OOPJ Notes Day-7 Date: 04/05/2023

Inheritance

```
class SingleInherit {

    int a;

    SingleInherit()
    {
        System.out.println("Am a Default user defined constructor of parent class");
    }

    void Getdata()
    {

        System.out.println("Am Get data of Parent class"+a);
    }
    void SetData()
    {
        System.out.println("Am Set data of Parent class");
    }

}

class SingleInheritChild extends SingleInherit
{
    int b;
    public SingleInheritChild()
    {
    super(); //
    System.out.println("Am def cons of child class");
    }
    void gdata()
    {
        //super.Getdata();
        System.out.println("Am g data of child class"+b);
    }
    void sData()
    {

        System.out.println("Am s data of child class");
    }

}
 class SingleInheritTest
 {
     public static void main(String args[])
     {
         SingleInheritChild ch1=new SingleInheritChild();
```

```
            ch1.a=10;
            ch1.b=20;
            ch1.gdata();
        }
    }
```

Constructor calling and super statement

- constructor of child class call constructor of parent class implicityly whenever the instance of child class created.

Types of Inheritance

1. Single Level Inheritance
2. Multi-Level Inheritance
3. Hierarchical Inheritance

```java
class Shape
{
    void Area()
    {
        System.out.println("Am Area of Shape");
    }
}
class Circle extends Shape
{
    void Area()
    {
        System.out.println("Am Area of circle");
    }
}
class Rectangle extends Shape
{
    void Area()
    {
        System.out.println("Am Area of Rect");
    }
}


class HierInheritDemo {

    public static void main(String[] args) {


        Shape sp1=new Circle();


    }


}
```

4. Hybrid Inheritance

Multiple inheritance in Java

- Java does not support multiple inheritance directly.
- But we can achieve this using Interface

Access modifiers

1. Default
2. Public
3. Protected
4. Private

Object Upcasting and Downcasting

```java
class Student {
    int RollNo;
    int fees;
    void AcceptRecord()
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the Roll No:");
        RollNo=sc.nextInt();
        System.out.println("Enter the Fees:");
        fees=sc.nextInt();
    }
    void PrintRecord()
    {
        System.out.println("Student Information");
        System.out.println(RollNo+"    "+fees);
    }

}
class Admin extends Student
{
    String CourseName;

    public void AcceptRecord()
    {
        super.AcceptRecord();
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Course Name:");
        CourseName=sc.nextLine();

    }
    public void PrintRecord()
    {
        super.PrintRecord();
        System.out.println("Course Information:");
        System.out.println(CourseName);

    }
}
public class MethodOverTest {

    public static void main(String[] args) {

        Student a1=new Admin();  //Object Upcasting
        a1.AcceptRecord();
        a1.PrintRecord();

    }
}
```

Method overridng and its rule

- Concept Defining method in child class with same name and signature which is already defined in base class is known as method overidding.
- method overridding can be achieved only in case of inheritance

```java
import java.util.Scanner;

class Student {
    int RollNo;
    int fees;
    void AcceptRecord()
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the Roll No:");
        RollNo=sc.nextInt();
        System.out.println("Enter the Fees:");
        fees=sc.nextInt();
    }
    void PrintRecord()
    {
        System.out.println("Student Information");
        System.out.println(RollNo+"    "+fees);
    }

}
class Admin extends Student
{
    String CourseName;

    public void AcceptRecord()
    {
        super.AcceptRecord();
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Course Name:");
        CourseName=sc.nextLine();

    }
    public void PrintRecord()
    {
        super.PrintRecord();
        System.out.println("Course Information:");
        System.out.println(CourseName);

    }
}
public class MethodOverTest {

    public static void main(String[] args) {

        Admin a1=new Admin();
        a1.AcceptRecord();
        a1.PrintRecord();

    }
}
```

Object Slicing

```
class A
{
    int a=2;
    int b=3;
    int e=123;
}
class B extends A
{
    int c;
    int d;
}

class ObjectSlicingTest {

    public static void main(String[] args) {
        A a1=new A();

        B b1=new B();

        b1.a=10;
        b1.b=30;
        b1.c=40;
        b1.d=50;
        System.out.println("Value inside b1:");
        System.out.println(b1.a+" "+b1.b+" "+b1.c+" "+b1.d);
        a1=b1;

        System.out.println("Value inside a1:");
        System.out.println(a1.a+" "+a1.b+" "+a1.e);
    }

}
```

Dynamic method dispatch

- To acheive this there should be inheritance upto multi-level
- There should be method overridding
- The method you want to be dispatched dynamically that method should be asbtract in super parent class. (which have no body)
- That means you need to make super parent class as abstract.
- There should be object upcasting: