COJ :: Polymorphism

TalentSprint

Licensed To Skill

Version 1.0.4

Learning Objectives

By the end of this session, you will be able to:

- Polymorphism and Overriding

Polymorphism

Exercise 1

Create the following Employee, Manager, Clerk, SalesPerson classes in java.

– Employee
Instance variables : name, salaryBasic, HRAPer, DAPer, PT.
Methods : computePayroll()

– Manager (a sub **class** of Employee)
Instance variable : projectAllowance
Methods : computePayroll()

– Clerk (a sub **class** of Employee)
Instance variable : **int** typingSpeed , **int** typingAccuracy
Methods : computePayroll()

– SalesPerson (a sub **class** of Employee)
Instance variable : noOfTargetsCompleted, perkTarget
Methods : computePayroll()

Create appropriate constructors **for** all the classes.

Create a **class** with main and add a displaySalary() method which takes one parameter of type employee and complete the application by using the computePayroll() method in displaySalary() method.

Polymorphism

Polymorphism is a concept where a single name may denote objects of different classes that are related by some common base class.
Polymorphism is the ability to create an attribute, a method, or an object that has more than one form.

Polymorphism

- A polymorphic reference variable can refer to different types of objects at different times
- In java every reference can be polymorphic except of references to base types and final classes
- It is the type of the object being referenced, not the reference type, that determines which method is invoked
- Polymorphic references are therefore resolved at run-time, not during compilation; this is called dynamic binding

Polymorphism

Example: Overriding Methods

```
class Employee {
    int empId;
    String name;
    Employee(int id,String eName) {
        empId = id;
        name = eName
    }
    void display() {
        System.out.println("id and name: "
                + id + "  " + name);
    }
}
```

```
class Manager extends Employee {
    double projAllowance;
    Manager(int id,String name double pAllowance) {
        super(id, name);
        projAllowance = pAllowance;
    }
    void display() {
        System.out.println("id, name and pAllowance:" + id +
                "  "+ name + " " + projAllowance);
    }
}
```

```
class MainClass{
    public static void main(String args[]){
        Employee e1 = new Employee(11,"scott");
        Manager m1 = new Manager(23,"roy",300.00);
        e1.display();
        m1.display();
    }
}
```

id and name: 11 scott

id , name and pAllowance: 23 roy 300.00

Polymorphism

How does it work?

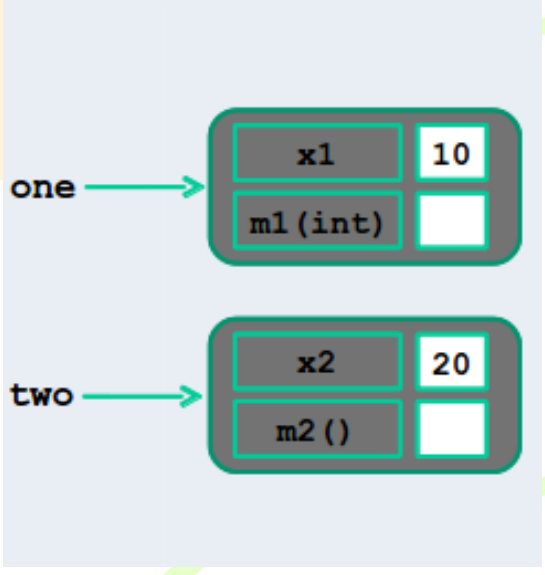
What happened to the **display()** method of super class while calling **m1.display()**

Can we access the **display()** method of the super class Employee from sub class Manager ?

Polymorphism

Overriding

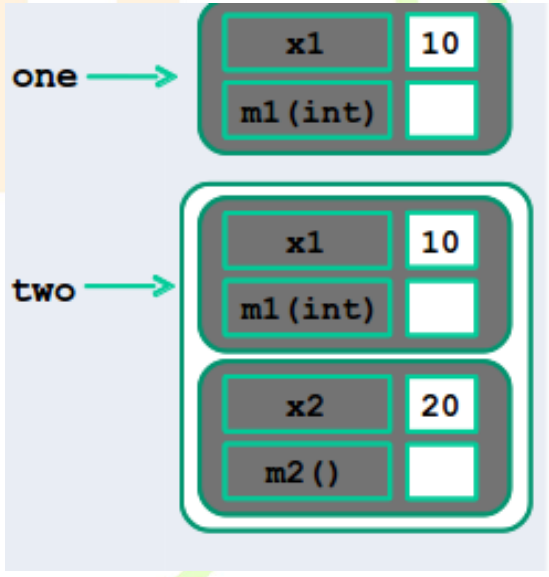
```
class  One {
      int x1 = 10;
      public int m1(int x1) {
            return x1 * x1;
      }
}

class Two {
      int x2 = 20;
      public int m2() {
            return x2;
      }
}

class MainClass {
   pubic static void main(String args[]) {
         One one = new One();
         Two two = new Two();
      }
}
```



Polymorphism

Overriding

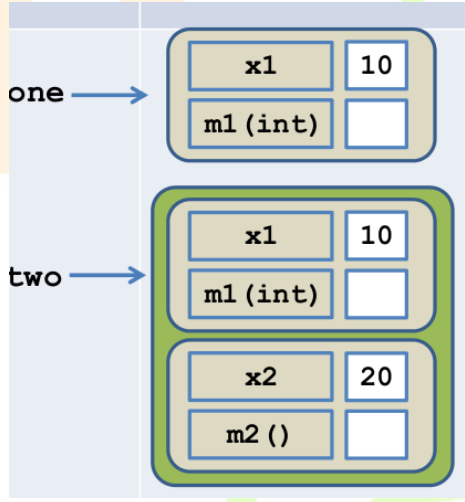
```
class  SuperClass {
    int x1 = 10;
    public int m1(int value) {
        return value * value;
    }
}
class SubClass extends SuperClass {
    int x2 = 20;
    public int m2(){
        return x2;
    }
}
class MainClass {
    pubic static void main(String args[]) {
        SuperClass one = new SuperClass();
        SubClass two = new SubClass();
    }
}
```



Polymorphism

Overriding

```
class  SuperClass{
int x1 = 10;
public int m1(int value){
return value * value;
}
}
class SubClass extends SuperClass{
int x2 = 20;
public int m2(){
return x2;}
}
class MainClass{
pubic static void main(String args[]){
SuperClass one = new SuperClass();

  SubClass two = new SubClass();

}
}
```

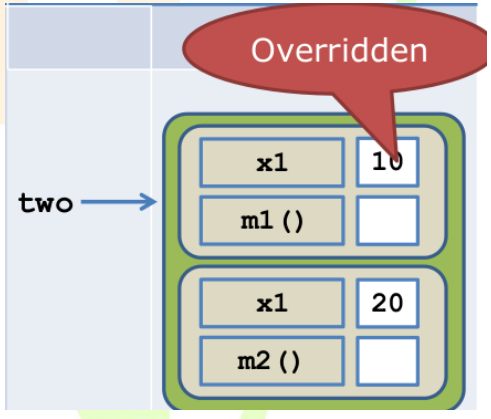


Polymorphism

Overriding

```
class  SuperClass{

  int x1 = 10;
public int m1(){
return x1 * x1;
}
}
class SubClass extends SuperClass{

 int  x1=20;
public int m2(){
return x1;
}
}
class MainClass{
pubic static void main(String args[]){
SubClass two = new SubClass();

System.out.println(two.x1);
}
}
```

Output:
20



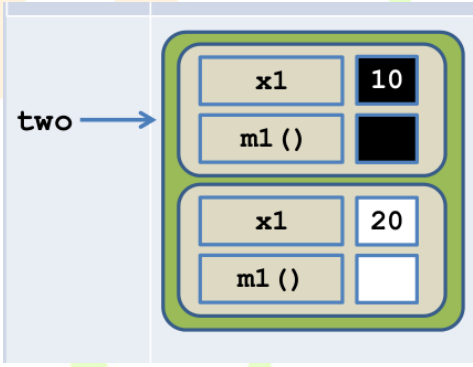
Polymorphism

Overriding

```
class  SuperClass{
int x1 = 10;
public int m1(){
return x1 * x1;
}
}
class SubClass extends SuperClass{
int x1 = 20;
public int m1(){
return x1;
}
}
class MainClass{
pubic static void main(String args[]){
SubClass two = new SubClass();
System.out.println(two.x1);
System.out.println(two.m1());
}
}
```

Output:

20
20



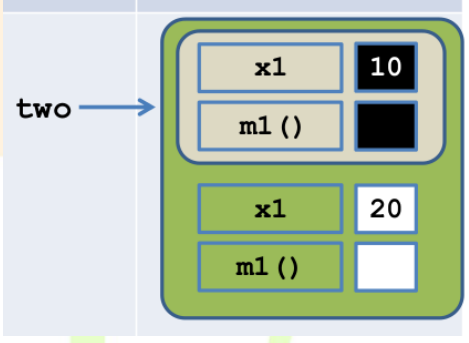
Overloading Methods

- When a method of a sub-class has the same name and type as a method of the super-class, we say that this method is overridden.
- Overriding method has the same name, number and type of parameters, and return type as the method it overrides.

Overriding

```
class  SuperClass{
int x1 = 10;
public int m1(){
return x1 * x1;
}
}
class SubClass extends SuperClass{
int x1 = 20;
public int m1(){
return x1;
}
}
class MainClass{
pubic static void main(String args[]){
SubClass two = new SubClass();
System.out.println(two.x1);
System.out.println(two.m1());
}
}
```

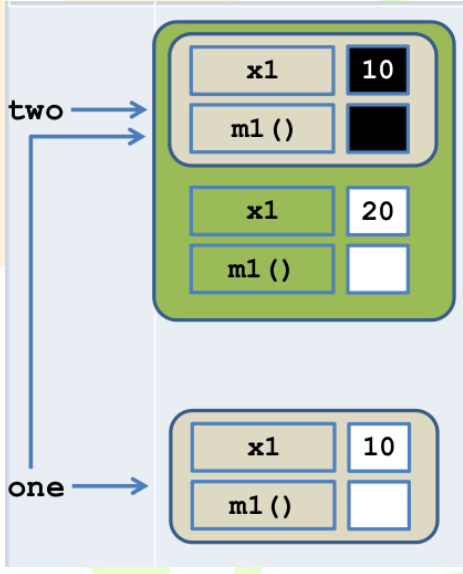
Output:
20
20



Overriding

```
class  SuperClass{
int x1 = 10;
public int m1(){
return x1;
}
}
class SubClass extends SuperClass{
int x1=20;
public int m1(){
return x1;
}
}
class MainClass{
pubic static void main(String args[]){
SuperClass one = new SuperClass();
SubClass two = new SubClass();
System.out.println(one.m1());
System.out.println(two.m1());

one = two;

System.out.println(one.m1());}
}
```

Output:
10
20
20



Introduction To Overloading

- The ability to allow different methods or constructors of a class to share the same name
- Always remember that overloaded methods have the following properties:
  - The same method name
  - type of parameters or number of parameters or order of parameters should be different.
  - Return types can be different or same

Introduction To Overloading

- A method can be overloaded in the same class or in a subclass.
- Access modifier can be different.

Overloading A Method Name

Same Method Name Means (does) different things in different circumstances

Method Overloading

```
class Example {
// same method name but 3 different methods
int area(double radius){
return 3.14 * radius * radius;
}
int area(int length, int breadth ) {
return length * breadth;
 }
int area (int side){
side * side;
}
}
```

Polymorphism