Java is a case sensitive language, we have to follow the naming conventions

Naming conventions in Java

1) Pascal case Convention
2) Lower case convention
3) Upper case convention
4) Camel case convention

1) Pascal case Convention:

- The first letter of the identifier should start with Upper case letter.
- If the Identifier is the combination of multiple words then every word first letter should be capital.

Applicable for:

- class names, interface names, enumeration names, exception names.

Examples:

Classes: String, Scanner, Applet, BufferReader, InputStreamReader, etc.

Interfaces: Runnable, ActionListener, MouseListener, MouseMotionListener, etc.

Enumerations: Thread.State, ClientInfoStatus, etc.

Exceptions: Exception, ArithmeticException, ArrayIndexOutOfBoundsException, etc.

2) Lower case convention:

All the letters in the identifier should be small.

Applicable for:

packages in Java, keywords

Examples:

Some packages: java.lang.*, java.io.*, java.math.*, java.sql.*, java.util.*, javax.servlet.*, javax.httpservlet.*, etc.

keywords: int ,char,try, finally, etc.

3) Upper case convention:

All the letters in the identifier should be capital.

Applicable for:

constants in Java (optional)

Examples:

MIN_PRIORITY, MAX_PRIORITY, NORM_PRIORITY, PI, CASE_INSENSITIVE_ORDER, etc.

4) Camel case convention:

- If the Identifier is single word then all the letters in that word should be small.
- If the Identifier is the combination of multiple words then all the letters in the first word should be small after that every word first letter should be capital.

Applicable for:

All the elements in the Java apart from the Pascal case convention, lowercase convention and uppercase convention.

Examples:(Method names, variable names, array names, object reference names, etc.)

Method names: charAt(), toLower(), toUpper(), main(), nextInt(), next(), getConnection(), etc.

Variable names: your wish….

Array names: your wish….

Object reference names: your wish….

Java Class Members

1) Variables
2) Methods
3) Objects of the class
4) Static block
5) Constructors
6) Initializer block
7) Exception blocks
8) Inner classes

1. Variables:

I) class level / Global variables

II) Local variables / arguments / perameters

I) class level variables:

- The variables which we are declaring in the class level
- The Scope of these variables are available in the entire class, we can assess them at anywhere in the class (but based up the nature of the variable).

Nature of the class level variable:

i) Instance / object nature / non-static

ii) static nature

iii) final static nature (constant variables)

i) Instance variable nature:

1) It is object specific, you can access it directly in the instance / object context but not in the static context.
2) If you want to access it in the static context, you have to create the object of the class and access it with the help of the object of the class.
3) If you want to update the value of the instance variable in one object then it will not reflect in the other object (Because it is object specific).
4) If you are not assigning any value to the instance variable at the time of declaration, it will take the default value of the data type associated with it.
5) If there is a naming ambiguity in between local variables and instance variables, to call the instance variables use "this" keyword in the object context.
6) this and super keywords are restricted in static context (static blocks, static methods).

ii) Static nature:

1) It is class specific, you can directly access it at anywhere in the class (i.e. both in the static context as well as instance context) with the help of

- member_name (or)
- class_name.member_name (or)
- object_reference.member_name.

   But it is always recommend to use class_name.member_name to overcome naming ambiguity between local and class level variables.

2) It is sharable among the all the objects in the class, if you want to update the value of the static variable in one object then it will reflect in to all other objects in that class (Because it is class specific).
3) If you are not assigning any value to the static variable at the time of declaration, it will take the default value of the data type associated with it.
4) this and super keywords are restricted in static context (static blocks, static methods).

iii) final static nature (constants):

1) There is no keyword to define constants in java, we have to use the combination of final, static keywords to define constants in Java.
2) Constants are nothing but the fixed values, at the time of declaration you have to assign the value to the constant. Once if you assign some value the constant variable, you cannot modifiy that value, it will be fixed.

II) Local Variables:

1) Which are specific to particular block.

2) The scope of these variables are within that block only

3) If you are not assigning any value to this variable at the time of declaration and you what to use it then it will show an error message - variable might not have been initialized

4) If you what to use the local variable, you have to assign some value to it.

5) If is there any ambiguity between local variables and class level variables, please do the following

- If the ambiguity occurs in between local and static variables please use class_name.member_name to access static varible and use member_name for local variable.

- If the naming ambiguity occurs in between local and instance variables please use this.member_name to access instance members in instance context and object_ref.member_name in static context, use member_name for local variable.

2. Methods

I) Static methods

II) Instance methods

I) Static methods:

1) You can directly access this method, you need not to create object of the class to access it.

2) You can use

i) method_name (or)

ii) class_name.method_name() (or)

iii) object_reference.method_name() to access these methods.

But the recommended way is to access class_name.method_name();

Restrictions for the static method:

1. The static method cannot use non static data member or call non-static method directly.

(You have to create the object of the class to access non-static context in static context).

2. this and super cannot be used in static context.

II) Instance methods:

1) You need to create the object of the class in order to invoke the instance methods

2) If you want to invoke one instance method from another instance method, we need not require object, directly we can invoke another instance method.

Note: Better to use this keyword to invoke one instance method from another instance method (To overcome the ambiguity between super class instance methods and sub class instance methods).

3. static block:

Syntax:

---

static

{

	//content

}

Importance of static block:

1) It is used to initialize the static data member.

2) It is executed before the main method at the time of class loading.

Memory storage in Java:

Before constructors concept we have to know about the memory storage information about all the members in Java:

Stack and heap are the memories allocated by the OS to the JVM that runs in the system.

- Stack is a memory place where the methods calls, intermediate results and the local variables are stored.
- Variable references either primitive or object references are also stored in the stack.
- Heap is a memory place where the objects and its instance variables are stored.

Summary:

- Class objects, including method code and static fields: heap.
- Objects, including instance fields: heap.
- Local variables and calls to methods: stack.
- Variable references either primitive or object references are also stored in the stack.
- all satic variables, methods, instance variables, instance methods, objects of the class - heap
- Object or primitive data type references, methods calls, intermediate results from the method calls, local variables - static memory.

5. Constructors:

A constructor is a block of codes similar to the method.

Syntax of the constructor:

---

access_modifier class_name()

{

}

- 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 (i.e. initializing instance variables).
- Every time an object is created using the new() keyword, at least one constructor is called.
- If there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.
- With the help of the constructors we can initialize the instance variables of the class.

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

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.

Syntax:

---

access_modifier constructor_name()

{

}

Test()

{

}

Types of Java constructors:

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)

2. Parameterized constructor

Important Note: In each and every constructor the first statement by default is super(), which is used to invoke the default constructor of the super class.

6. Initializer blocks in Java:

{

}

- Initializer block contains the code that is always executed whenever an instance is created.
- It is used to declare/initialize the common part of various constructors of a class.
- The order of initialization constructors and initializer block doesn't matter, initializer block is always executed before constructor.
- It seems that instance initializer block is firstly invoked when compare to the constructor but NO.
- Instance initializer 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().

Rules for instance initializer block:

- The instance initializer block is created when instance of the class is created.
- The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
- The instance initializer block comes in the order in which they appear.

Packages in Java:

java package is a group of similar types of classes, interfaces, Exceptions, Enumerations, Error, Annotations, and sub packages.

API (Application Programming Interface) 1.6, 1.7, 1.8, 1.9 , 10,11, 12, 14, 15

Packages are used for:

- Preventing naming conflicts.

Example:

- lbrce.it.Employee
- lbrce.cse.Employee
- lbrce.eee.Employee

- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier.
- Providing controlled access.
- Packages can be considered as data encapsulation (or data hiding).

All we need to do is put related classes into packages. After that, we can simply write an import class from existing packages and use it in our program. A package is a container of a group of related classes where some of the classes are accessible are exposed and others are kept for internal purpose.

We can reuse existing classes from the packages as many time as we need it in our program.

Package categories in Java:

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

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

Syntax to define the package:

How to access package from another package?

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

1) import package.*;

2) import package.classname;

3) fully qualified name.

import java.util.*/ import java.util.Scanner;

java.util.Scanner sc= new java.util.Scanner(System.in);

Java Static Import:

The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Advantage of static import:

Less coding is required if you have access any static member of a class oftenly.

Disadvantage of static import:

If you overuse the static import feature, it makes the program unreadable and unmaintainable.

What is the difference between import and static import?

The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

Access Modifiers in Java

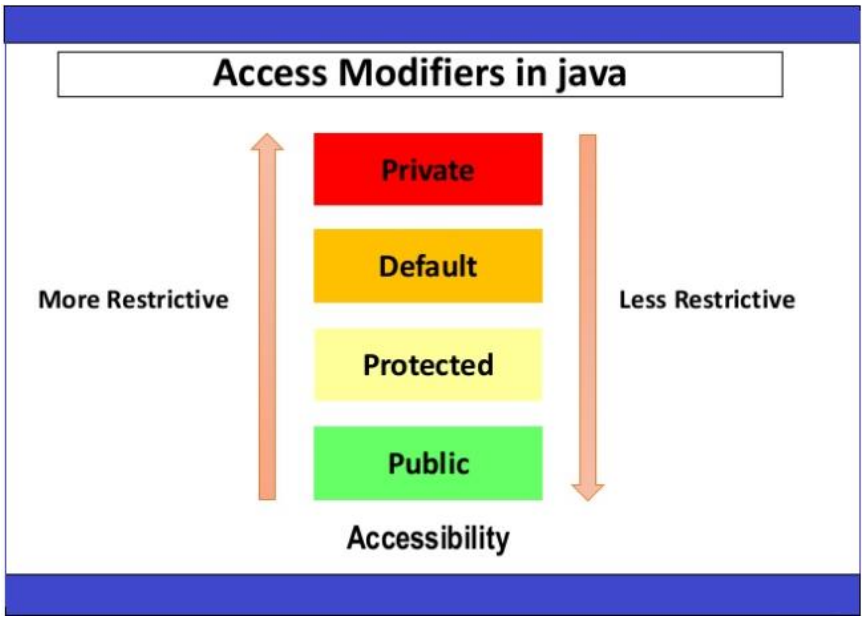
There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

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 the class.
2) **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside 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 the package through child class. If you do not make the child class, it cannot be accessed from outside 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.

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes



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

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc.

Protected Access Modifier:

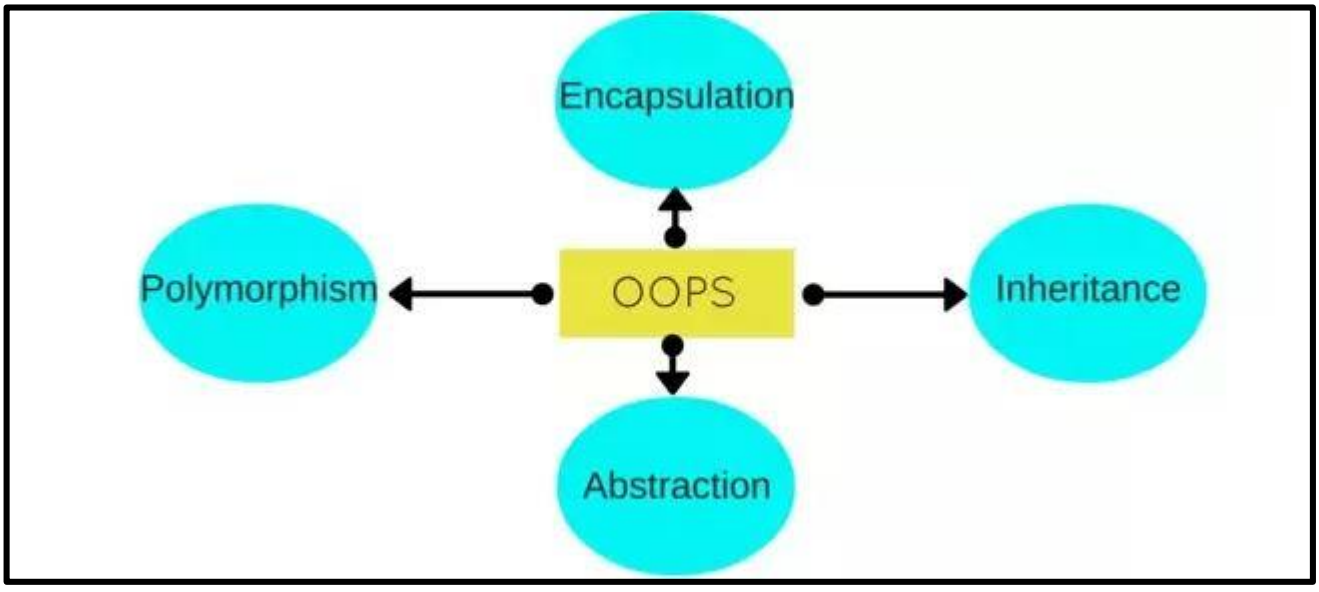
The **protected access modifier** is accessible within package and outside the package but through inheritance only.

Important Note: Protected member of the super class will be accessed in the sub class with the help of sub class object only (If those two classes are available in different packages), it will not be accessed by the super class object.

Protected member of the super class will be accessed by the sub class object as well as super class object in sub class (If those two classes are available in same package).

Note1: Protected member accessibility scope is as like as default member in the same package, you can access it with or without inheritance from one class to another class in the same package.

Note2: Protected member accessibility scope will become private accessibility with in the sub class of another package.



Encapsulation in Java

Encapsulation simply means binding object state (fields) and behaviour (methods) together. If you are creating class, you are doing encapsulation.

What is encapsulation?

The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class.

However if we setup public getter and setter methods to update (for example void setSSN (int ssn)) and read (for example int getSSN()) the private data fields then the outside class can access those private data fields via public methods.

Encapsulation is essential in Java because:

- It controls the way of data accessibility
- Modifies the code based on the requisites
- Helps us to achieve a loose couple
- Achieves simplicity of our application
- It also allows you to change the part of the code without disrupting any other functions or code present in the program

Benefits of Encapsulation

Data Hiding: Here, a user will have no idea about the inner implementation of the class. Even user will not be aware of how the class is storing values in the variables.

He/she will only be aware that we are passing the values to a setter method and variables are getting initialized with that value.

Increased Flexibility: Here, we can make the variables of the class as read-only or write-only depending on our requirement.

In case you wish to make the variables as read-only then we have to omit the setter methods like setName(), setAge() etc. or if we wish to make the variables as write-only then we have to omit the get methods like getName(), getAge() etc. from the above program.

Reusability: It also improves the re-usability and easy to change with new requirements.

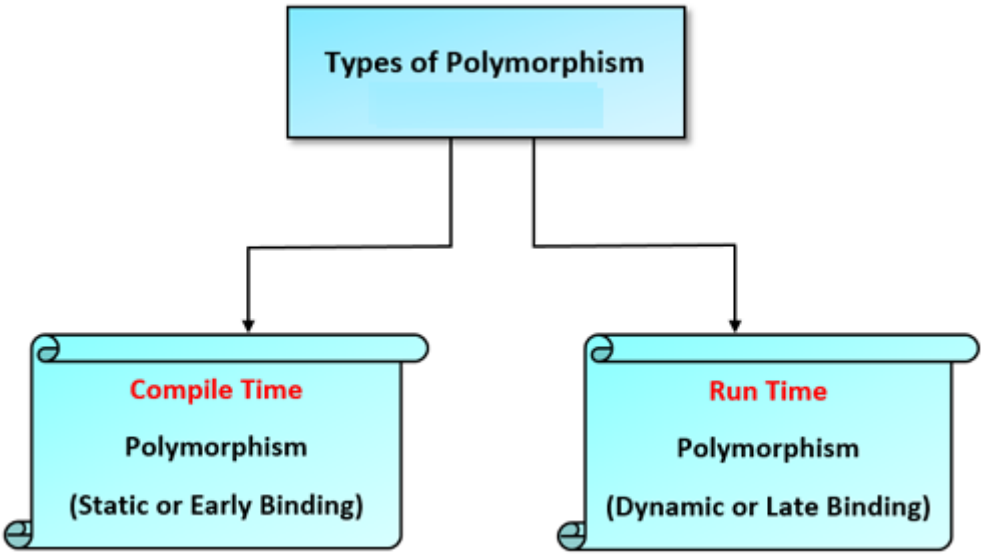
Encapsulation is also known as “data hiding“.

Polymorphism in Java

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.

In Java polymorphism is mainly divided into two types:

1) Compile time Polymorphism
2) Runtime Polymorphism



1) Compile time polymorphism:

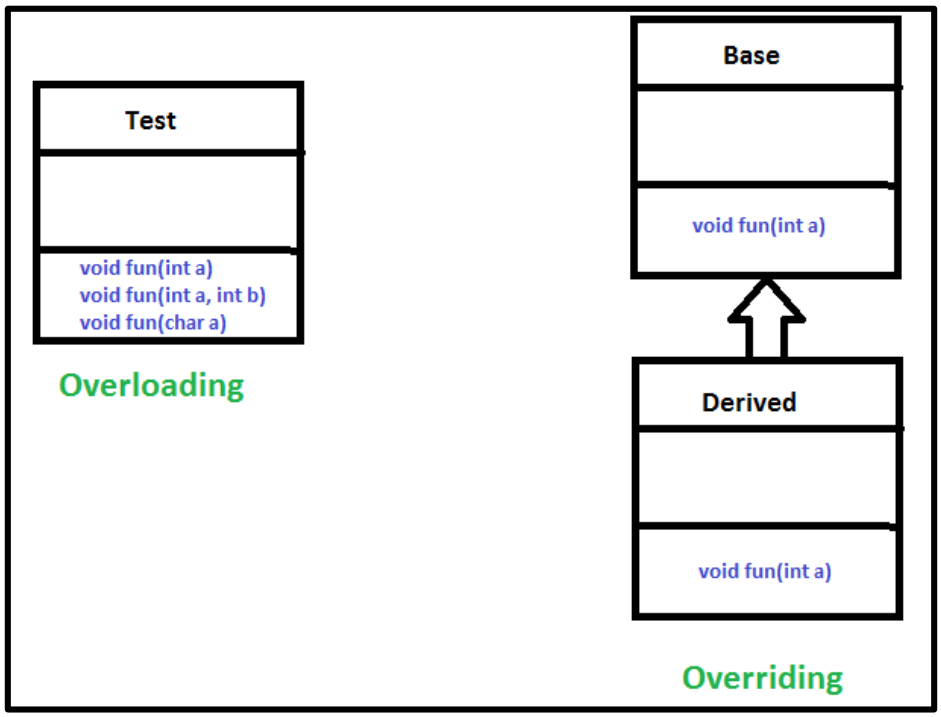
It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

- **Method Overloading:** When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments.
- **Operator Overloading:** Java also provide option to overload operators. For example, we can make the operator ('+') for string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands. So a single operator '+' when placed between integer operands, adds them and when placed between string operands, concatenates them.
  In java, only "+" operator can be overloaded.

2) Runtime polymorphism:

It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Method overriding, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.



Polymorphism Real-time examples:

Polymorphism is a concept, which allows us to redefine the way something works, by either changing how it is done or by changing the parts using which it is done. Both the ways have different terms for them.

Method Overloading Examples:

Example1: If we walk using our hands, and not legs, here we will change the parts used to perform something. Hence this is called **Overloading**.

Example2: Consider your Mobile phone. You can save your Contacts in it. Now suppose you want to save 2 numbers for one person. You can do it by saving the second number under the same name.

Similarly, in an object-oriented language like Java, suppose you want to save two numbers for one person. You must have a function, which will take the two numbers and the person name as arguments to some function void createContact(String name, int number1, int number2).

Now it's not necessary that every person will have 2 numbers. Many other contacts might have only a single number. In such a situation, instead of creating another method with different name to save one number for a contact, what you can do is that you can have the same name of the method i.e. createContact() but instead of taking 2 numbers as parameters, you can take only 1 number as parameter in it i.e. void createContact(String name, int number1).

This is Polymorphism. There is only one method named createContact(), but it has two definitions. Now which definition is to be executed depends upon the number of parameters being passed. If 1 parameter is passed, then only a single number is saved under the contact and if 2 numbers are passed to the same method name, then two numbers will be saved under the contact. This is also known as **Method Overloading**.

Method Overriding Examples:

Example1: And if there is a defined way of walking, but I wish to walk differently, but using my legs, like everyone else. Then I can walk like I want, this will be called as **Overriding**.

Example2: Suppose you go to an Ice Cream Parlor (ABC Ice Cream) near your home one day and you buy a vanilla flavored ice-cream. A week later, while traveling to the town nearby, you spot another Ice Cream Parlor (of the same chain, the ABC Ice Cream). You went to that shop and found a new variant of the Vanilla flavor ice-cream which had a twist of Chocolate flavor too in it. You really liked the new flavor. Once back home, you again went to the ice cream parlor near your home to get that amazing new flavor ice cream, but unfortunately, you couldn't, because that was a specialty of the parlor which was located in the neighboring town.

Now relating this to the functioning of an object-oriented language like Java, suppose you have a class named XIceCream which includes a method named icecream(). Using this method, you can get a vanilla flavor ice cream. For the ice cream parlor in the neighboring town, there is another class YIceCream. Both the classes XIceCream and YIceCream extends the parent class ABCIceCream. The YIceCream class includes a method named icecream(), using which you can get a vanilla and chocolate flavor ice cream.

Code:

```
class ABCIceCream {
   public void icecream() {
      System.out.println("Default Vanilla Icecream");
   }
}

class XIceCream extends ABCIceCream {
   public void icecream() {
      System.out.println("Default Vanilla Icecream");
   }
}

class YIceCream extends ABCIceCream {
   // This is overridden method
   public void icecream() {
      System.out.println("Vanilla with Chocolate Icecream");
   }
```

So, instead of creating different methods for every flavor, we can have a single method icecream(), which can be defined as per the different child classes. Thus, the method named icecream() has two definitions- one with only vanilla flavor and one with both vanilla with chocolate flavor.

Which method gets invoked depends upon the type of object i.e. the object is of which class. If you create ABCIceCream class object, then there will be only one vanilla flavor in available. But if you create YIceCream class object, that extends ABCIceCream class, then you can have both vanilla as well as vanilla with chocolate flavor. This is also known as **Method Overriding**.

Thus, Polymorphism makes the code more simple and readable. It also reduces the complexity of reading and saves many lines of codes. Polymorphism is a very useful concept in object-oriented programming and it can be applied in the real world scenarios as well.

Note: Java programming does not support static polymorphism because of its limitations and java always supports dynamic polymorphism.

Advantages of dynamic binding along with polymorphism with method overriding are.

- Less memory space
- Less execution time
- More performance

Advantages of Dynamic Polymorphism

Dynamic Polymorphism allows Java to support overriding of methods which is central for run-time polymorphism.

It allows a class to specify methods that will be common to all of its derivatives while allowing subclasses to define the specific implementation of some or all of those methods.

It also allows subclasses to add its specific methods subclasses to define the specific implementation of same.

Static polymorphism

The process of binding the overloaded method within object at compile time is known as Static polymorphism due to static polymorphism utilization of resources (main memory space) is poor because for each and every overloaded method a memory space is created at compile time when it binds with an object. In C++ environment the above problem can be solve by using dynamic polymorphism by implementing with virtual and pure virtual function so most of the C++ developer in real worlds follows only dynamic polymorphism.

Dynamic polymorphism

In dynamic polymorphism method of the program binds with an object at runtime the advantage of dynamic polymorphism is allocating the memory space for the method (either for overloaded method or for override method) at run time.

Conclusion

The advantage of dynamic polymorphism is effective utilization of the resources, So Java always use dynamic polymorphism. Java does not support static polymorphism because of its limitation.

Abstraction in Java

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

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

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

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

1) Abstract class in Java

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Abstract Method in Java

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

2) Interface in Java

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

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

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

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

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

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

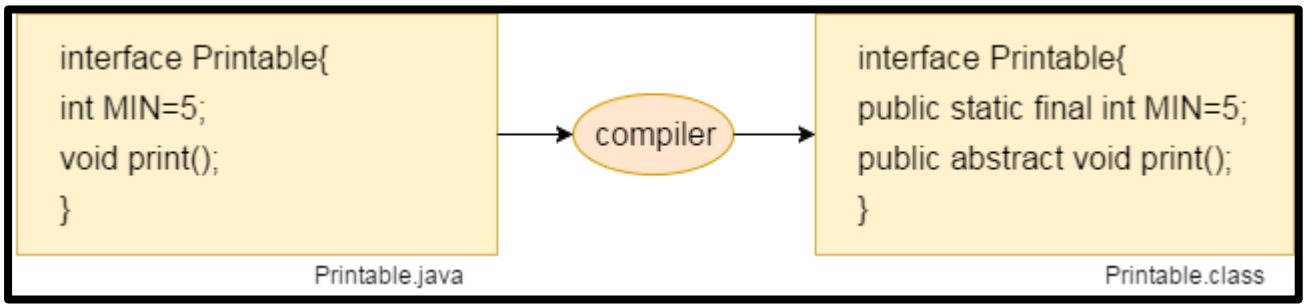
How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Internal addition by the compiler

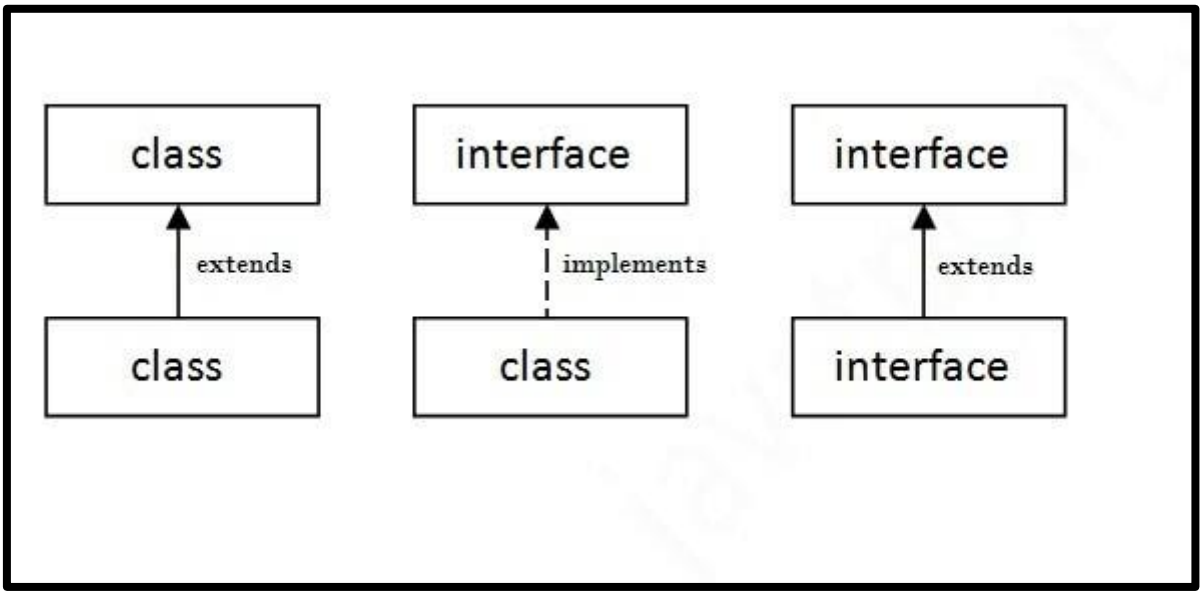
The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.



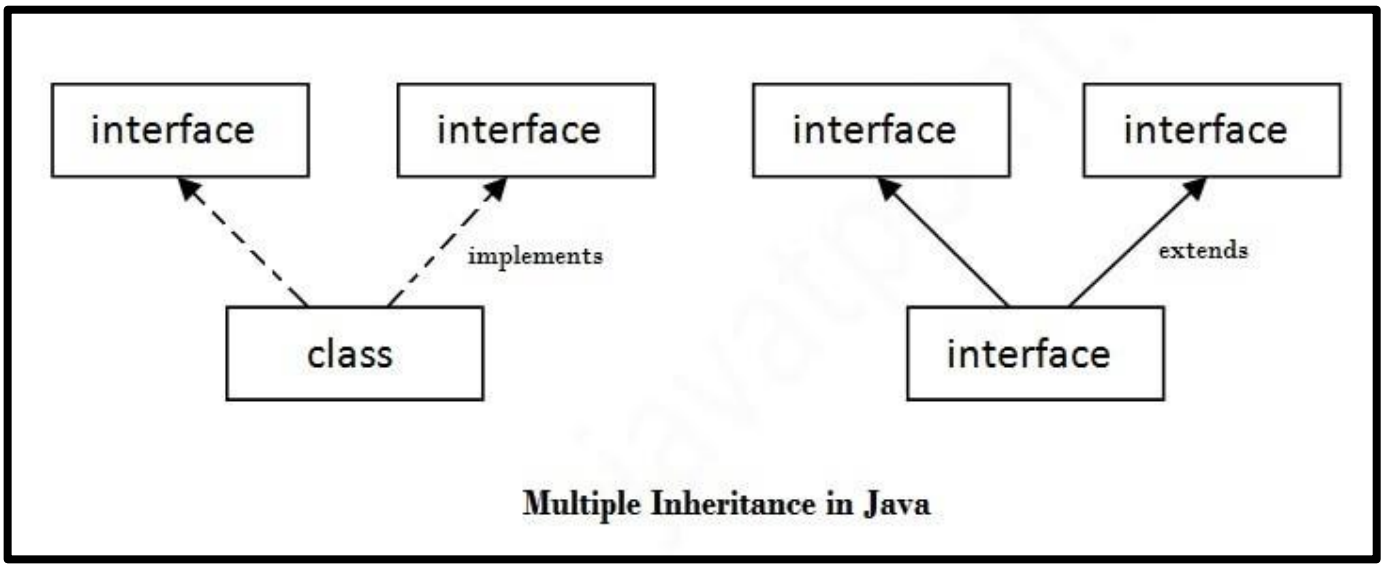
The relationship between classes and interfaces

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.



What is marker or tagged interface?

An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.