Agenda

- Revision
- Singleton Design Pattern
- Association
- Inheritance
- super keyword
- Types of inheritance
- Method Overriding
- Upcasting & Downcasting
- Final Method & Class
- Object class
    - Methods of object class
        - toString();
        - equals();

Singleton Desing Patttern (Demo01)

- It is a design pattern which is used to provide only single object of the class.
- to make a class singleton we have to make all its constructors as private.
- add a static field inside a class of the same type of that of the class which will hold the single object of the current class.
- Initialize this static field to null using field initializer Or you can initialize it with new object inside static block.
- provide a getter for this static field which will return the reference that points to the single object of the current class.

Association (Demo02)

- When has-a relation ship exists between two entities, use association
- it can be further divided as
    - 1. Composition
        - If the entities are tightly coupled we use composition
        - eg
            - Human has-a heart
            - Car has-a engine
    - 2. Aggegration
        - If the two entities are loosely coupled we use aggegration
        - eg
            - Employee has-a vehicle
            - Room has-a window
- In association we create object of one class as a field inside another class.

Inheritance (Demo03)

- If is-a relationship exists between two entities, we use inheritance
- If their is inheritance between two classes, one class is called as parent class and another is called as child class.
- In java, parent class is also called as Super class and child class is called as sub class.
- All the members of the Superclass gets inherited into the subclass.
- to perform inheritance in java we use extends keyword
- read the theory of inheritance from the CPP demos.
- eg
    - Employee is a person
    - Manager is a employee
    - Rectangle is a shape

super

- It is a keyword in java.
- It is used to invoke the Parent/Superclass constructors from the child/subclass constructor
- super statement should be the first statement inside tthe subclass constructor.
- super is also used to unhide the methods of superclass inside the sub class.

Types of Inheritance

- 1. Single Inheritance
    - When subclass is extended from one superclass.

```
class A{
}
class B extends A{
}
```

- 2. Multilevel Inheritance
    - When subclass is extended from one superclass and then we have another sub class that is extended from the previous subclass.

```
class A{
}
class B extends A{
}
class C extends B{
}
```

- 3. Multiple Inheritance
    - Java does not support multiple implementation(class) inheritance

```
class A{
}
```

```
class B {
}
class C extends B,A{} // NOT Supported
```

- Java does support multiple interface inheritance

- 4. Hirerachical Inheritance
  - multiple sub classes extending a single super class

```
class A{
}
class B extends A {
}
class C extends A{
}
```

- 5. Hybrid Inheritance
  - mixture of any two types of above inheritances.

Polymorphism

- poly = many , morphism = forms
- their are two types of ploymorphism
  - 1. Compile time
    - it is acheived using method overloading
  - 2. Runtime
    - it is acheived using method overriding

Method Overriding (Demo04)

- redefining the method of super class once again inside the sub class with same name and signature is called as method overriding
- method overrriding is done when
  - 1. Super class methods are abstract
  - 2. super class method implemetation is incomplete or partial completed
  - 3. we require complete differnt implementation for the methods of the super class inside subclass.
- when we override the methods then the super class methods gets hidden by the overriden methods of the subclass.
- we can call the methods of the superclass from the sub class methods using this if the method overrding is not done.
- we can call the methods of the superclass from the sub class methods using super if the method overrding done.
- method overiding is useful to perform run time polymorphism
- run time polymorphism is also called as DMD (Dynamic Method Dispatch)
- In java 5 an annotation named @override was added with is used to check for any human errors at the time of performing method overriding

- while performing method overriding remember below points
    1. keep the name of the method same.
    2. keep the signature of the method same.
    3. keep the return type of method same or the sub type of the return type of super class method.
    4. access modifier should be of same as that of super class method or of wider visibilility
    5. If super class method is throwing checked exceptions then the method in the sub class should throw the same exceptions or the subset of the exceptions that are thrown.

Upcasting and Downcasting (Demo04-> Program02)

- Keeping the object in super class reference is called as upcasting
- when upcasting is done the super class reference can only point at the super class methods that are inherited in the sub class.
- if the methods are overriden then these overriden methods will get called.
- super class reference cannot point at the sub class methods.
- this is called as object slicing
- to call the sub class methods we have to convert the reference of super class into sub clss.
- converting the reference of super class into sub class is called as downcasting.
- at the time of downcasting explicit typecasting is mandatory.
- if downcasting fails the JVM throws an exception ClassCastException.
- to avoid such riskier downcasting it is better to check for the instance of subclass is creted or not.
- to check for the instance java had provided an operator called as 'instanceof'
- instanceof operator checks for weather the reference is having the specified instance or not.
- it return true if the instance is present and false it the instance is diffrerent then the specified one.

Demo05 - > Implemenation of inheritance,uocasting,downcasting, method overriding

Final (Demo06)

- Go through the previous notes of final variable and field
- we can also make method and class as final
- Method as final
    - make the methods final only when the implementation of the method is 100% complete.
    - subclasses cannot override final methods.
- Class as final
    - make the class final onnly when the mplementation of the class is 100% complete.
    - we cannot extend final classes
    - eg
        - java.lang.System class

Object class

- Object class is the super class of all the classes in java.
- It is either directly or indirectly superclass of all the classes.
- Even if we design a class it is still inheriting the object class.
- object class is defined into java.lang package

- object class has only 1 constructor which is a prameterless constructor

Methods of Object class

- Their are 11 methods in 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

toString() (Demo07-> Program01)

- this is the method of Object class which gets inherited into all the classes of java.
- this method is used to return the state of the object in string format.
- the purpose of this method is to provide the human redable state of an object in string format.

```
// implementation of toString in Object class
public String toString(){
return getClass().getName() + '@' + Integer.toHexString(hashCode())
}
```

equals() (Demo07 -> Program02)

- Indicates whether some other object is "equal to" this one.
- The equals method implements an equality relation on non-null object references:
- equals methed shoud be
    - reflexive
        - x.equals(x) should return true
    - symmetric
        - if x.equals(y) is true then y.equals(x) should be also true
    - transitive
        - if x.equals(y) is true and y.equals(z) is true then x.equals(z) should be also true

- consistent
  - multiple call of x.equals(y) should consistently return true or consistently return false