Java

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere.**

Java is −

* **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
* **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
* **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
* **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
* **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
* **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
* **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
* **Multithreaded** − With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
* **Interpreted** − Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
* **High Performance** − With the use of Just-In-Time compilers, Java enables high performance.
* **Distributed** − Java is designed for the distributed environment of the internet.
* **Dynamic** − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

**OOP in Java**

Class: is a template, blueprint, or contract that defines what an object’s data fields and methods will be.

Object: is an instance of a class. You can create many instances of a class.

Constructor: special method used to create new object.

Attributes: characteristics that define objects.

There are three main features of OOPS.

**1)** Encapsulation

**2)** Inheritance

**3)** Polymorphism

Let’s we discuss the features in details.

**Encapsulation**

Encapsulation means putting together all the variables (instance variables) and the methods into a single unit called Class. It also means hiding data and methods within an Object. Encapsulation provides the security that keeps data and methods safe from inadvertent changes. Programmers sometimes refer to encapsulation as using a “black box,” or a device that you can use without regard to the internal mechanisms. A programmer can access and use the methods and data contained in the black box but cannot change them. Below example shows Mobile class with properties, which can be set once while creating object using constructor arguments. Properties can be accessed using getXXX() methods which are having public access modifiers.

**Inheritance**

An important feature of object-oriented programs is inheritance—the ability to create classes that share the attributes and methods of existing classes, but with more specific features. Inheritance is mainly used for code reusability. So you are making use of already written the classes and further extending on that. That why we discussed the code reusability the concept. In general one line definition, we can tell that deriving a new class from existing class, it’s called as Inheritance. You can look into the following example for inheritance concept. Here we have Mobile class extended by other specific class like Android and Blackberry.

**Polymorphism**

In Core, Java Polymorphism is one of easy concept to understand. Polymorphism definition is that Poly means many and morphos means forms. It describes the feature of languages that allows the same word or symbol to be interpreted correctly in different situations based on the context. There are two types of Polymorphism available in Java. For example, in English, the verb “run” means different things if you use it with “a footrace,” a “business,” or “a computer.” You understand the meaning of “run” based on the other words used with it. Object-oriented programs are written so that the methods having the same name works differently in different context. Java provides two ways to implement polymorphism.

**Static Polymorphism (compile time polymorphism/ Method overloading):**

The ability to execute different method implementations by altering the argument used with the method name is known as method overloading. In below program, we have three print methods each with different arguments. When you properly overload a method, you can call it providing different argument lists, and the appropriate version of the method executes.

**Dynamic Polymorphism (run time polymorphism/ Method Overriding)**

When you create a subclass by extending an existing class, the new subclass contains data and methods that were defined in the original superclass. In other words, any child class object has all the attributes of its parent. Sometimes, however, the superclass data fields and methods are not entirely appropriate for the subclass objects; in these cases, you want to override the parent class members. Let’s take the example used in inheritance explanation.

**Abstraction**

All programming languages provide abstractions. It can be argued that the complexity of the problems you’re able to solve is directly related to the kind and quality of abstraction. An essential element of object-oriented programming is an abstraction. Humans manage complexity through abstraction. When you drive your car you do not have to be concerned with the exact internal working of your car(unless you are a mechanic). What you are concerned with is interacting with your car via its interfaces like steering wheel, brake pedal, accelerator pedal etc. Various manufacturers  of car have different implementation of the car working but its basic interface has not changed (i.e. you still use the steering wheel, brake pedal, accelerator pedal etc to interact with your car). Hence the knowledge you have of your car is abstract.

A powerful way to manage abstraction is through the use of hierarchical classifications. This allows you to layer the semantics of complex systems, breaking them into more manageable pieces. From the outside, a car is a single object. Once inside, you see that the car consists of several subsystems: steering, brakes, sound system, seat belts, heating, cellular phone, and so on. In turn, each of these subsystems is made up of more specialized units. For instance, the sound system consists of a radio, a CD player, and/or a tape player. The point is that you manage the complexity of the car (or any other complex system)through the use of hierarchical abstractions.

An abstract class is something which is incomplete and you can not create an instance of the abstract class. If you want to use it you need to make it complete or concrete by extending it. A class is called concrete if it does not contain any abstract method and implements all abstract method inherited from abstract class or interface it has implemented or extended. By the way, Java has a concept of abstract classes, abstract method but a variable can not be abstract in Java.

Let's take an example of Java Abstract Class called Vehicle. When I am creating a class called Vehicle, I know there should be methods like start() and Stop() but don't know start and stop mechanism of every vehicle since they could have different start and stop mechanism e.g some can be started by a kick or some can be by pressing buttons.

The advantage of Abstraction is if there is a new type of vehicle introduced we might just need to add one class which extends Vehicle Abstract class and implement specific methods.  The interface of start and stop method would be same.

package oopsconcept;

public abstract class VehicleAbstract {

public abstract void start();

public void stop(){

System.out.println("Stopping Vehicle in abstract class");

}

}

class TwoWheeler extends VehicleAbstract{

@Override

public void start() {

System.out.println("Starting Two Wheeler");

w }

}

class FourWheeler extends VehicleAbstract{

@Override

public void start() {

System.out.println("Starting Four Wheeler");

}

}

package oopsconcept;

public class VehicleTesting {

public static void main(String[] args) {

VehicleAbstract my2Wheeler = new TwoWheeler();

VehicleAbstract my4Wheeler = new FourWheeler();

my2Wheeler.start();

my2Wheeler.stop();

my4Wheeler.start();

my4Wheeler.stop();

}

}

Output :

**Summary**

* An object is an instance of a class.
* Encapsulation provides the security that keeps data and methods safe from inadvertent changes.
* Inheritance is a parent-child relationship of a class which is mainly used for code reusability.
* Polymorphism definition is that Poly means many and morphos means forms.
* Using abstraction one can simulate real world objects.
* Abstraction provides advantage of code reuse
* Abstraction enables program open for extension

WARM UP Problems

1. Monkey Trouble: We have two monkeys, a and b, and the parameters aSmile and bSmile indicate if each is smiling. We are in trouble if they are both smiling or if neither of them is smiling. Return true if we are in trouble.
2. Parrot Trouble: We have a loud talking parrot. The "hour" parameter is the current hour time in the range 0-23. We are in trouble if the parrot is talking and the hour is before 7 or after 20. Return true if we are in trouble.
3. Given a string, return a new string where the first and last chars have been exchanged.

Suppose the string "yak" is unlucky. Given a string, return a version where all the "yak" are removed, but the "a" can be any char. The "yak" strings will not overlap.  
stringYak("yakpak") → "pak"  
stringYak("pakyak") → "pak"  
stringYak("yak123ya") → "123ya"

1. Similar as Mobile class, create a model for Vehicle class.