CPE207 Object Oriented Programming

Week 10

OOP Concepts: Polymorphism

Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

These Slides mainly adopted from Assist. Prof. Dr. Ozacar Kasim lecture notes

Table of Contents

Some Exercises: *OUTPUT???*

```
class A {
  public void method1() { System.out.println("A1"); }
  public void method3() { System.out.println("A3"); }
}

class B extends A {
  public void method2() { System.out.println("B2"); }
  public void method3() { System.out.println("B3"); }
}

Class C {
     public static void main(String[] args){
          A a = new A();
          B b = new B();

          a.method1();  //
          a.method2();  //
          a.method3();  //

          b.method1();//
          b.method2();//
          b.method3();//
     }
}
```

Polymorphism?

- **Polymorphism** (having multiple forms: **poly** + **morph**") is the characteristic of being able to assign a different meaning or usage to something.
- In OOP, it allows a variable, a method, or an object **to have more than one form**

Different Types of Polymorphism

- Java supports 2 types of polymorphism:
  1. **Compile-time (Static)**
     - Method overloading
  2. **Run-time (Dynamic)**
     - Method overriding

1-Compile-time Polymorphism

Overloading is compile-time polymorphism where more than one methods share the same name with different parameters or signature and different return type.

Example

```
class SimpleCalculator
{
    int add(int a, int b)
    {
        return a+b;
    }
    int  add(int a, int b, int c)
    {
        return a+b+c;
    }
}
public class Demo
{
   public static void main(String args[])
   {
          SimpleCalculator obj = new SimpleCalculator();
       System.out.println(obj.add(10, 20));
       System.out.println(obj.add(10, 20, 30));
   }
}
```

2-Run-time Polymorphism

Example

Method Overriding is run time polymorphism having same method with same parameters or signature but associated in a class & its subclass.

superclass

```
public class Animal {

    protected String color;

    void eat(){
        System.out.println("animal eating...");
    }
}
```

```
public class Cat extends Animal {
    int age;

    @Override
    void eat() {
        System.out.println("cat is eating");
    }
    void meow(){}
}
```

subclass

Polymorphism and arrays

- Example: The problem we have is, we have 3 different classes (Cat, Dog, Horse) all with the same behavior: eat().
- **An array can be only one type, how can we put them all in an array?**
  - TypeOfArray[] myArray = new TypeOfArray[]{cat, dog, horse};
- **Do we need 3 types of arrays?**
- That's the problem we can solve using polymorphism.
- Using Polymorphism, we can deal with different sub classes as the same super class
  - Animal[] myArray = new Animal[]{cat, dog, horse};

Polymorphism and arrays: Simple example

- Let's create Dog, Horse, and Cat classes. And using an array, to call eat() foreach animal.

```
public static void main(String[] args) {

    Dog dog= new Dog();
    Cat cat = new Cat();
    Horse horse = new Horse();
    Animal[] animals = new Animal[]{dog, cat, horse};
    //dog, cat, and horse are animals, so we can put them in Animal list
    for(Animal animal: animals)
        animal.eat();
}
```

Object Type Casting: Upcasting & downcasting

converting one type to another type is known as **type casting**

- **Primitive type casting**

```
double myDouble = 1.1;
int myInt = (int) myDouble;
//we're "turning" one type into another.
```

- **Reference type casting**
  - Casting from a subclass to a superclass is called **upcasting**
  - Upcasting is closely related to inheritance
    - `Cat cat = new Cat();`
    - `Animal animal = cat;`
    - `animal = (Animal) cat;`
  - Casting from a superclass to a subclass is called **downcasting**
    - `Animal animal = new Cat()`
    - `((Cat) animal).meow();`

Upcasting narrows the list of methods and properties available to this object, and downcasting can extend it.

Some exercises: code that won't compile!!!

- Not every Animal "is-a" Cat

  `Cat cat = new Animal(); //wont compile`
  - cannot call a Cat method on a variable of type Animal

- `Animal a = new Cat();`
  - `a.eat(); //will compile`
  - `a.meow(); //wont compile,` object `a` can only use `Animal` behaviors.

Which method gets called?

```
Animal c = new Cat();
```


```
c.eat();
```

- Will it call the **eat** method in Cat Class?

or

- Will it call the **eat** method in Animal Class?

A problem

```java
public class Animal {

    protected String color;

    void eat(){
        System.out.println("animal eating...");
    }
}
```

dummy implementations

```java
public class Cat extends Animal {
    int age;

    @Override
    void eat() {
        System.out.println("cat is eating");
    }
    void meow(){}
}
```

subclass

```java
public class Dog extends Animal{
    String breed;

    @Override
    public void eat(){
        System.out.println("Dog is eating");
    }

    void bark(){}
}
```

subclass

What if we don't want to implement a method in a super class?

Abstract Classes

Using abstract class

- An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes **cannot be instantiated**, but **they can be subclassed.**
- An abstract class can be used as a type of template for other classes. The abstract class will hold common functionality for all classes that extend it.
- Without abstract classes, you would have to provide dummy implementations of the methods you intend to override.

```java
public class Animal {

    protected String color;

    void eat(){
        System.out.println("animal eating...");
    }
}
```

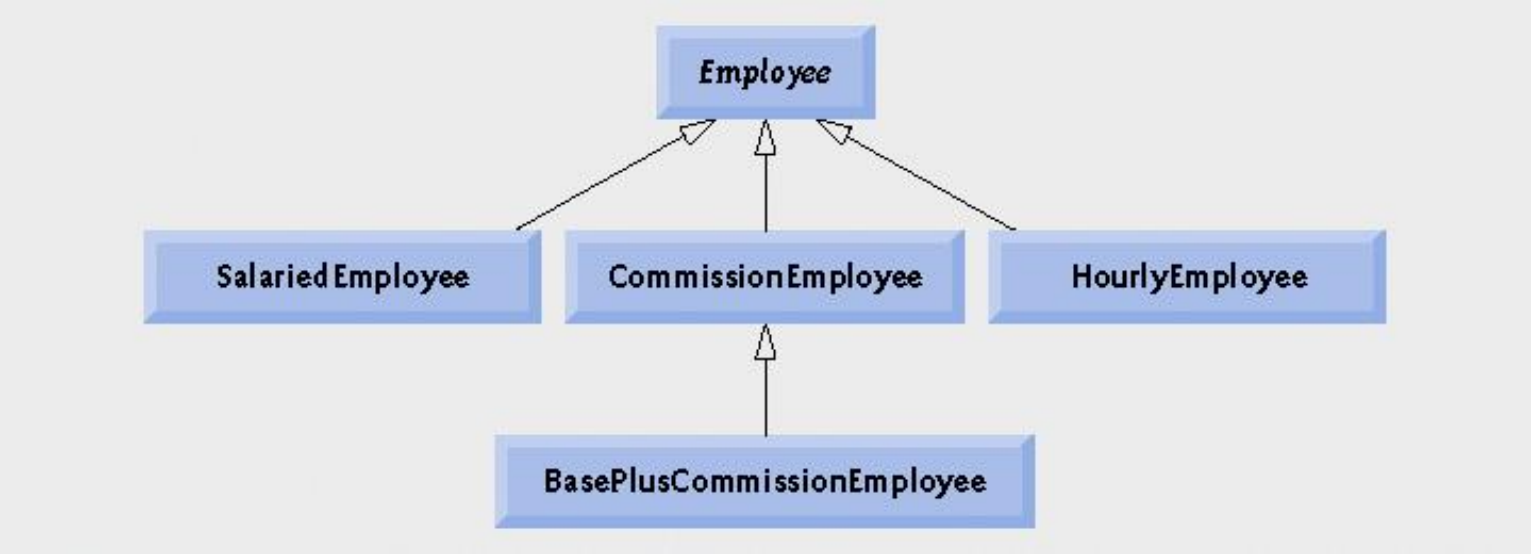
dummy implementations

```java
public abstract class Animal {
    protected String color;
    abstract void eat();
}
```

no body, no implementations

ANOTHER EXAMPLE: EMPLOYEE CLASSES

Employee hierarchy UML class diagram.



Polymorphic interface for the Employee hierarchy classes

	earning()	toString()
Employee	abstract	*firstName lastName* social security number: *SSN*
Salaried-Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *SSN* weekly salary: *weeklysalary*
Hourly-Employee	*If hours <= 40* wage * hours *If hours > 40* 40 * wage + (hours - 40) * wage * 1.5	hourly employee: *firstName lastName* social security number: *SSN* hourly wage: *wage*; hours worked: *hours*
Commission-Employee	commissionRate * grossSales	commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*
BasePlus-Commission-Employee	(commissionRate * grossSales) + baseSalary	base salaried commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*; base salary: *baseSalary*

Creating Abstract Superclass `Employee`

- abstract superclass Employee
  - earning is declared abstract
    - No implementation can be given for earnings in the Employee abstract class
  - An array of Employee variables will store references to subclass objects
    - earning method calls from these variables will call the appropriate version of the earnings method

Employee.java

```java
package oopweek9nightpoly2;

public abstract class Employee {
    String name_surname;
    int ssn;
    public Employee(String n, int ssn){
        setName_surname(n);
        setSsn(ssn);
    }
    public void setName_surname(String name_surname) {
        this.name_surname = name_surname;
    }

    public void setSsn(int ssn) {
        this.ssn = ssn;
    }

    abstract double earning();

    @Override
    public String toString() {
        return "name: "+ this.name_surname +" ssn: " + this.ssn;
    }
}
```

- You cannot create an instance from Employee, because it is an abstract class
- But you can create an instance from its subclasses

HourlyEmployee.java

- HourlyEmployee class extends Employee.

```java
package oopweek9nightpoly2;

public class HourlyEmployee extends Employee {
    double wage;
    int hours;

    public HourlyEmployee(String n, int ssn, double w, int h) {
        super(n, ssn);
        this.wage =w;
        this.hours =h;
    }

    @Override
    double earning() {
        if (hours<=40)
            return wage * hours;
        else
            return wage * hours + (hours - 40) * wage *1.5;
    }

    @Override
    public String toString() {
        return super.toString() + " wage: "+this.wage + " hours "+ this.hours;
    }
```

SalariedEmployee.java

- SalariedEmployee class extends Employee.

```
package oopweek9nightpoly2;

public class SalariedEmployee extends Employee {

    double weeklySalary;

    public SalariedEmployee(String n, int ssn, double wSalary){
    super(n, ssn);
        this.weeklySalary =wSalary;
    }

    @Override
    double earning() {
        return weeklySalary;
    }
}
```

CommissionEmployee.java

- CommissionEmployee class extends Employee.

```java
package oopweek9nightpoly2;

public class CommisionEmployee  extends Employee{
    double grossSale;
    double commissionRate;

    public CommisionEmployee(String n, int ssn, double gSale, double cRate) {
        super(n, ssn);
        this.commissionRate = cRate;
        this.grossSale = gSale;
    }

    @Override
    double earning() {
        return grossSale * commissionRate;
    }

    @Override
    public String toString() {
        return super.toString() + " gSale:" + this.grossSale +" cRate: "+this.commissionRate;
    }
```

BasePlusCommissionEmployee.java

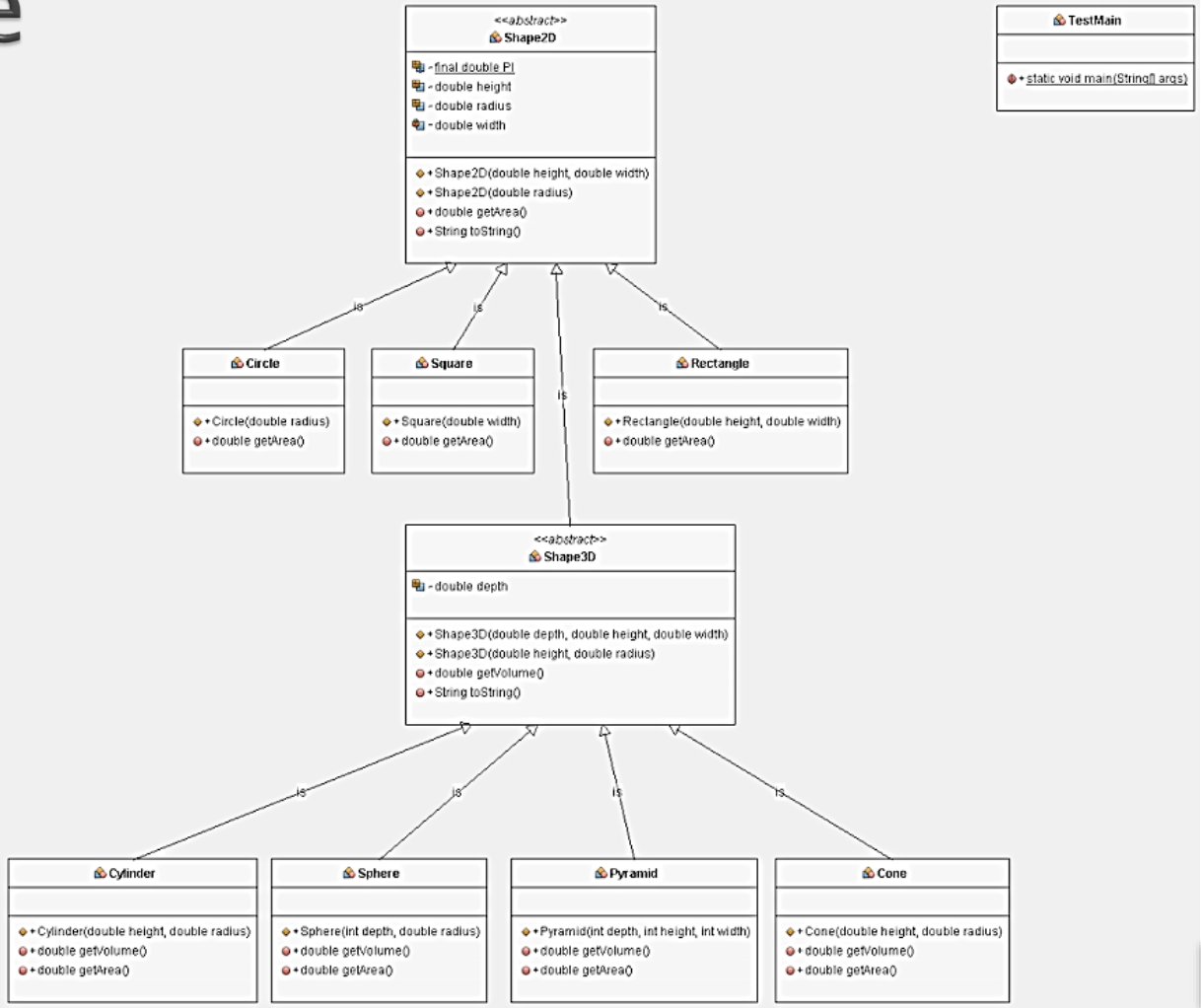
- BasePlusCommissionEmployee class extends CommissionEmployee.

```
package oopweek9nightpoly2;

public class BasePlusCommisionEmployee extends CommisionEmployee{
   double baseSalary;

    public BasePlusCommisionEmployee(String n, int ssn, double gSale, double cRate, double bSalary) {
        super(n, ssn, gSale, cRate);
        this.baseSalary = bSalary;
    }

    @Override
    double earning() {
        return super.earning() + this.baseSalary;
    }
}
```

MainClassTest.java

```java
public class  MainClassTest  {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Employee e1 = new HourlyEmployee("jack sparrow", 789745, 30, 35);
        Employee e2 = new SalariedEmployee("jane", 789979, 1800);
        Employee e3 = new CommisionEmployee("tom jerry", 212334, 100000, 0.1);
        Employee e4 = new BasePlusCommisionEmployee("john Zuckerberg", 212314, 1000000, 0.1, 5000);

        Employee[] employees = new Employee[]{e1,e2,e3,e4};

        for(Employee e : employees)
            System.out.println(e + " salary " + e.earning());

    }

}
```

LAB Exercise



Thanks ☺