Java Introduction & OOPS Concepts

By

Praveen Naga

Blog: https://praveennagatech.blogspot.com
Facebook Group: https://www.facebook.com/groups/268426377837151
Github repo: https://github.com/praveennaga
Email: opraveennaga@gmail.com

Buy me a Cup of Coffee:
https://docs.google.com/forms/d/e/1FAIpQLSfZxtgAxzMv83uwdEXezVFfLNSKlQgu7eDcMOeGtS2cRnOBrA/viewform?vc=0&c=0&w=1

Introduction To Java

James Gosling introduced Java on 1991 but it was released on 1995.

Java is a object oriented programming language and it is platform independent.

Java code(.java) -> compiler -> Byte code(.class)

class file (Runtime) -> classloader -> byte code verifier -> Interpeter ->runtime -> Hardware

Path: Path variable is set for providing path for all java tools like java, javac, javap, javah, jar,applet viewer.

set PATH="C:\Java\jdk\bin"
Classpath: classpath variable is set for providing path for all java classes which is used in our application.

set CLASSPATH="C:\Java\jre\lib\rt.jar"

set JAVA_HOME="c:\Java\jdk"

JVM (Java Virtual Machine)

It is a virtual machine used for converting the byte code to machine specific code for execution. JVM is platform dependent.

JVM performs following operations

- ✓ Loads the code
- ✓ Verifies the code
- ✓ Executes the code
- ✓ Provides the runtime environment

JRE (Java Runtime Environment)

JRE -> JVM + Libraries.

It is runtime environment and required to run a .class file. JRE is platform dependent.

JDK (Java Development Kit)

JDK-> JRE+ development tools

It is a complete kit which consists of everything which is required for creating, compile and debug a .java file and it is platform dependent.

JVM, JRE and JDK are platform dependent because configuration of each OS differs but java is platform independent.

setx path "%path%:; c:\Java\jdk\bin

Please check out my other ebook **Memory Management in Java** in

https://github.com/praveennaga/PraveenNaga-Tech-Ebooks/blob/master/Praveen%20Naga_%20Java%20Memory%20Management.pdf

OOPS Concepts

OOP (Object Oriented Programming)
It is a programming technique based on the concept of Class and Object.
Class
A class is a template which describe about data and behavior of an object.
Object
An object is an instance of class which can be related to real life example with its attributes (data) and methods (behavior).

Core OOPS concepts are:

- ✓ Abstraction.
- ✓ Encapsulation.
- ✓ Polymorphism.
- ✓ Inheritance.
- ✓ Association.
- ✓ Aggregation.
- ✓ Composition.
- ✓ Coupling.
- ✓ Cohesion.

1. Abstraction

It is a process of hiding the implementation details from the user, only functionality will be provided to the user.
This can be achieved by Abstract class and Interface.

Interface

- ✓ It is an abstract type that is used to specify a behaviour that class must implement
- ✓ By default the variables in an interface are public final static.

- ✓ By default the methods are public abstract but in Java 8 default, static concrete methods are also introduced.
- ✓ In Java 8, interface having one abstract method is denoted by annotation @FunctionalInterface.
- ✓ Interface can extend one or more interface.
- ✓ Interface can be nested inside another interface.

Abstract class

- ✓ An abstract class is a class with both abstract and concrete methods.
- ✓ An abstract class cannot be instantiated but they can be sub-classed.
- ✓ An abstract class can support non-static, non-final methods and attributes(protected, private in addition to public)
- ✓ An abstract class can hold state of the object.
- ✓ It can have constructor and member variables where as interface default methods cannot hold state and it cannot have constructor and member variables as well.

2. Encapsulation

Encapsulation or data binding is about packaging of data(variable) and behavior(method) together in a class.

a) Declare the variables of a class as private

b) Provide public setter and getter methods to modify and view the variable values.

```
public class CheckingAccount {
private double balance=0;
public void SetBalance(double bal) {
this.balance=bal;
}
public double getBalance() {
return balance;
}
}
```

3. Polymorphism

It is a concept by which we can perform a single action by different ways.

1. Overriding (dynamic/runtime)
2. Overloading (static/compile time)

Method Overriding

Rules:

- ✓ Defining multiple methods with same name,same parameters and same return type(or co-variant return type) in super class and sub class known as method overriding.
- ✓ We cannot override a private method of super class in sub class as private methods are not accessible in sub class.
- ✓ We cannot override a static method of super class in sub class as static means class level.

- ✓ We can change accessibility modifier in sub class overridden method but should increase the accessibility if we decrease compiler will throw an error message.

For Exceptions

- ✓ If sub class throws checked exception super class should throw same or super class exception of this.
- ✓ If Super class method throws an exception, then Subclass overriden method can throw the same exception or no exception, but must not throw parent exception of the exception thrown by Super class method.It means, if Super class method throws object of **NullPointerException** class, then Subclass method can either throw same exception, or can throw no exception, but it can never throw object of **Exception** class (parent of NullPointerException class).

Sample code:

```
package com.praveen.polymorphism;
public class MethodOverriding {
public static void main(String[] args) {
A a= new A();// In Class A
a.Show(); // show A
a.Print();// print A
//a.Display(); // compile time error Display method of A is not visible
B b= new B(); // In Class A In Class B
b.Show(); // show B
b.Print(); // print B
b.Display(); // display B
A ab= new B();// In Class A In Class B
ab.Show();// show B
ab.Print(); // print A
//ab.Display(); // compile time error Display method of A is not visible
ab.Check(); // check A
}

}

class A{
A(){
System.out.println(“In Class A”);
}
void Show() {
System.out.println(“show A”);
}

static void Print() {
```

```
System.out.println("print A");
}
private void Display() {
System.out.println("display A");
}

void Check() {
System.out.println("check A");
}
}
class B extends A{
B(){
System.out.println("In Class B");
}
void Show() {
System.out.println("show B");
}

static void Print() {
System.out.println("print B");
}

void Display() {
System.out.println("display B");
}
}
```

Output:

```
In Class A
show A
print A
In Class A
In Class B
show B
print B
display B
In Class A
In Class B
show B
print A
check A
```

Method Overloading

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

- ✓ There are 2 types to overload a method
  a) by changing number of arguments
  b) by changing the datatype
- ✓ Method overloading is not possible by changing the return type of method only.
  One type is promoted to another implicitly if no matching datatype is found.
- ✓ There will be ambiguity in some cases when there are no matching type arguments, each method promotes similar number of arguments.
  For example
  void sum(int a, long b) { s.o.p("a") ; }
  void sum(long a, long b) { s.o.p("b") ;}
  obj.sum(20, 20) ; // ambiguity // compile time error
  One type is not depromoted implicitly for example double cannot be depromoted to any type implicitly.
- ✓ Thrown exceptions from methods are not considered.

Normally we are developing our application interface base approach for example service and
serviceImpl so my controller need to inject serviceImpl to get business functionality so in this case in
controller class we are injection service bean by taking service interface reference as below .

```
public interface BankService {
public void doTransaction();
}
@Service
public class BankServiceImpl implements BankService{
@Override
public void doTransaction() {
// LOGIC
}
}
@Controller
public class BankController {
@Autowired(required = true)
private BankService service;
}
```

Here internally IOC container instantiate my service bean as below approach
BankService service=new BankServiceImpl();//Runtime polymorphism

4. Inheritance (IS-A relationship)

A subclass can inherit the states and behaviors of it super class is known as Inheritance.
Multiple inheritance is not possible due to ambiguity problem and cyclic inheritance is not allowed on Java.

5. Association (HAS-A relationship)

Association is a relationship between two classes where one class utilizes feature of another class. It is HAS-A relationship.

For example

- ✓ Student and ContactInfo.
- ✓ All Student HAS A ContactInfo
- ✓ Student and StudentIdCard
- ✓ All Student HAS A StudentIdCard

6. Aggregation

It is a type of association where it is HAS-A relationship and existence of both the class are not dependent on each other.

For example

- ✓ a department has several professors.
- ✓ Student has a ContactInfo. They are dependent on each other as ContactInfo can be used anywhere else for example Employee class.

7. Composition

It is a type of association where it is HAS-A relationship. In this relationship ClassB cannot exists without Class A whereas Class A can exist without ClassB.
For example, A university has several departments.
Student and StudentIdCard. StudentIDCard class dependent on Student Class as it cannot be used anywhere for example Employee class.

8. Coupling

Coupling indicates level of dependency between two classes.
In good software design, classes should be loosely coupled to each other for example Spring Framework.

9. Cohesion

Cohesion indicates level of logical connection of methods and attributes used in a class.
In good software design, Classes should be highly Cohesive.

Please check out my other ebooks in https://github.com/praveennaga/PraveenNaga-Tech-Ebooks

Buy me a Cup of Coffee:
https://docs.google.com/forms/d/e/1FAIpQLSfZxtgAxzMv83uwdEXezVFfLNSKlQgu7eDcMOeGtS2cRnOBrA/viewform?vc=0&c=0&w=1