Agenda

- Revison
- polymorphism (DMD)
- upCasting
- Downcasting
- instanceof Operator
- final (method,class)
- Object class
- toString()
- equals() & hashcode()
- Abstract Class
- ~~Java 7 Interface~~
- ~~Marker Interface~~

Polymorphism

- ploy -> many
- morphism -> forms
- One entity taking multiple forms is called as polymorphism
- their are two types of polymorphism
    - 1. Compile Time Ploymorphism
        - It is acheived using method overloading
        - It is also called as Early binding
    - 2. Runtime Polymorphism
        - It is acheived using method overriding
        - It is also called as Late binding
        - It is also called as DMD (Dynamic Method Dispatch)

Upcasting (Demo01)

- Keeping the object of subclass inside reference of superclass is called as upcasting.
- Using the superclass reference we can call the overriden methods of sub class.
- using the superclass reference we cannot call the individual methods that belong to sub class.
- It is beacause of object slicing.

Downcasting (Demo01)

- Converting the reference of superclass into the subclass reference is called as downcasting.
- At the time of downcasting explicit typecasting is mandatory
- To call the individual methods given by the subclass we need to convert the reference of superclass into subclass.
- If upcasting is not done, i.e if the superclass reference is pointing at a differnt object then your downcasting fails.
- In java when downcasting fails we get an exception called as ClassCastException

instanceof (Demo01)

- It is an operator used to check for the instance of subclass object inside superclasss reference.
- It returns true if the instance is present and false if instance is not present
- To avoid ClassCastException it is recommended to check for the instance using instanceof operator.

```
Employee employee = new Manager();

if(employee instanceof Manager)
{
    Manager manager = (Manager)employee;
    manager.calculateTotalSalary();
}
```

final Keyword (Demo02)

- We can make below types as final
    - 1. variable
        - once initialized cannot change the value
    - 2. field
        - once initialized cannot change the value
    - 3. method
        - If implementation of method in superclass is 100% complete them make such method as final.
        - final methods cannot be overriden inside the subclass.
        - we can make all the getters and setters of the superclass as final.
    - 4. class
        - If the implementation of superclass is 100% complete then make the class as final
        - we cannot extend final classes
        - eg
            - java.lang.System
            - java.lang.Integer

Object class

- Object class is the superclass of all the classes in java.
- It directly or indirectly inherited in all the java given classes or in the classes that we are going to define.
- 11 Methods of object class
    - public final Class<?> getClass()
    - public int hashCode()
    - public boolean equals(Object obj)
    - protected Object clone() throws CloneNotSupportedException
    - public String toString()
    - public final void notify()
    - public final void notifyAll()
    - public final void wait(long timeout) throws InterruptedException
    - public final void wait(long timeout,int nanos) throws InterruptedException

    - public final void wait()throws InterruptedException
    - protected void finalize() throws Throwable
- All the methods of object classs are inherited into the subclasses
- final methods declared inside the object class cannot be overriden
- Object class have only 1 constructor which is a paramterless ctor

toString() (Demo03)

- It is a non final method declared inside object class.
- This method is provided by the object class so that we can override it to return the state of an object in string format in human readabale form.
- It is recommended to override toString() in all the subclasses
- the object class toString() returns the state of on obeject inthe form of fully qualified class name @ hashcode

```
public String toString(){
    return getClass().getName() + '@' + Integer.toHexString(hashCode());
}
```

- To print the state of object in human readable form return the state in string format from the toString() by overiding it into the subclass

```
@Override
    public String toString() {
        return day+"/"+month+"/"+year;
    }
```

equals() (Demo03)

- It is reflexive: for any non-null reference value x, x.equals(x) should return true.
- It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
- It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
- It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
- For any non-null reference value x, x.equals(null) should return false.

hashcode() (Demo03)

- It is a method of the object class which is responsible to generate hashcode for every objet.
- If the state of two objects is equal then their hash value should also be same.
- It is recommended to overide the hashcode() at the time of overriding of equals method.

Abstract (Demo04)

- abstract is an keyword in java
- It is an accesss modifier
- We can make the method and class as abstract
- If the implemention of the method is 100% incomplete, i.e we dont kow the implementation of the method in superclass.
- Such methods can only be declared without the body with the help of abstract keyword.
- such methods are called as abstract methods
- the class in which abstact methods exists, that class has to be made as abstract.
- we cannot create object of the abstract, we can only create the reference
- If we want to create the objet of abstact class, inherit such class inside a subclass, and we can then create the object of the subclass and assign it ti the reference of the super class.
- To create the object of the subclass, it is compulsary for the subclass to override all the abstarct methods of the superclass.
- Abstract classes are used to keep the method design same across related types.
- Abstact classes can have fields as well as constructors
- Abstact classes can have abstract as well as non abstract methods.