Day 6

Inheritance

```
- According to client's requirement, if implementation of super class
method is partialy complete or logically incomplete then we should
redifine method in sub class. In other words we should override method in
sub class.
- If name of super class members[ field/method] and sub class members are
same then sub class members hides implementation of super class members
hence preference is always given to sub class members. This is called as
shadowing.
- If name of super & sub class method is same and if we try to call such
method on sub class instance then due to shadowing preference is given to
sub class method.
- If we want to access members of super class, inside method of sub class
then we should use super keyword.
- Without changing implementation of exisiting class, if we want to extend
meaning of that class then we should use inheritance.
```

Typing

```
- It is minor pillar of oops
- Typing is also called as polymorphism
- polymorphism it is greek word wich combination of "Poly"(means many )
and "morphism"( means forms/behavior).
- An ability of object to take multiple forms is called polymorphism.
- Main purpose of typing/polymorphism is to reduce maintenance of system.
- Types of polymorphism
    - ** Compile time polymorphism **
    - ** Run time polymorphism **
```

Compile time polymorphism

```
- It is also called as static polymorphism/ early binding / weak typing /
false polymorphism
- We can achive it using method overloading
```

Run time polymorphism

```
- It is also called as dynamic polymorphism/ late binding / strong typing
/ true polymorphism
- We can achive it using method overriding
```

Dynamic Method Dispatch(also called as Runtime polymorphism)

```
- During inheritance, members of sub class do not inherit into super class
hence using super class instance, we can access members of super class
only.
```

```
    Person p = new Person("Sandeep", 36);
        //p.showRecord();       //OK : Person.showRecord
        //p.printRecord();      //Ok : Person.printRecord
        //p.displayRecord();//Error displayRecord is method of sub class
```

```
- During inheritance, members of super class inherit into sub class hence
using sub class instance we can access members of super class as well as
sub class.
```

```
    Employee emp = new Employee("Sandeep", 36, 33, 45000);
    //emp.showRecord();         //OK : Person.showRecord
    //emp.printRecord();        //OK : Due to shadowing :
Employee.printRecord
    //emp.displayRecord();      //OK:Employee.displayRecord();
```

```
- Since members of super class inherit into sub class, we can consider sub
class instance( e.g employee instance ) as a super class instance( e.g
person instance ).
- Since sub class instance can be considered as super class instance, we
can use it in place of super class instance.
```

```
    Person p1 = new Person();   //Ok
        Person p2 = new Employee();     //OK
```

Upcasting

```
- It is the process of converting, reference of sub class into reference
of super class.
- Main purpose of upcasting is to reduce instance/object dependancy in the
code.
```

```
    Employee emp = new Employee();
        //Person p = ( Person)emp;      //Upcasting
        Person p = emp; //Upcasting
```

```
- As shown in above code, in case of upcasting, explict typecasting is
optional.
- In case of upcasting, using super class reference variable we can access
fields of super class only.
- ( Definition 2)Super class reference variable can contain reference of
sub class instance. It is called upcasting.
- example : Person p = new Employee();  //Upcasting
```

Downcasting

```
- Process of converting reference of super class into reference of sub
class is called downcasting.
- In case of upcasting, if we want to access sub class specific members
then we should do downcasting.
```

```
    Person p = new Employee();  //Ucpasting
    p.name = "Sandeep";
    p.age = 36;
    Employee emp = (Employee) p;        //Downcasting
    emp.empid = 33;
    emp.salary = 45000;
    emp.displayRecord();
```

```
- As shown in above code, in case of downcasting, explict typecasting is
mandatory.
- In case of inheritance, members of sub class do not inherit into super
class hence, super class instance can not be considered as sub class
instance.
- Example: Every employee is a person but every person is not a employee.
```

- Since super class instance can not be considered as sub class instance, we can not use it in place of sub class instance.

```
    Employee emp1 = new Employee();     //Ok
    Employee emp2 = new Person();       //Not Ok
```

```
        Person p = null;
    System.out.println(p);      //null
    Employee emp = (Employee) p;        //Downcasting
    System.out.println(emp);    //null
```

```
    Person p = new Employee();  //Upcasting : OK
    Employee emp = (Employee) p;        //Downcasting : OK
```

- If dowcasting fails then JVM throws ClassCastException

```
    Person p = new Person();    //Ok
        Employee emp = (Employee) p;    //ClassCastException
```

- Process of calling method of sub class using reference of super class is called dynamic method dispatch.

```
    Person p = new Employee("Sandeep",36,33,45000);     //Upcasting
        p.printRecord();        //DMD
```

- Process redefining method of super inside sub class is called method overriding

Rules of method Overriding

1. Access modifier in sub class method should be same / it should be wider.
2. Return Type in sub class method should be same or it should be sub

```
type( interface/class).
3. Method name , number of parameters and type of parameters in sub class
method must be same.
4. Checked exception list in sub class method should be same or it should
be sub set.
- ** We can not override following methods in sub class **
    1. constructor
    2. static method
    3. final method
    4. private method
```

> Can we override static method? Why? - Overridden methods are designed access using object referece. - Static methods are designed to access using class name. - Since static methods are not designe to call using object reference we can not override it.

Override

```
- It is annotation declared in java.lang package.
- It is introduced in java SE 5 ( jdk1.5 ).
- Developer should use this annotation to override method.
- It help us to override method prperly.
```

Difference between operator== and equals method

```
- If we want to compare state of variable of primitive /value type then we
should use operator ==.
```

```
    int num1 = 10;
    int num2 = 10;
    if( num1 == num2 )
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //output : Equal
```

```
- We can use operator == with variables of reference type ( i.e object
reference ).
- If we want to compare state of object reference then we should use
operator ==.
```

```
        public static void main(String[] args)
        {
                Employee emp1  = new Employee("Sandeep", 33, 65000);
                Employee emp2  = new Employee("Sandeep", 33, 65000);
                if( emp1 == emp2 )
                        System.out.println("Equal");
                else
                        System.out.println("Not Equal");
                //Output : Not Equal
        }
```

- "equals" is non final method of java.lang.Object class
- Syntax:
  "public boolean equals(Object obj)"
- If we want to compare state of instance of reference type then we should use equals method.
- Following code define equals method of object class

```
public class Object
{
    public boolean equals(Object obj)
    {
        return (this == obj);
    }
}
```

- If we do not define equals method inside class then its super class's equals method gets called.
- equals method of java.lang.Object class do not compare state of instances rather it compares state of references.
- If we want to compare state of instances then we should override method in sub class.

```
        //Employee this = emp1;
        //Object obj = emp2;    //Upcasting
        @Override
        public boolean equals( Object obj )
        {
                if( obj != null )
                {
                        Employee other = (Employee) obj;
//Downcasting
                        if( this.empid == other.empid )
```

```
                                return true;
                }
                return false;
        }
```

```
- We can use equals method with object reference but we can not use it
with varibles of pritive/value type.
```

> What is difference between operator == and equals method?

Final method and class

```
- If implementation of super class method is logically 100% complete then
we should declare super class method final.
- Final method inherit into sub class but we can not override it into sub
class.
- Overriden method can be declared as final.
- If implementation of any class is logically 100% complete then we should
declare such class final.
- We can not create sub class of final class but we can instantiate final
class.
- Example of final methods:
    1. getClass( )
    2. wait( )
    3. notify( )
    4. notifyAll( )
- Example of final class:
    1. java.lang.Sytem
    2. java.lang.Math
    3. All Wrapper classes
    4. java.lang.String/StringBuffer/StringBuilder
    5. java.util.Scanner
```

Abstract method and class

```
- If implementation of super class method is logically 100% incomplete
then we should declare super class method abstract.
- abstract is keyword in java.
- We can not provide body for abstract method.
- If method is abstract then it is mandatory to declare class abstract.
- Without declaring method abstract, we can declare class abstract.
- We can not instantiate abstract class but we can create reference of
abstract class.
- If we extend abstract class then
```

```
    1. either we should override method in sub class
    2. or we should declare sub class abstract
```

```
abstract class A
{
    public abstract void f1();
}
class B extends A   //Ok
{
    @Override
    public void f1( )
    {   }
}
```

or

```
abstract class A
{
    public abstract void f1();
}
abstract class B extends A   //Ok
{   }
```

```
- Example of Abstract class
    1. java.lang.Number
    2. java.lang.Enum
    3. java.util.Calendar
    4. java.util.Dictionary
```

instanceof

```
- It is operator in java which returns boolean value.
- If we want to check inheritance relationship at runtime then we should
use instanceof operator. In other words if we want to check, reference of
which sub class instance is stored in super class refernce variable then
we should use instanceof operator.
```

```
    private static void acceptRecord(Shape shape)
        {
                if( shape instanceof Rectangle )
                {
                        //TODO
```

```
                }
                else
                {
                        //TODO
                }
        }
```

Sole Constructor

```
- Constructor of super class, that is designed to call from constructor of
sub class only, is called sole constructor.
```

```
abstract class A
{
        private int num1;
        private int num2;
        public A( int num1, int num2 )  //Sole Constructor
        {
                this.num1 = num1;
                this.num2 = num2;
        }
        public void printRecord( )
        {
                System.out.println("Num1        :       "+this.num1);
                System.out.println("Num2        :       "+this.num2);
        }
}
class B extends A
{
        private int num3;
        public B( int num1, int num2, int num3 )
        {
                super( num1, num2 );
                this.num3 = num3;
        }
        public void printRecord( )
        {
                super.printRecord();
                System.out.println("Num3        :       "+this.num3);
        }
}
public class Program
{
        public static void main(String[] args)
        {
                A a  = new B( 10,20,30);        //Upcasting
                a.printRecord();        //DMD
        }
}
```