Inheritance

Part 3 - Abstract Classes

Chapter 5, Core Java Volume I and

Chapter 10, Java How to Program, 10th ed.

Contents

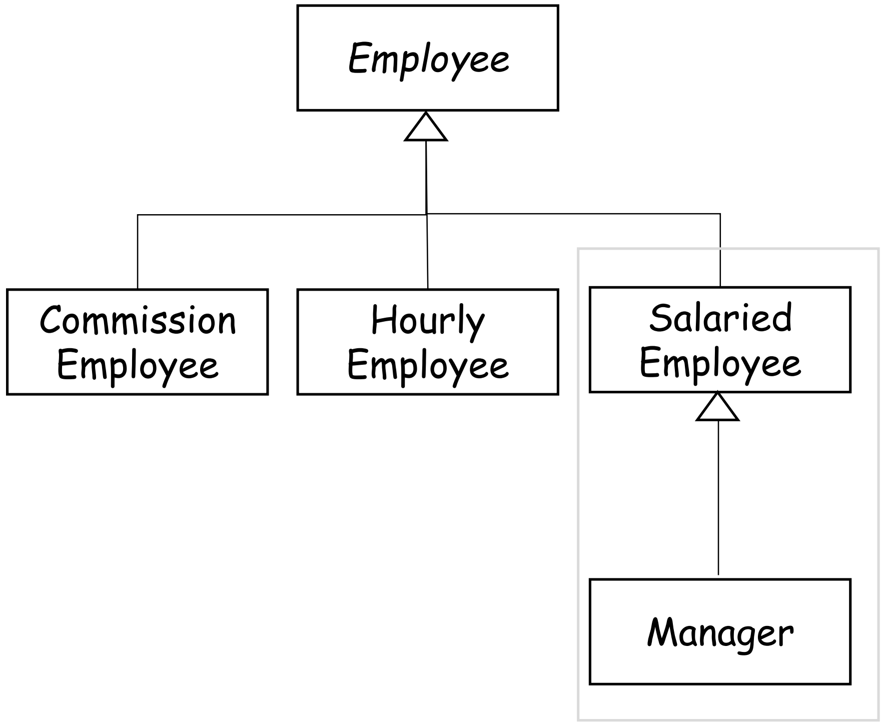
Abstract Classes

- When factoring out common classes, it can become difficult to implement methods in the most general classes.

- Example: Employee classes(*See* Deitel 2015)
  - Commission Employee
  - Hourly Employee
  - Salaried Employee
  - Manager
- Each subclass defines a `getEarnings()` method:

  - earning = commission
  - earning = hours worked per week * hourly wage
  - earning = weekly salary
  - earning = weekly salary + bonus

- What is the earning of an `Employee`?



(Deitel 2015) P. Deitel and H. Deitel, Java: How to Program, 10th Ed., Pearson, 2015, (Chapter 10).

Abstract Classes

- Abstract methods are defined without implementation and must be declared abstract:

```
// in class Employee
 public abstract double getEarnings();  // no implementation
```

- An abstrct class is a class that is declared *abstract*. (cf. concrete class)

```
public abstract class Employee { … }
```

- A class that has an abstract method must be declared abstract.
- Ok for abstract classes to have fields, constructors, and *concrete* methods:

```
public abstract class Employee
{
   private String name;
   private String ssn;
   public Employee(String n, String ssn) { name = n; this.ssn = ssn; }
   public String getName() { return name; } // concrete method
   public abstract double getEarnings();      // abstract method
   ...
}
```

Abstract Classes

- Abstract classes cannot be instantiated:

```
Employee e = new Employee("Vince Vu"); // Error!
```

- An abstract class can be used to declare variables to refer to objects of its subclasses:

```
Employee e = new Manager("Vince Vu", "000-11-1111", 2500.0); // Ok
```

- A class can be declared abstract even if it has no abstract methods.
  - with a concrete appearance, but cannot be instantiated

- If some or all methods of an abstract class are undefined in its subclass, the subclass must be declared abstract.

Abstract Classes

- What is the purpose of abstract classes?
  - Used as a base class that can be inherited by multiple subclasses (is-relationship).
  - Abstract methods can be overrided in the subclasses.
  - Thus, they provide polymorphism.

```
Employee[] emp = new Employee[4];
emp[0] = new CommissionEmployee(…);
emp[1] = new HourlyEmployee(…);
emp[2] = new SalariedEmployee(…);
emp[3] = new Manager(…);

for (Employee e : emp)
    System.out.println(e.getName()+" : " + e.getEarnings());   // polymorphism
```

Example : Employee Hierarchy

```
// Employee.java
public abstract class Employee
{
   private final String name;
   private final String ssn;

   // constructor
   public Employee(String name,String ssn)
   {
      this.name = name;
      this.ssn = ssn;
   }
  // return name
   public String getName()
   {
      return name;
   }
```

```
// return social security number
   public String getSsn()
   {
      return ssn;
   }

 //abstract method
   public abstract double getEarnings();

} // end abstract class Employee
```

Example : Employee Hierarchy

```java
// CommissionEmployee.java
public class CommissionEmployee extends Employee
{
   private double grossSales; // gross weekly sales
   private double commissionRate; // commission percentage
   // constructor
   public CommissionEmployee(String name, String ssn,
            double grossSales,  double commissionRate)
   {
      super(name, ssn);
      if (commissionRate <= 0.0 || commissionRate >= 1.0)
         throw new IllegalArgumentException(
            "Commission rate must be > 0.0 and < 1.0");

      if (grossSales < 0.0) // validate
         throw new IllegalArgumentException(
             "Gross sales must be >= 0.0");

      this.grossSales = grossSales;
      this.commissionRate = commissionRate;
   }
```

```java
 // get/set methods here …


// override abstract method earnings in Employee
   @Override
   public double getEarnings()
   {
      return commissionRate * grossSales ;
   }
} // end of CommissionEmployee class
```

Example : Employee Hierarchy

```
// HourlyEmployee.java
public class HourlyEmployee extends Employee
{
  private double wage; // wage per hour
  private double hours; // hours worked for week
  // constructor
  public HourlyEmployee(String name,
    String ssn, double wage, double hours)
  {
    super(name, ssn);
    if (wage < 0.0) // validate wage
      throw new IllegalArgumentException(
        "Hourly wage must be >= 0.0");

    if ((hours < 0.0) || (hours > 68.0)) // validate hours
      throw new IllegalArgumentException(
        "Hours worked must be >= 0.0 and <= 68.0");
    this.wage = wage;
    this.hours = hours;
  }
```

```
 // get/set methods here …


  // override abstract method earnings in Employee
  @Override
  public double getEarnings()
  {
    if (hours <= 40) // no overtime
      return wage * hours;
    else
      return 40 * wage + (hours - 40) * wage * 1.5;
  }
} // end class HourlyEmployee
```

Example : Employee Hierarchy

```
// SalariedEmployee.java
public class SalariedEmployee extends Employee
{
   private double weeklySalary;

   // constructor
   public SalariedEmployee(String name, String ssn,
             double weeklySalary)
   {
      super(name, ssn);

      if (weeklySalary < 0.0)
         throw new IllegalArgumentException(
            "Weekly salary must be >= 0.0");

      this.weeklySalary = weeklySalary;
   }
```

```
// get/set methods


   public void raiseSalary(double rate)
  {
      weeklySalary += (weeklySalary * rate/100);
   }

   // override abstract method earnings in Employee
   @Override
   public double getEarnings()
   {
      return weeklySalary;
   }

 } // end class SalariedEmployee
```

Example : Employee Hierarchy

```
// Manager.java

public class Manager extends SalariedEmployee
{
   private double bonus; // weekly bonus

   // constructor
   public Manager(String name, String ssn, double
weeklySalary)
   {
      super(name, ssn, weeklySalary);
      bonus = 0.0;
   }
```

```
public void setBonus(double bonus)
   {
      this.bonus = bonus;
   }

   //override method earnings in CommissionEmployee
   @Override
   public double getEarnings()
   {
      return super.getEarnings() + bonus;
   }

} // end class BasePlusCommissionEmployee
```

Example : Employee Hierarchy

```
public class PayrollSystemTest
{
  public static void main(String[] args)
  {
    // create subclass objects
    SalariedEmployee salariedEmployee =
      new SalariedEmployee("John", "111-11-1111", 800.00);
    HourlyEmployee hourlyEmployee =
      new HourlyEmployee("Karen", "222-22-2222", 16.75,
40);
    CommissionEmployee commissionEmployee =
      new CommissionEmployee( "Sue",  "333-33-3333",
10000,  .06);
    Manager manager =
      new Manager("Bob", "444-44-4444", 2500.0);
```

```
    System.out.println("Employees processed individually:");

    System.out.printf("%n%s%s: $%,.2f%n%n",
      salariedEmployee.getName(), " earned",
salariedEmployee.getEarnings());
    System.out.printf("%s%s: $%,.2f%n%n",
      hourlyEmployee.getName(), " earned",
hourlyEmployee.getEarnings());
    System.out.printf("%s%s: $%,.2f%n%n",
      commissionEmployee.getName(), " earned",
commissionEmployee.getEarnings());
    System.out.printf("%s%s: $%,.2f%n%n",
      manager.getName(),   " earned",
manager.getEarnings());
```

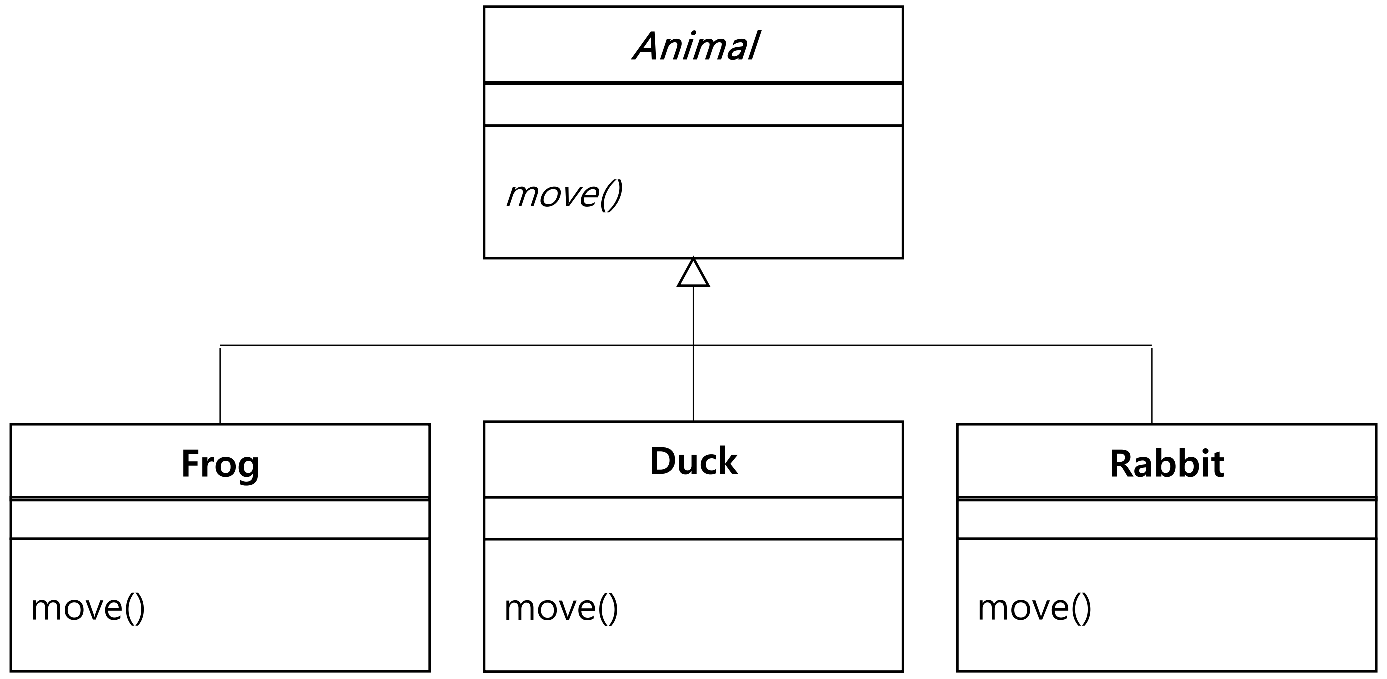
Example : Employee Hierarchy

```
// create four-element Employee array
    Employee[] employees = new Employee[4];

    // initialize array with Employees
    employees[0] = salariedEmployee;
    employees[1] = hourlyEmployee;
    employees[2] = commissionEmployee;
    employees[3] = manager;

    System.out.printf("Employees processed
polymorphically:%n%n");
```

```
// generically process each element in array employees
    for (Employee currentEmployee : employees)
    {
      System.out.println(currentEmployee.getName());
     if (currentEmployee instanceof Manager)
      {
        // downcast Employee reference to Manager
reference
        Manager employee =  (Manager) currentEmployee;
        employee.setBonus(100.0);
      }
      System.out.printf("%s earned $%,.2f%n%n",
          currentEmployee.getName(),
          currentEmployee.getEarnings());
    }
  } // end main
} // end class PayrollSystemTest
```

More Examples

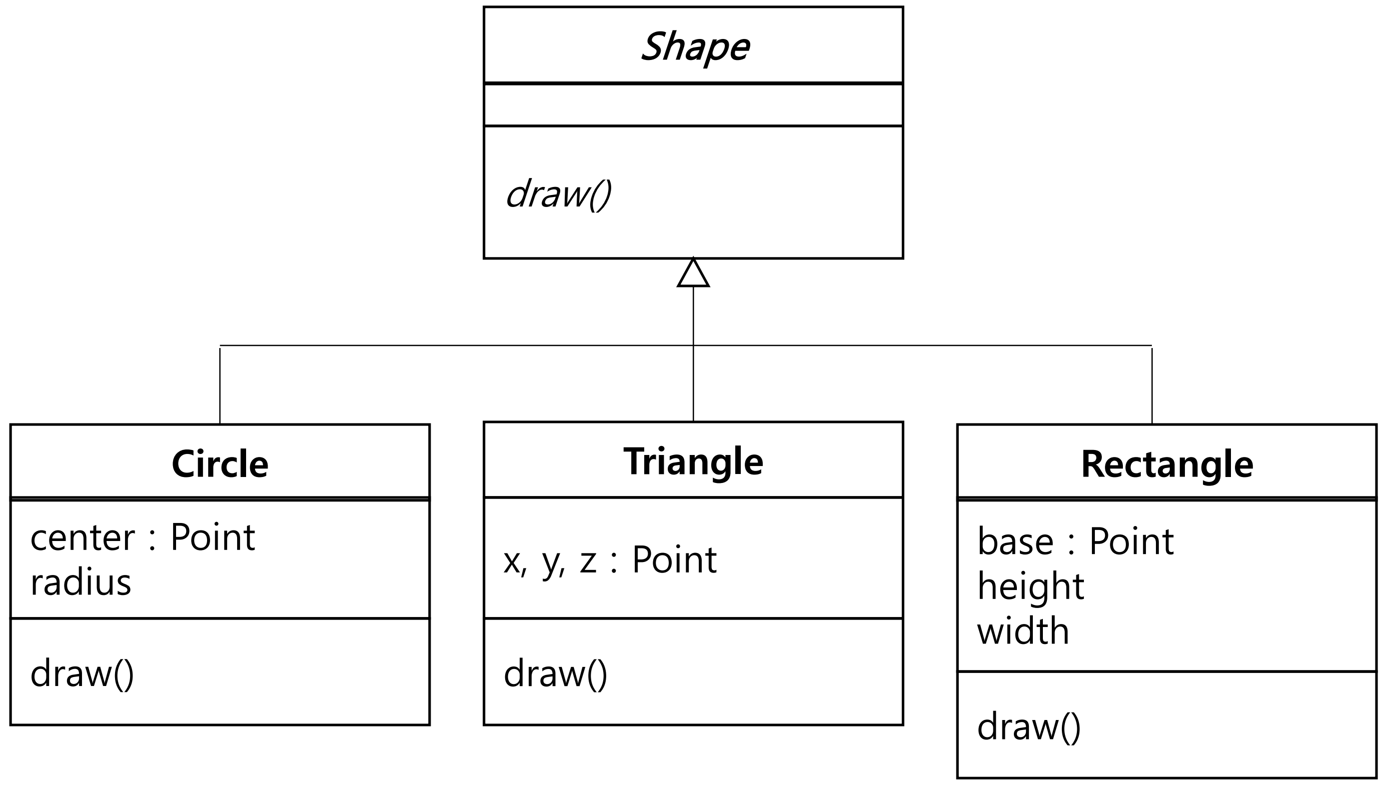
- Animal Hierarchy



```
class Zoo
{
  Animal[] all;
  public Zoo() {
     all = new Animal[3] ;
     all[0] = new Frog(…);
     all[1] = new Duck(…);
     all[2] = new Rabbit(…);
  }
  void moveAll()
  {
     for(Animal a : all)
       a.move();
  }
  public static void main(…)
  {
      Zoo zoo = new Zoo();
      zoo.moveAll();
  }
}
```

More Examples

- Shape Hierarchy



```
class Board
{
  Shape[] all;
  public Board() {
     all = new Shape[3] ;
     all[0] = new Circle(…);
     all[1] = new Triangle(…);
     all[2] = new Rectangle(…);
  }
  void drawAll()
  {
     for(Shape a : all)
       a.draw();
  }
  public static void main(…)
  {
      Board board = new Board();
      board.drawAll();
  }
}
```