Objectives of Lecture 24

Inheritance

- Introduce the notion of inheritance in object-oriented programming;
- Understand the concepts of superclass (base class) and subclass (derived class);
- Learn how to take advantage of similarities between objects from different classes to derive one class from another and inherit instance variables and methods.

Outline of Lecture 24

- Subclasses and Superclasses
- Type inheritance
- Method inheritance
- Representation inheritance
- Constructor inheritance
The Idea Behind Inheritance

- Extending the capabilities (i.e. behaviour and state) of a class C1 in order to generate a new class C2 with the same capabilities as C1 in addition to new capabilities.

Object of class C1
- instance variable_1
- instance variable_2
- ...
- instance variable_n
- Method_a
- Method_b

Object of class C2
- instance variable_1
- instance variable_2
- ...
- instance variable_n
- Method_a
- Method_b
- instance variable_n+1
- Method_c

Inheritance in the Real World

- How is a student like a person?
- Well, every student is a person!
- Students have all of the “properties” of persons, plus some others.
- For example, every person has a name and an age and so does every student.
- However, not every person is a student.
- Every student has a student id and a grade point average, that other persons don't have.

Two Different Approaches

- In Java, we model a person by a Person class.
- In Java, we model a student by a Student class.
- Introduce two independent classes, one for Student and one for Person
  - we lost relationships between the two
  - a Student class has to redefine all the properties of a Person class
- Define a Student class as a specialization of a Person class
  - characterize special relationships
  - software reusability
Subclasses and Superclasses

- Since a student is like a person with extra properties, we say the class Student is a subclass of the class Person (or derived class).
- We also say that Person is a superclass of Student (or base class).

The Java Inheritance Tree

- In general, Person can have other subclasses as well, say Teacher.
- We put all the classes in an inheritance tree with class Object as the root.
- We draw the tree with the root at the top.

Outline of Lecture 24

- Subclasses and Superclasses
- Type inheritance
- Method inheritance
- Representation inheritance
- Constructor inheritance

Type Inheritance

- We say that a subclass inherits all of the messages from its superclass.
- Any message that can be sent to an instance of a class can also be sent to an instance of its subclasses.
- However, you can add additional instance messages and static messages to a subclass.
Type Inheritance (con’t)

- If you declare the type of a variable to be some class, it can then be bound to an instance of that class or any subclass.
- If the type of a message parameter or the return type of a message is a class, you can use any subclass as well.
- The property of being able to use an instance of a subclass, wherever you can use an instance of a class is called **substitutability**.

Type Inheritance Example

- Assume that we are defining a class called Store.
- Assume that we have already defined a class called Person, with a message called name() and two subclasses: Student and Teacher.
- Assume that we have defined a message in this “Store” class called register that takes a Person as a parameter:

  ```java
  public void register(Person aPerson) {
  // Register the given Person as a customer.
  }
  ```

Type Inheritance Example (con’t)

- Here is a method that creates a Person, Student or Teacher customer, depending on a char parameter.

  ```java
  public Person createCustomer(char aChar, String aString) {
      Person customer;
      if (aChar == 'T') customer = new Teacher(nameString);
      else if (aChar == 'S') customer = new Student(nameString);
      else customer = new Person(nameString);
      System.out.println("Welcome " + customer.name());
      this.register(customer);
      return customer;
  }
  ```

Instance Variable and Static Variable (Representation) Inheritance

- In Java, a subclass also inherits all of the instance variables and all of the static variables of its superclass.
- However, if a variable is private, it cannot be accessed directly in the subclass code.
- If a variable is declared as **protected** it can be accessed directly in the subclass code.
- A subclass can also add state by defining additional instance and static variables.
Outline of Lecture 24

- Subclasses and Superclasses
- Type inheritance
- Method inheritance
- Representation inheritance
- Constructor inheritance

Method (Implementation) Inheritance

- In Java, a subclass also inherits the methods of its superclass, so they do not have to be re-implemented.
- However, you can also override any method if you want.
- In addition, you can add some code to an inherited method, using the `super` object reference.

Method Override

Object of class C1

<table>
<thead>
<tr>
<th>instance variable_1</th>
<th>instance variable_2</th>
<th>...</th>
<th>instance variable_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method_a</td>
<td>Method_b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Object of class C2

<table>
<thead>
<tr>
<th>instance variable_1</th>
<th>instance variable_2</th>
<th>...</th>
<th>instance variable_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method_a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>instance variable_n+1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method_a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Class C1

- Method_a

Class C2

- Method_a

Outline of Lecture 24

- Subclasses and Superclasses
- Type inheritance
- Method inheritance
- Representation inheritance
- Constructor inheritance
Representation (or Data) Inheritance

Object of class C1

instance variable_1
instance variable_2
... instance variable_n
Method_a
Method_b

Object of class C2

instance variable_1
instance variable_2
... instance variable_n
Method_a
Method_b
instance variable_n+1
Method_c

Representation/Implementation Inheritance - Example

public class Person {
    // Each instance represents a Person.
    ...
    // Public methods
    public void output() {
        // Output a representation of myself
        System.out.print("name: " + this.name + " age: ");
        System.out.println(this.age());
    }
    ...
    // Instance Variables
    protected String name;
    private Date birthdate;
    ...
    name is protected: it is accessed only by class Person and its subclasses.
}

Representation /Implementation Inheritance - Example (con’t)

public class Student extends Person {
    // Each instance represents a Student.
    // Public methods
    public void output() {
        // Output a representation of myself
        super.output();
        System.out.print(" id: ");
        System.out.println(this.id);
    }
    ...
    // Instance Variables
    // cannot access birthdate, but can access name because it is protected
    private int id;
    ...
    Calls the output() method of the superclass Person.

Outline of Lecture 24

- Subclasses and Superclasses
- Type inheritance
- Method inheritance
- Representation inheritance
- Constructor inheritance
Constructor Chaining

• Constructors are not inherited like other methods. We say constructors are chained.
• If you want to call another constructor in the same subclass, you just use “this()” with the appropriate arguments.
• If you want to call another constructor in the superclass, you just use “super()” with the appropriate arguments.

Constructor Chaining (con’t)

• However, each constructor must “ultimately” call one of the constructors in its superclass.
• This can be done in one of three ways:
  – An explicit call to super() with arguments.
  – A call to another constructor in the subclass using this() with arguments.
  – If neither of these appear as the first statement of the subclass constructor, the compiler inserts an implicit call to the zero argument super constructor super(). However, the a constructor with no arguments should exist in the superclass.

Constructors - Example

```java
public class Person {
    // Each instance represents a Person.
    // Constructors
    public Person() {// Set the name “unknown” and birtdate: today
        this.name = “unknown”;
        this.birthdate = new Date();
    }

    public Person(String nameString) {
        // Set the given name and birthdate: today
        this(); // do the 0 argument constructor first
        this.name = nameString;
    }
}
```

Constructors - Example (con’t)

```java
public class Student extends Person {
    // Each instance represents a Student.
    public Student() {// Set the name: “unknown”, birtdate: today, id: 0
        this.id = 0; // implicit call to super(); first
    }

    public Student(String nameString) {
        // Set the given name, birthdate: today, id: 0
        super(nameString); // explicit call
        this.id = 0;
    }

    public Student(String nameString, int anInt) {
        // Set the given name and id, birthdate: today
        this(nameString); // or super(nameString)
        this.id = anInt;
    }
}
```
Multiple Inheritance

- Multiple inheritance is the inheritance of properties from more than just one base class.
- Java does not allow multiple inheritance.
- Other Object-Oriented languages such as C++ allow multiple inheritance;

```
parent1
  ↘
  child

parent2
  ↘
  child
```

Simple inheritance
Multiple inheritance