Chapter 4

Subclasses and inheritance

Lesson page 4-1. Subclasses

Activity 4-1-1 The need for better structuring mechanisms

Activity 4-1-2 The subclass

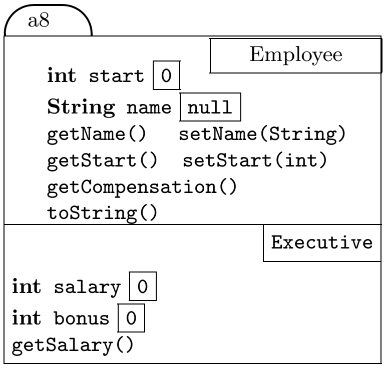
Question 1. Class `B` is a subclass of class `A` if class `B` extends class `A` –or if class `B` is a subclass of some class `C` that is a subclass of `A`.

Question 2. Class `B` is a superclass of class `A` if `A` is a subclass of `B`.

Question 3. In Java, a subclass inherits the instance methods and variables that are declared in or inherited by its superclass.

Question 4. The underlined answers are: `B`; `A`; `A`; `B`.

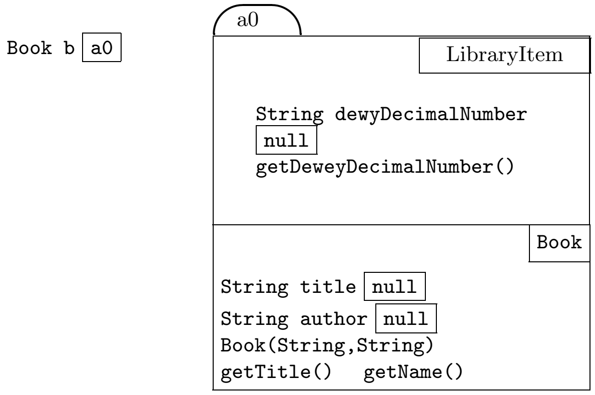
Question 5. Here is the instance:



Activity 4-1-3 The subclass (continued)

Question 6. True. A subclass inherits all the instance variables and methods of its superclass `C`. However, it won't be able to reference the inherited variables and methods that are declared **private**.

Question 7. Below is the filled-in variable and object:



Lesson page 4-2. Constructors and inherited methods

Activity 4-2-1 Writing a constructor for a subclass

Question 1. Of the two choices given, it is best to initialize inherited fields using a call on a constructor in the superclass.

Question 2. This call must be the first statement in the body of the constructor: `super(<`*`arguments`*`>);`

Activity 4-2-2 Overriding an inherited method

Question 3. In Java, a method in a subclass overrides a method with the same signature in a superclass.

Question 4. Overriding allows one to define a general method and then to specialize it in each subclass.

Activity 4-2-3 Calling an overridden method of the superclass

Question 5. The underlined answer is: `super.`.

Activity 4-2-4 Use of keywords this and super

Question 6. Keyword `super` is used (1) as the name in a constructor call in order to call a constructor of the superclass and (2) as a prefix (along with "`.`") of a reference to an instance variable or method to reference the variable or method of the superclass.

Question 7. Keyword `this` is used (1) as the name in a constructor call in order to call a constructor of the class in which the call appears and (2) as a reference to the object in which it appears. For example, in the latter case, "**this**.x" refers to variable x of the object, and "**this**" can be used by itself as an argument to denote the (name of the) object.

Activity 4-2-5 Exercises on subclasses

Activity 4-2-6 Access modifier protected

Question 8. True, because all your classes will be in the default package.

Question 9. A protected entity can be referenced in the class in which it is declared, in subclasses of that class, and in other classes in the same package.

Question 10. From any class in the same package.

Activity 4-2-7 The class hierarchy

Question 11. Class Object is at the top of the class hierarchy.

Question 12. The most useful methods in Object are equals and toString.

Question 13. Here is the constructor for class Hourly:

```
// Constructor: an instance with hire
// date year and salary salary
public Hourly(int year, int salary) {
    super(year);
    this.salary= salary;
}
```

Question 14. The list below shows the class hierarchy:

```
Object
    Employee
        Exec
        PartTime
        Hourly
            Temp
        Salaried
```

Lesson page 4-3. Casting and a new model of execution of method calls

Activity 4-3-1 Widening

Question 1. If B is a subclass of class C, then C is wider than B.

Question 2. False.

Question 3. True. Compiling the program below would result in an error message like: Method stringBleh() not found in class C.

```
public class C {/* Empty class. */ }

public class SC extends C {
    public String stringBleh()
        { return "Bleh."; }
}

public class Testing {
    public static void main(String[] pars) {
        SC subVar= new SC();
        String s= subVar.stringBleh();
        C superVar= subVar;
        s= superVar.stringBleh(); // ILLEGAL!
    }
}
```

Question 4. The legal statement is: `A a= new B();`

Question 5. The apparent class is `Animal` because `s` *appears* to be of type `Animal`. The real class is `Cat`, and `s` contains all `Cat` information.

Question 6. The one defined in `Cat`.

Activity 4-3-2 Narrowing

Question 7. A subclass is a narrower class-type than a superclass.

Question 8. This one does not have to be explicit: `A a= (A) new B();`

Question 9. False; no information is lost.

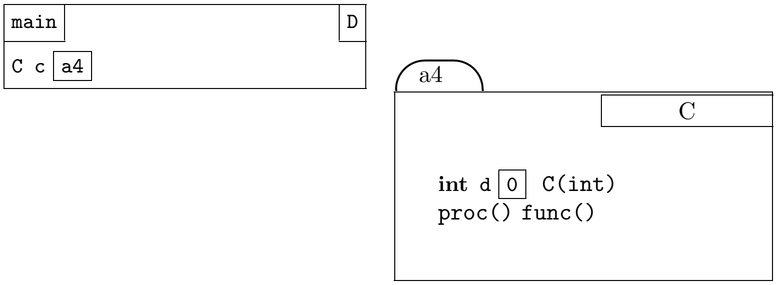
Question 10. The meaning is the value of the sentence "`x` is an instance of class `C` —or can be cast to class `C`".

Question 11. `a instanceof C`

Activity 4-3-3 Execution of a method call

Question 12. A function call is an expression, and its evaluation yields a value; a procedure call is a statement.

Question 13. We show below the frame for the call and the resulting object.



Activity 4-3-4 Referencing an item within a method body

Question 14. The function `noise` that is defined in `Cat` will be called. This is because one looks for the name `noise` in an upward direction, beginning at the bottom of the object.

Question 15. We do not answer this question.

Activity 4-3-5 A final look at class `Employee`

Question 16. Here are two classes. This is not the only way to write these classes! Use your imagination. For example, what happens to the number of hours worked when the Hourly employee is paid?

```
public private Hourly extends Employee {
    private double perHour= 6.75; // Minimum wage
    private double hrsWorked= 0; // Since last paycheque

    // Constructor: a person with name n, year hired d,
    // and hourly rate r.
    public Hourly(String n, int d, double r) {
        super(n, d);
        setRate(r);
    }

    // Set the per hour rate of this Hourly employee.
    public void setRate(double r) { perHour= r; }

    // = the per hour rate of this Hourly employee.
    public double getRate() { return perHour; }

    // Add h to the hours worked for this Hourly employee.
    public void addHours(double h)
        { hrsWorked= getHours() + h; }

    // = the number of hours worked by
    //   this Hourly employee.
    public double getHours() { return hrsWorked; }

    // = Hourly employee's total yearly compensation
    // (assumes paid yearly)
    public int getCompensation()
        { return (int)(perHour * getHours()); }

    // = String representation of this Hourly employee
    public String toString() {
        return super.toString() + ", hourly rate: $" +
            getRate() + ", hours worked: " + getHours();
    }
}

public private Temp extends Hourly {
    private int endDate; // Always in years

    // Constructor: a person with name n, year hired d,
    // hourly rate r, and ending year e.
    public Temp(String n, int d, double r, int e) {
        super(n, d, r);
        setEndDate(e);
    }

    // Set the ending year for this Temp employee
    public void setEndDate(int e) { endDate= e; }

    // = the ending year for this Temp employee
    public int getEndDate() { return endDate; }

    // = String representation of this Temp employee
    public String toString() {
        return super.toString() + ", ending date: " +
            getEndDate();
    }

    // Test the various methods of Temp.
    public static void main(String[] google) {
        Temp t= new Temp("Fred", 2000, 23, 2001);
        t.addHours(8);
        t.addHours(2);
        System.out.print(t + ", total compensation: $" +
            t.getCompensation());
    }
}
```

Lesson page 4-4. Object-oriented design

Question 1. The problem domain is the body of knowledge for which a program is being written or that the program is supposed to model.

Activity 4-4-1 Object-oriented design with subclasses

Question 2. We used noun phrases.

Question 3. False: every `B` is an `A`.

Question 4. The three guidelines are:

- Make `B` a subclass of `C` if each instance of `B` is a `C`.
- Structure classes to put behavior common to several classes in their superclass.
- Make instance variables private and provide getter methods for them.

Activity 4-4-2 Classes `Shape` and `Parallelogram`

Activity 4-4-3 Sublasses `Rhombus` and `Square`

Activity 4-4-4 Using the shape classes

Lesson page 4-5. Abstract classes

Activity 4-5-1 Abstract classes

Question 1. Method `drawShape` is there only to provide a method that can (and, as an `abstract` method, *must*) be overridden.

Question 2. Make a class abstract to prohibit its instantiation —to prohibit its use in a `new` expression.

Question 3. Make a method abstract to force each subclass to define the method.