CS 171: Introduction to Computer Science II

OO, Inheritance, and Libraries

Li Xiong
Announcements/Reminders

• Introductory/Eclipse help session
  – Session 1: Wednesday, 1/23, 1:20-2:20pm
  – Session 2: Friday, 1/25, 4-5pm

• Hw1
  – Assigned Tuesday 1/22, due Thursday 1/31
Roadmap

• Java Review/Basics
  – Types, variables, assignments, expressions
  – Control flow statements
  – Methods

• Arrays
  – Array basics
  – Binary Search Algorithm
  – Binary Search program
  – Hw1: number guessing game
  – ArrayList

• OO and Inheritance
Objects and Classes

• Object: entity that you can manipulate in your programs
  – Data fields: state of an object
  – Methods: instructions that accesses or modifies the object

• Class: construct that defines objects of the same type (set of objects with the same behaviour)
  – Definition of data fields: properties of defined objects
  – Definition of methods: behaviours of defined objects
    • Constructors are special type of methods used to construct and initialize objects from the class
Modular Programming

- Software is composed of separate, interchangeable components (modules)
- Library of classes that are intended for use by others
- Application programming interfaces (APIs) describe the library name and methods
- Allows us to reuse code and work with modules of reasonable size
Java Libraries – standard system libraries

• `java.lang.*`: standard system library
  – Math: common mathematical functions
    • `abs()`, `max()`, `min()`, `random()`, ...
  – Integer, Double: wrapper classes of primitive types
  – String, StringBuilder
  – System
    • `System.in`, `System.out`, `System.arraycopy()`, ...
Java Libraries – imported system libraries

• java.util.Arrays
  – array manipulation, such as searching and sorting
• java.util.ArrayList
  – Resizable-array implementation of List
• java.util.Scanner
  – text input scanner
Libraries from the textbook

- Textbook libraries (stdlib.jar)
  - StdIn
  - StdOut
  - StdRandom
  - StdStats
  - ...

Have you heard about the object-oriented way to become wealthy?
Inheritance

• Different types of employees
  – Hourly employee
  – Salaried employee
  – Volunteer

• What features are common for all employees?
• What features are specific?
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 extends and super

• Use `extends` keyword to declare a subclass
• Use `super` keyword to refer to the superclass
  – To call a superclass constructor
    • Explicit : `super()`
    • Implicit: if no superclass constructor is explicitly invoked, compiler puts `super()` as the first statement in the constructor
  – To call a superclass method
    • `Super.methodName()`
  – To access a superclass public data field
    • `Super.fieldName`
public class Employee
{
    private String name; // name of the employee

    public Employee (String n) { name = n; }
    public Employee () { name = "Unknown"; }

    public String getName() { return name; }
    public String toString() { return name; }
}

public class EmployeeTester
{
    public static void main(String[] args) {
        Employee e = new Employee("The Best Employee");
        System.out.println(e);
    }
}
public class Employee
{
    private String name; // name of the employee

    public Employee (String n) { name = n; }
    public Employee () { name = "Unknown"; }

    public String getName() { return name; }
    public String toString() { return name; }
}

public class SalariedEmployee extends Employee
{
    private double weeklySalary;

    public SalariedEmployee(String n, double salary) {
        super(n);
        weeklySalary = salary;
    }

    public double earnings() {
        return weeklySalary;
    }
}
Calling Superclass Methods

• super can be used to call method from superclass

```java
public class Employee
{
    private String name; // name of the employee

    public Employee(String n) { name = n; }
    public Employee() { name = "Unknown"; }

    public String getName() { return name; }
    public String toString() { return name; }
}

public class SalariedEmployee extends Employee
{
    // ...

    public double printName() {
        System.out.println(super.getName());
    }
}
```
Overriding Methods in the Superclass

• Subclass can modify the implementation of a method defined in the superclass

• *Method overriding*

• A private method cannot be overridden, because it is not accessible outside its own class

• Different from overloading
public class Employee
{
    private String name; // name of the employee

    public Employee (String n) { name = n; }
    public Employee () { name = "Unknown"; }

    public String getName() { return name; }
    public String toString() { return name; }
}

public class SalariedEmployee extends Employee
{
    private double weeklySalary;

    public SalariedEmployee(String n, double salary) {
        super(n);
        weeklySalary = salary;
    }

    public double earnings() {
        return weeklySalary;
    }
}
Overriding toString() Method

class Employee
{
    private String name;  // name of the employee

    public Employee(String n) { name = n; }
    public Employee() { name = "Unknown"; }

    public String getName() { return name; }
    public String toString() { return name; }
}

class SalariedEmployee extends Employee
{
    // Other methods omitted
    public String toString() {
        return name + "", " + earnings(); // wrong, why?
    }
}
public class Employee
{
    private String name; // name of the employee

    public Employee (String n) { name = n; }
    public Employee () { name = "Unknown"; }

    public String getName() { return name; }
    public String toString() { return name; }
}

public class SalariedEmployee extends Employee
{
    // Other methods omitted
    public String toString() {
        return getName() + "", " + earnings(); // correct
    }
}
public class Employee {
    private String name; // name of the employee
    // other methods omitted
    public String getName() { return name; }
    public String toString() { return name; }
}

public class SalariedEmployee extends Employee {
    // Other methods omitted
    public String toString() {
        return getName() + "", " + earnings(); // correct
    }
}

public class EmployeeTester {
    public static void main(String[] args) {
        SalariedEmployee e = new SalariedEmployee("BestEmployee", 2000);
        System.out.println(e); // System.out.println(e.toString());
    }
}
public class Employee {
    private String name; // name of the employee
    // other methods omitted
    public String getName() { return name; }
    public String toString() { return name; }
}

public class SalariedEmployee extends Employee {
    // Other methods omitted
    public String toString() {
        return getName() + " , " + earnings(); // correct
    }
}

public class EmployeeTester {
    public static void main(String[] args) {
        Employee e = new SalariedEmployee("BestEmployee", 2000);
        System.out.println(e); // System.out.println(e.toString());
    }
}
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
  – Use the `instanceof` operator to test
instanceof

```java
object instanceof TypeName

Example:
if (anObject instanceof Employee)
{
    Employee e = (Employee) anObject;
    . . .
}

Purpose:
To return true if the object is an instance of TypeName (or one of its subtypes), and false otherwise
```
public class EmployeeTester {
    public static void main(String[] args) {
        SalariedEmployee s = new SalariedEmployee("S", 2000);
        Employee e = new Employee("E");
        e = s;
        s = (SalariedEmployee) e;
        if (e instanceof SalariedEmployee)
            s = (SalariedEmployee) e;
        // HourlyEmployee h = (HourlyEmployee) e; //exception
        if (e instanceof HourlyEmployee)
            h = (HourlyEmployee) e;
        System.out.println(e);
    }
}
Polymorphism

- Method overriding allows Java to invoke method based on a particular object at runtime instead of declared type
- Late binding (as opposed to early binding for method overloading)
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
Review questions

• Which of the following statements are true?
  A. A subclass is a subset of a superclass.
  B. A subclass is usually created 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.
Another Example: Bank Accounts

- Bank Account
  - getBalance
  - deposit
  - withdraw
  - transfer
- Savings Account
  - Earns interest that compounds monthly
- Checking account
  - no interest
  - small number of free transactions per month, additional transactions are charged a small fee

- ~cs171000/share/code/BankAccount
CheckingAccount Class

• 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)*
public void deposit(double amount) {
    transactionCount++; 

    // Now to add amount to balance
    balance = balance + amount; // wrong
}
public void deposit(double amount) {
    transactionCount++;

    // How to add amount to balance
    deposit(amount); // wrong
}

Implementing deposit() method
public void deposit(double amount) {
    transactionCount++;

    // How to add amount to balance
    super.deposit(amount); // correct
}

Implementing deposit() method
Bonus Question

• BankAccountTester.java
  – What’s the output?
  – Hint: in the `transfer` method, depending on types of `other`, different versions of `withdraw` and `deposit` are invoked - polymorphism