Chapter 10 Inheritance and Polymorphism

Motivations

- What features are common for all the shapes?
- What features are specific to:
  - Triangle!
  - Circle?
  - Rectangle

Superclasses and Subclasses

```java
public class GeometricObject {
    public boolean isFilled;
}
```

 inherit the method from the superclass

```java
public class Circle extends GeometricObject {
    public double radius;
}
```

Override the methods of the superclass

- sideLength
- radius
- width
- height

Inheritance - idea

```
public class GeometricObject {
    public Color color;
    public boolean isFilled;
}
```

```
public class Circle extends GeometricObject {
    public double radius;
}
```

```
public boolean isFilled;
```

Inheritance

- A subclass inherits all fields and methods from the superclass
- A subclass can also:
  - Add new fields
  - Add new methods
  - Override the methods of the superclass
- Superclass’s constructor are not inherited
  - Unlike fields and methods
  - They are invoked explicitly or implicitly

extends keyword

- Use extends keyword to tell that one class inherits from another class

```
public class GeometricObject {
    public boolean isFilled;
}
```

```
public class Circle extends GeometricObject {
    public double radius;
}
```

- What are properties of instances of class Circle?

Inheritance - idea

```
public class GeometricObject {
    public Color color;
    public boolean isFilled;
}
```

```
public class Circle extends GeometricObject {
    public double radius;
}
```

```
public boolean isFilled;
```

Inheritance

- A subclass inherits all fields and methods from the superclass
- A subclass can also:
  - Add new fields
  - Add new methods
  - Override the methods of the superclass
- Superclass’s constructor are not inherited
  - Unlike fields and methods
  - They are invoked explicitly or implicitly
Using the Keyword super

- super refers to the superclass of the class in which super appears
- This keyword can be used in few ways:
  - To call a superclass constructor
  - To call a superclass method
  - To access a superclass public data field

Invoking Superclass Constructor

- Superclasses’ constructors can (only) be invoked from subclasses’ constructors explicitly
- Use the keyword super to call the superclass constructor
- Java requires that the statement that invokes superclass’ constructor using the keyword super appear first in the constructor

Superclass’s Constructor Is Always Invoked

- If no superclass constructor is explicitly invoked, the compiler puts super() as the first statement in the constructor

Constructor Chaining

- public class Faculty extends Employee {
  - Faculty f = new Faculty("Brad Pitt");
  - public Faculty(String name) {
    - System.out.println("(3) Faculty's constructor");
  - }
  - class Employee extends Person {
    - System.out.println("(2) Employee's no-arg constructor");
  - }
  - class Person {
    - System.out.println("(1) Person's no-arg constructor");
  - }

Calling Superclass Methods

- super can be used to call method from superclass
  - public void printCircle() {
    - System.out.println("The circle is created " + super.getDateCreated() + " and the radius is " + radius);
  - }
- super can be omitted sometimes… when!
  - public void printCircle() {
    - System.out.println("The circle is created " + getDateCreated() + " and the radius is " + radius);
  - }
Overriding Methods in the Superclass
- A subclass inherits methods from a superclass
- Subclass can modify the implementation of a method defined in the superclass.
- Method overriding:

```java
public class Circle extends GeometricObject {
    // Other methods are omitted

    /** Override the toString method defined in GeometricObject */
    public String toString() {
        return super.toString() + " radius is " + radius;
    }
}
```

NOTE
- An instance method can be overridden only if it is accessible
- A private method cannot be overridden, because it is not accessible outside its own class

Overriding vs. Overloading
- `p.13`

Another Example: Bank Accounts
- `p.13`

Inheritance
- SavingsAccount and CheckingAccount can extend BankAccount class and automatically inherit all methods and instance fields of BankAccount.
- Extended class = superclass (BankAccount), extending class = subclass (SavingsAccount, CheckingsAccount)

Hierarchy of Bank Accounts
- Methods
  - All bank accounts support the `getBalance` method
  - All bank accounts support the `deposit` and `withdraw` methods, but the implementations differ.
  - Checking account needs a method `deductFees`; Savings account needs a method `addInterest`

  - Instance fields
    - SavingsAccount class needs an `interestRate` instance field
    - Which instance field will we need to add to the CheckingAccount class?
Implementing the CheckingAccount Class

- Checking account: small number of free transactions per month, additional transactions are charged a small fee
- Instance fields:
  - balance (inherited from BankAccount)
  - transactionCount (new to CheckingAccount)
- Methods:
  - getBalance() (inherited from BankAccount)
  - deposit(double amount) (overrides BankAccount method) – need to update the transaction count
  - withdraw(double amount) (overrides BankAccount method) – need to update the transaction count
  - deductFees() (new to CheckingAccount)

Implementing deposit() method

- Can’t just add amount to balance, balance is a private field of the superclass
- Can’t just call deposit(amount) in deposit method of CheckingAccount – the same as this.deposit(amount) -- infinite recursion
- Invoke superclass method super.deposit(amount)

```
public void deposit(double amount)
{
    transactionCount++;
    // Now add amount to balance
    super.deposit(amount);
}
```

Implementing the SavingsAccount Class

- Savings account: earns interest with a interest rate
- Instance fields:
  - balance (inherited from BankAccount)
  - New instance field?
- Methods:
  - getBalance()
  - deposit(double amount)
  - withdraw(double amount)
  - addInterest() (new to SavingsAccount)

Review questions

- Which of the following statements are true?
  A. A subclass is a subset of a superclass.
  B. A subclass is usually extended to contain more functions and more detailed information than its superclass.
  C. “class A extends B” means A is a subclass of B.
  D. “class A extends B” means B is a subclass of A.

Review questions

- Which of the following statements are true?
  A. A method can be overloaded in the same class.
  B. A method can be overridden in the same class.
  C. If a method overloads another method, these two methods must have the same signature.
  D. If a method overrides another method, these two methods must have the same signature.

Object: The Cosmic Superclass

- All classes defined without an explicit extends clause automatically extend Object
Object: The Cosmic Superclass

- Most useful methods:
  - String toString()
  - boolean equals(Object otherObject)
- Good idea to override these methods in your classes

The toString() method in Object

- The toString() method returns a string representation of
  the object
- The method is called whenever the object is converted to a
  string
  System.out.println("Bankaccount: " + myaccount);
- The default implementation returns a string consisting of
  a class name of which the object is an instance, the at sign
  (@), and a number representing this object.
  BankAccount account = new BankAccount();
  System.out.println(account); //BankAccount@15037e5

Overriding the toString() Method

- To provide a nicer representation of an object, override toString().
  public String toString()
  {
    return "BankAccount\{balance=\" + balance + \"\}";
  }
- This works better:
  BankAccount momsSavings = new BankAccount(5000);
  System.out.println(momsSavings); //BankAccount\{balance=5000\}
- More examples: Card.java

The equals Method

- The equals() method compares two objects.
- The default implementation of the equals method in the
  Object class is as follows:
  public boolean equals(Object obj) {
    return (this == obj);
  }
- Java classes such as String override equals() method so
  that it compares the content of two objects.
- It is a good idea to override equals() method for your
  own classes

Overriding equals method

```java
public class Circle {
    private double radius;
    public Circle(double r) {
        radius = r;
    }
    public boolean equals(Object o) {
        Circle other = (Circle)o;
        return other.radius == radius;
    }
}
```

Review questions

- What is the output of the following code:
  public class Test {
      public static void main(String[] args) {
          Object o1 = new Object();
          Object o2 = new Object();
          System.out.print((o1 == o2) + " " + (o1.equals(o2)));
      }
  }
  A. false false
  B. true true
  C. false true
  D. true false
Review questions

What is the output of the following code:

```java
public class Test {
    public static void main(String[] args) {
        String s1 = "Java";
        String s2 = "Java0".substring(0, 4);
        System.out.println((s1 == s2) + " " + (s1.equals(s2)));
    }
}
```

A. false false
B. true true
C. false true
D. true false

Converting Between Subclass and Superclass Types

- Ok to convert subclass reference to superclass reference
- Need cast to convert from a superclass reference to a subclass reference
- This cast is dangerous: if you are wrong, an exception is thrown

TIP

- To help understand casting, consider the analogy of fruit, apple, and orange.
- Fruit class is the superclass for Apple and Orange.
- An apple is a fruit, so you can always safely assign an instance of Apple to a variable of Fruit.
- However, a fruit is not necessarily an apple, so you have to use explicit casting to assign an instance of Fruit to a variable of Apple.

Casting

- Casting from a superclass to a subclass is dangerous: if you are wrong, an exception is thrown
- Use the `instanceof` operator to test whether an object belongs to a particular type before casting

```
object instanceof TypeName
```

Example:

```java
if (anObject instanceof BankAccount) {
    BankAccount anAccount = (BankAccount) anObject;
    ...
}
```

Purpose:
To return true if the object is an instance of `TypeName` (or one of its subtypes) and false otherwise.

Converting Between Subclass and Superclass: Examples

```
SavingsAccount momsSavings = new SavingsAccount(0.5);
CheckingAccount harrysChecking = new CheckingAccount();
BankAccount a = new BankAccount();
// what happens to the following?
BankAccount b = harrysChecking;
CheckingAccount c = (CheckingAccount) a;
CheckingAccount d = b;
CheckingAccount e = (CheckingAccount) b;
SavingsAccount s = (SavingsAccount) b;
```

Polymorphism and Dynamic Binding

- Method calls are determined by type of actual object, not type of object reference
- Suppose an object o is an instance of classes C1, C2, ..., Cn-1, and Cn, where C1 is a subclass of C2, C2 is a subclass of C3, ..., and Cn-1 is a subclass of Cn
- In Java, Cn is the Object class. If o invokes a method p, the JVM searches the implementation for the method p in C1, C2, ..., Cn-1 and Cn, in this order, until it is found.

Since o is an instance of Cn, o is also an instance of C1, C2, ..., Cn-1, and Cn.
Polymorphism
- PolymorphismDemo.java
  - Which toString() method to call is determined by the object
  - What's the output?
- BankAccountTester.java (Bank account example revisited)
  - In the transfer method, depending on types of other, different versions of withdraw and deposit are called
  - What's the output?

Review questions
- Given the following code:
  ```java
class C1 {}
class C2 extends C1 {}
class C3 extends C2 {}
class C4 extends C1 {}
C1 c1 = new C1();
C2 c2 = new C2();
C3 c3 = new C3();
C4 c4 = new C4();
```
  Which of the following expressions evaluates to false?
  A. c1 instanceof C1
  B. c2 instanceof C1
  C. c3 instanceof C1
  D. c4 instanceof C2

The Array List
- ArrayList class that can be used to store an unlimited number of objects.
- TestArrayList.java
  ```java
  public class TestArrayList {
    private ArrayList<String> list;
    private ArrayList<Integer> numbers;
    public TestArrayList() {
      list = new ArrayList<String>();
      numbers = new ArrayList<Integer>();
    }
    public void addString(String s) {
      list.add(s);
    }
    public String removeFirst() {
      return list.remove(0);
    }
    public int size() {
      return numbers.size();
    }
  }
  ```

The protected Modifier
- The protected modifier can be applied on data and methods in a class
- A protected data or a protected method in a public class can be accessed by any class in the same package or its subclasses, even if the subclasses are in a different package

Accessibility Summary

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Accessed on members in a class</th>
<th>Accessed from the same class</th>
<th>Accessed from a different package</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>protected</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
A Subclass Cannot Weaken the Accessibility

- A subclass may override a protected method in its superclass and change its visibility to public.
- However, a subclass cannot weaken the accessibility of a method defined in the superclass.
- For example, if a method is defined as public in the superclass, it must be defined as public in the subclass.

The final Modifier

- The final class cannot be extended:
  ```java
class Math {
    ...
  }
```

- The final variable is a constant:
  ```java
  final static double PI = 3.14159;
  ```

- The final method cannot be overridden by its subclasses:

Practice questions

- Inheritance
  - 10.1, 10.3, 10.6, 10.7, 10.9
- Object class
  - 10.12
- Polymorphism
  - 10.13
- ArrayList class
  - 10.17
- Modifiers
  - 10.20