, (type)	Right to Left
Multiplication / Division / Modules	*, /, %	Left to Right
Addition / Subtraction	+, -	Left to Right
Shift operators	<<, >>, >>>	Left to Right
Relational operators	<, >, <=, >=, instanceof	Left to Right
Equal to / Not equal to	==, !=	Left to Right
Bitwise operators	&, ^, |	Left to Right
Short - circuit logical operators	&&, ||	Left to Right
Ternary operators	boolean expression? expression1 : expression2	Right to Left
Assignment operators	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	Right to Left

Type Conversion: -

This conversion is of two types:

- Implicit
- Explicit

Implicit Type Conversion is also known as **Widening** conversion.



Explicit Conversion is used when you want to assign a value of larger range data type to a smaller range data type. This conversion is not done by the compiler implicitly as there can be loss of data in some cases. Hence, programmer has to be cautious about such conversions. This is also known as **Narrowing** conversion.

E.g.:

```
double totalPrice = 200;

int newPrice = (int)totalPrice;
```

Java Control Statements

Java provides three types of control flow statements.

1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

1) If Statement:

Java, there are four types of if-statements given below.

1. Simple if statement
2. if-else statement
3. if-else-if ladder
4. Nested if-statement

1) Simple if statement:

It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

```
if(condition) {
statement 1; //executes when condition is true
}
```

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

4. Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
if(condition 1) {
  statement 1; //executes when condition 1 is true
```

```
    if(condition 2) {
   statement 2; //executes when condition 2 is true
                        }
  else{
  statement 2; //executes when condition 2 is false
      }
  }
```

Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.

```
switch (expression or variable) {

  case value1: <statements>;

            break;

  case value2: <statements>;

            break;

  default: <statements>;

}
```

- During execution, the result of expression or variable written in the switch statement is compared with the constant values of cases one by one. When a match is found, the set of statements present in that case are executed until a **break** statement is encountered or till the end of switch block, whichever occurs first.
- In the absence of break statement, the flow of control falls through subsequent cases and executes the statements of all those cases until it reaches a break statement or end of switch block.
- The switch block can have a special case called **default**. The default case is executed when none of the cases match with the value of expression/variable. default is optional. If none of the cases match and if there is no default statement, the control comes out of switch block without executing any case.
- The case variables can be int, short, byte, char, or enumeration. but float or double are not supported.
- String type is also supported since version 7 of Java.

- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int,char, long (with its Wrapper type), enums and string*.

Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. while loop
2. do-while loop
3. for loop

Java While Loop

The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.

The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while loop.

Syntax:

```
while (condition){
//code to be executed
Increment / decrement statement
}
```

If you pass **true** in the while loop, it will be infinitive while loop.

Java do-while Loop

Java do-while loop is called an **exit control loop**. Therefore, unlike while loop the do-while checks the condition at the end of loop body. The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do{
//code to be executed / loop body
//update statement
}while (condition);
```

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

JAVA FOR LOOP

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is **fixed**, it is recommended to use for loop.

There are three types of for loops in Java.

- Simple for Loop
- For-each or Enhanced for Loop
- Labeled for Loop

Java Simple for Loop

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value.

Syntax:

```
for(initialization; condition; increment/decrement){
//statement or code to be executed
}
```

Java Nested for Loop

If we have a for loop inside another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Java for-each Loop

The for-each loop is used to traverse array or collection in Java.

It works on the basis of elements and not the index. It returns element one by one in the defined variable.

Syntax:

```
for(data_type variable : array_name){
//code to be executed
}
```

Java Labeled For Loop

We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful while using the nested for loop as we can break/continue specific for loop.

Syntax:

```
<Labelname>:
for(initialization; condition; increment/decrement){
//code to be executed
}
```

The break and continue keywords breaks or continues the innermost for loop respectively.

Java Break Statement

break statement is used to terminate a loop.

After terminating the loop, the next statement following the loop gets executed. In case of break statement written in nested loops, the inner most loop gets terminated and the flow of control continues with the statements of outer loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Java Continue Statement

continue statement is used to skip the current iteration of a loop and continue with the next iteration. In case of while and do-while loops, continue statement skips the remaining code of the loop and passes the control to check the loop condition. Whereas in case of for loop, the control goes to the increment section and then the condition is checked.

break	continue
terminates a loop or a switch statement	skips the remaining statements of the loop for the current iteration
can be used with loops and switch	can be used only within loops