Saturday, 3 October 2020

Java OOP(Object Oriented Programming)

Classes

- User defined blueprint or prototype from which objects are created.
- Represents set of properties or methods common to all object of one type.
- Class Declaration can include :
  - *Modifier*
  - *Class Name*
  - *Extends One Super Class*
  - Implements multiple comma separated Interfaces
  - *Curly braces* for body
- Types of Classes : Nested, Anonymous, Lambda.

Object

- Basic unit of OOP and represents real life entities.
- Consists of :
  - *State* : Attributes of the object
  - *Behaviour* : Methods of the object
  - *Identity* : Unique name of the object
- **Declaring Objects / Instantiating a class** :
  - When object of class is created , the class is said to be instantiated.
  - Single class may have many instances and they share state and behaviour of class.
  - For reference variable , type should be strictly class name; ex : Dog tuff;
  - *We cannot create objects of abstract class or interface*
  - Simply declaring reference variable do not create object and will have undetermined(null) value till object is created.

- **Initialising Object / Creating Objects**:
  - *Using new Keyword* :
    - Test test = new Test( )
    - Allocates memory for the new object and returns reference of the memory
  - *Using Class.forName(String className) method* :
    - Predefined class in Java lang package : Class
    - Class.forName(String className) loads class but does not create object
    - newInstance() of this Class object returns ew instance of class with given string name
    - Test obj = (Test) Class.forName("com.pckg.Test").newInstance();
  - *Using clone method* :
    - JVM actually creates new object and copies all content of previous object to it
    - Does not invoke any constructor
    - Should Implement Cloneable Interface and define clone in it otherwise will throw CloneNotSupportedException

```
// Java program to illustrate creation of Object
// using clone() method
public class CloneExample implements Cloneable
{
    @Override
    protected Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
    String name = "GeeksForGeeks";

    public static void main(String[] args)
    {
        CloneExample obj1 = new CloneExample();
        try
        {
            CloneExample obj2 = (CloneExample) obj1.clone();
            System.out.println(obj2.name);
        }
        catch (CloneNotSupportedException e)
        {
            e.printStackTrace();
        }
    }
}
```

- *Using Deserialization* :
  - Technique of reading an object from saved state in file

```
DeserializationExample d;
FileInputStream f = new FileInputStream("file.txt");
ObjectInputStream oos = new ObjectInputStream(f);
d = (DeserializationExample)oos.readObject();
```

- *Using newInstance() of Constructor* :
  - Present in java.lang.reflect.Constructor class
  - newInstance of Class internally calls this method

```
Constructor<ReflectionExample> constructor
                = ReflectionExample.class.getDeclaredConstructor();
          ReflectionExample r = constructor.newInstance();
```

Inheritance

- Mechanism by which one class is allowed to inherit the features(fields and methods) of another class.
- *Super Class* : Parent class whose features are inherited
- *Sub Class* : Child/Derived/Extended class inheriting the features
- Syntax :

```
class derived-class extends base-class
{
   //methods and fields
}
```

- In inheritance object of only sub class is created
- When object is created , a copy of all methods , fields of superclass is acquired memory in this object.
- Constructor of superclass not inherited as not members of class and can be invoked using super keyword.
- *Default super class* : Apart form Object Class , every other class has one super class and implicitly a subclass of object class
- Private Member Inheritance not allowed; Can be used if public or protected method to access them in super class present.

- **Types** :
  - *Single Inheritance* : Subclasses inherit the features of one superclass
  - *Multilevel Inheritance* : A derived class will be inheriting a base class and as well as the derived class also act as the base class to other class
  - *Hierarchal Inheritance* : one class serves as a superclass (base class) for more than one sub class
  - *Multiple Inheritance* : one class can have more than one superclass and inherit features from all parent classes; Achieved by Interface in Java
  - *Hybrid Inheritance* : It is a mix of two or more of the above types of inheritance; Achieved by Interface in Java

---

Encapsulation

- Mechanism of binding code and data it manipulates together in single unit
- Variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- Achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.
- **Advantages** :
  - Data Hiding
  - Increased Flexibility : Can make variables read only or write only depending on requirement
  - Reusability
  - Good for unit testing.

Abstraction

- Property by virtue of which only the essential details are displayed to the user.
- Abstraction is detail hiding(implementation hiding).
- Achieved by interfaces and abstract classes
- Advantages :
  - Reduced code complexity

  - Resusability
  - Avoid code duplication
  - Increase security

- **Abstract Classes and Methods:**
- Abstract classes declared with abstract keyword
- Abstract method are without implementation
- A method defined abstract must always be redefined in the subclass, thus making overriding compulsory OR either make subclass itself abstract.
- Abstract class can have parameterised / default constructor and its implementation
- Abstract class can not be instantiated directly with new keyword , hence no object

---

Dynamic Method Dispatch / Runtime Polymorphism

- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs.
- A superclass reference variable can refer to a subclass object. This is also known as upcasting; superclass obj = new subclass()
- **Advantages**:
  - Overriding of methods
  - Allows a class of methods to be common to all its derivatives, while allowing subclasses to define the specific implementation of some or all of those methods.
- **Static Binding vs Dynamic Binding**
- Static binding is done during compile-time while dynamic binding is done during run-time.

- Private, final and static methods and variables uses static binding and bonded by compiler while overridden methods are bonded during runtime based upon type of runtime object

---

Association, Aggregation and Composition

- **Association** :
  - Relation between two separate classes which establishes through their Objects
  - one-to-one, one-to-many, many-to-one, many-to-many
  - Ex : Bank has many employees
- **Aggregation** :
  - Unidirectional Association / Weak Association
  - Has - A relationship
  - Both entries can survive independently
  - Code reuse best achieved
  - Ex : Department has students not vice versa
- **Composition** :
  - Part of relationship
  - Child is dependent on parent
  - Strong Association
  - Ex : Books in library. If library burns no books left.

Access Modifiers and Non Access Modifiers

- **Access Modifiers : Used to control access mechanism**
  - *Public*
  - *Private*
  - *Protected*
  - Default
- **Non - Access Modifiers : Used with classes, methods, variables, constructors etc to provide information about their behavior to JVM**

- *Static*
- *Final*
- *Abstract*
- Volatile
- Transient
- Synchronised
- Native

	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