Object Oriented Programming

#5 Inheritance

Hilmy A. Tawakal & Agung Prayoga

October 10, 2017

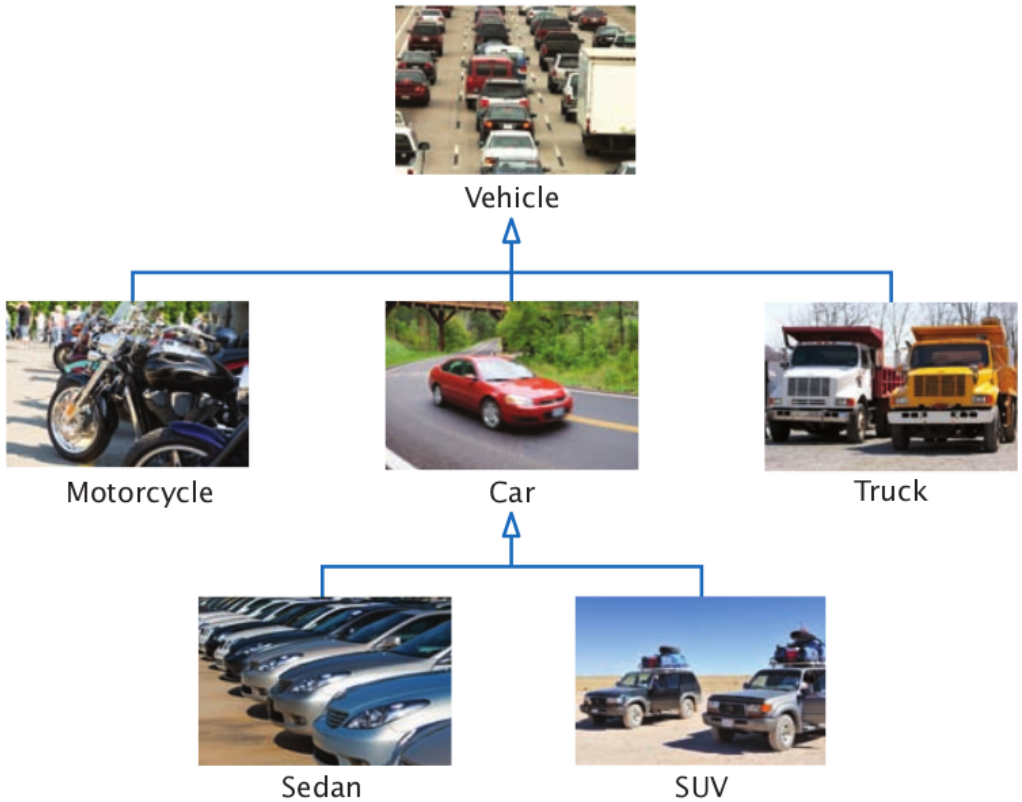
Table of contents

1. Inheritance Hierarchies
2. Implementing Subclasses
3. Override Methods

Definitions

Definition

In object-oriented design, **inheritance** is a relationship between a more general class (called the **superclass**) and a more specialized class (called the **subclass**).

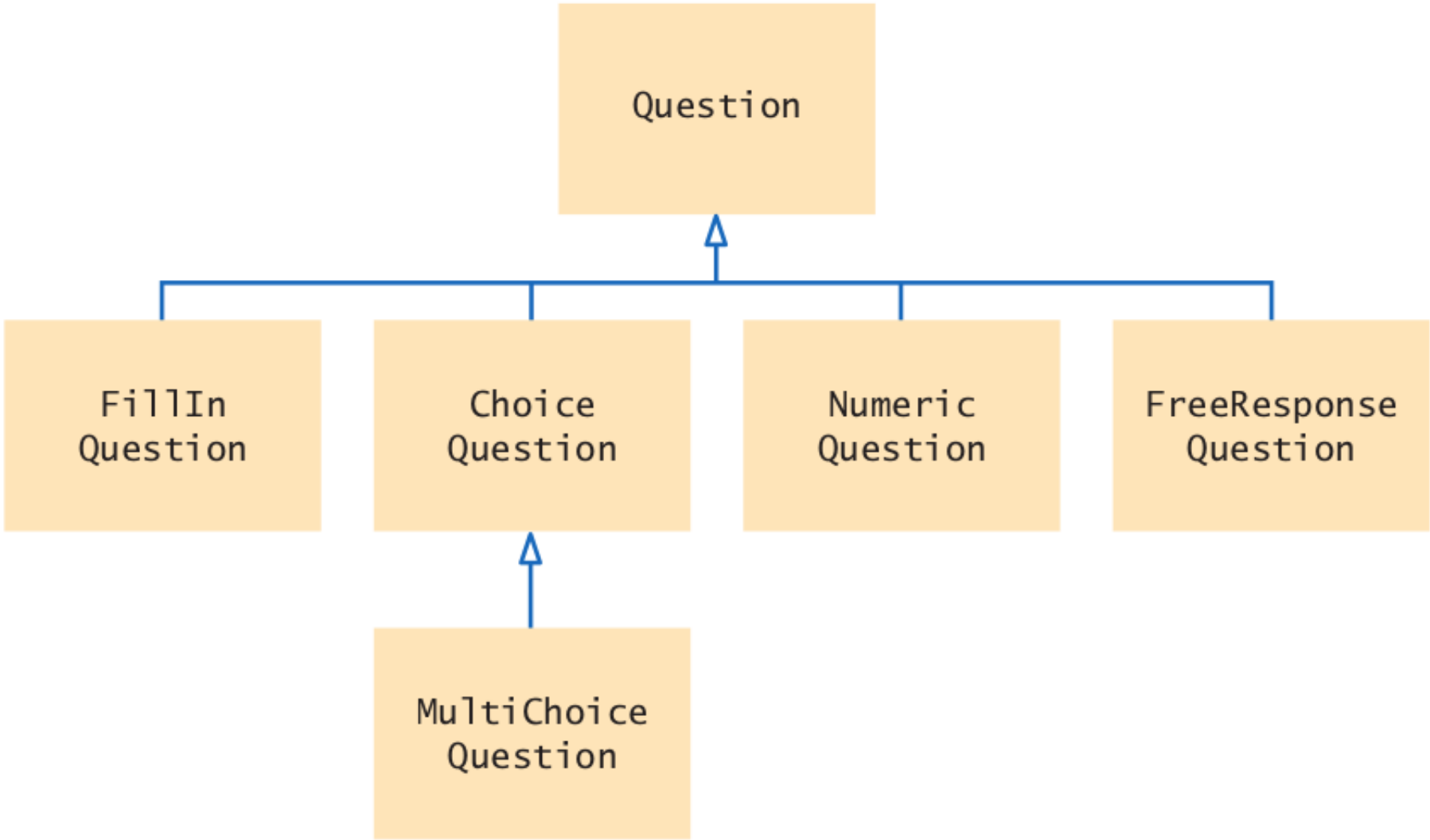


Advantages

Take an existing object type(collection of **fields** an **methods**) and **extend** it.

- create a special version of the code without re-writing any of the existing code (or even explicitly calling it!)
- End result is a more specific object type, called the sub-class / derived class / child class
- The original code is called the super-class / parent class / base class

Example


Question
FillIn Question
Choice Question
Numeric Question
FreeResponse Question
MultiChoice Question

Question.java

```
1  /**
2     A question with a text and an answer.
3  */
4  public class Question
5  {
6     private String text;
7     private String answer;
8
9     /**
10       Constructs a question with empty question and answer.
11    */
12    public Question()
13    {
14       text = "";
15       answer = "";
16    }
17
18    /**
19       Sets the question text.
20       @param questionText the text of this question
21    */
22    public void setText(String questionText)
23    {
24       text = questionText;
25    }
```

Question.java (cont.)

```
26
27      /**
28         Sets the answer for this question.
29         @param correctResponse the answer
30      */
31      public void setAnswer(String correctResponse)
32      {
33         answer = correctResponse;
34      }
35
36      /**
37         Checks a given response for correctness.
38         @param response the response to check
39         @return true if the response was correct, false otherwise
40      */
41      public boolean checkAnswer(String response)
42      {
43         return response.equals(answer);
44      }
45
46      /**
47         Displays this question.
48      */
49      public void display()
50      {
51         System.out.println(text);
52      }
53   }
```

QuestionDemo1.java

```
1  import java.util.Scanner;
2
3  /**
4     This program shows a simple quiz with one question.
5  */
6  public class QuestionDemo1
7  {
8     public static void main(String[] args)
9     {
10       Scanner in = new Scanner(System.in);
11
12       Question q = new Question();
13       q.setText("Who was the inventor of Java?");
14       q.setAnswer("James Gosling");
15
16       q.display();
17       System.out.print("Your answer: ");
18       String response = in.nextLine();
19       System.out.println(q.checkAnswer(response));
20    }
21 }
```

Program Run

```
Who was the inventor of Java?
Your answer: James Gosling
true
```

Extending Question.java

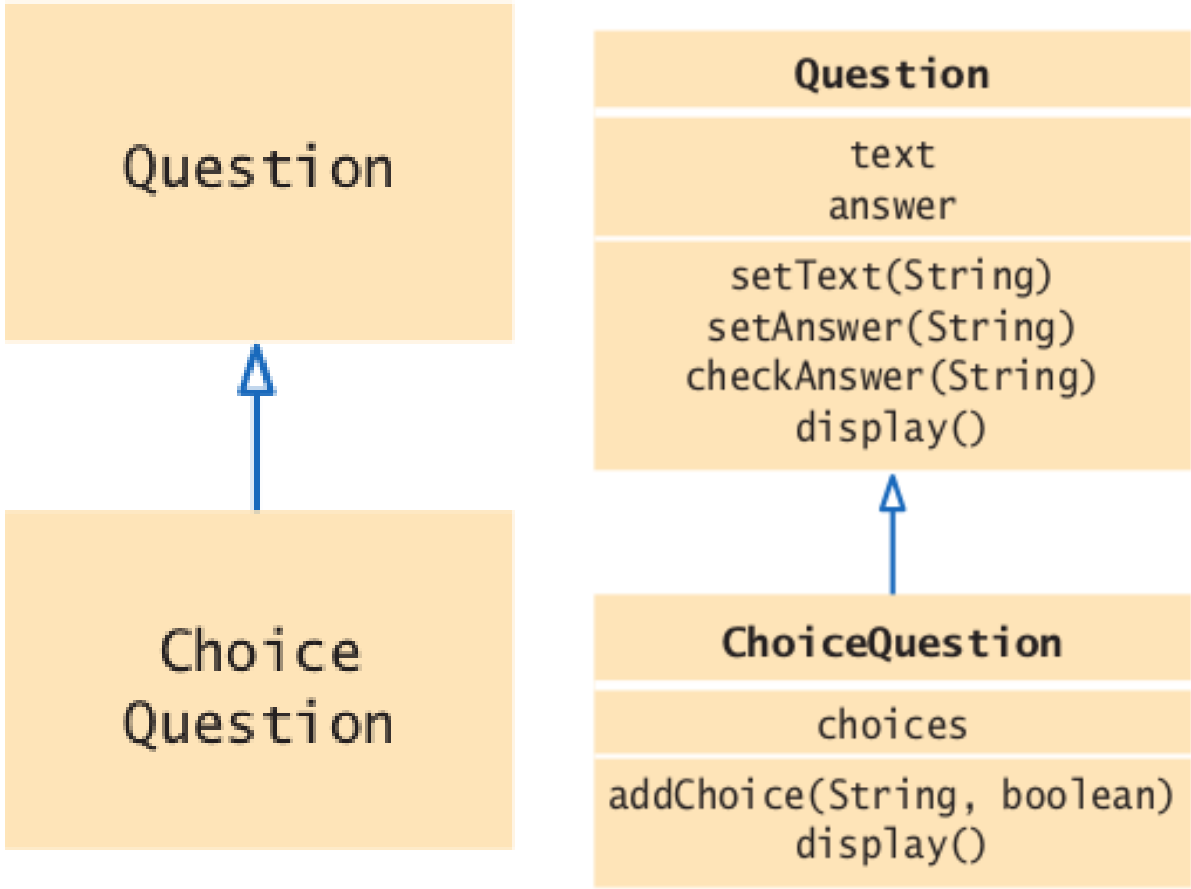
Suppose you want to write a program that handles questions such as the following:

```
In which country was the inventor of Java born?
1. Australia
2. Canada
3. Denmark
4. United States
```

You could write a ChoiceQuestion class from scratch, with methods to set up the question, display it, and check the answer. But you dont have to. Instead, use inheritance and implement ChoiceQuestion as a subclass of the Question class

ChoiceQuestion

In Java, you form a subclass by specifying what makes the subclass different from its superclass.



Subclass properties

- Subclass objects **automatically** have the instance variables that are declared in the **superclass**.
- You **only** declare instance variables that are **not** part of the **superclass** objects.
- The subclass **inherits all public methods** from the superclass.
- You **declare** any methods that are **new** to the subclass, and **change** the implementation of inherited methods if the inherited behavior is not appropriate.
- When you supply a **new implementation** for an inherited method, you **override** the method.

ChoiceQuestion difference

A ChoiceQuestion object differs from a Question object in three ways:

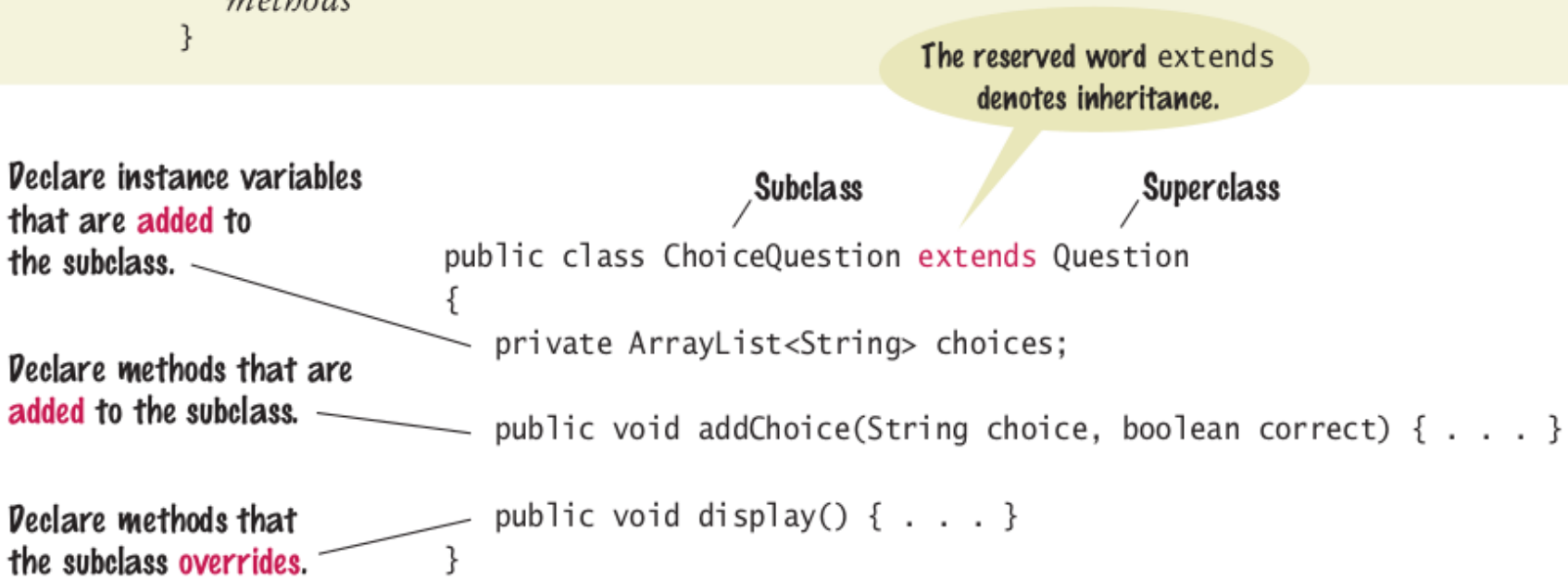
- Its objects store the various choices for the answer.
- There is a method for adding answer choices.
- The display method of the ChoiceQuestion class shows these choices so that the respondent can choose one of them.

```
public class ChoiceQuestion extends Question
{
  // This instance variable is added to the subclass
  private ArrayList<String> choices;
  // This method is added to the subclass
  public void addChoice(String choice, boolean correct) { . .
      . }
  // This method overrides a method from the superclass
  public void display() { . . . }
}
```

Subclass declaration

Syntax

```
public class SubclassName extends SuperclassName
{
   instance variables
   methods
}
```



Override Methods

- The subclass inherits the methods from the superclass.
- If you are not satisfied with the behavior of an inherited method, you override it by specifying a new implementation in the subclass.
- Consider the display method of the ChoiceQuestion class. It overrides the superclass display method in order to show the choices for the answer.
- This method extends the functionality of the superclass version. This means that the subclass method carries out the action of the superclass method, and it also does some additional work.
- In other cases, a subclass method replaces the functionality of a superclass method, implementing an entirely different behavior.

ChoiceQuestion example

- Display method of the ChoiceQuestion class needs to:
  - **Display the question text.**
  - **Display the answer choices.**
- The second part is easy because the answer choices are an instance variable of the subclass.
- But how do you get the question text? You cant access the text variable of the superclass directly because it is private.

ChoiceQuestion example

```
public class ChoiceQuestion
{
  . . .
  public void display()
  {
    // Display the question text
    . . .
    // Display the answer choices
    for (int i = 0; i < choices.size(); i++)
    {
      int choiceNumber = i + 1;
      System.out.println(choiceNumber + ": " + choices.get(i));
    }
  }
}
```

Calling Superclass method

Syntax `super.methodName(parameters);`

```
public void deposit(double amount)
{
   transactionCount++;
   super.deposit(amount);
}
```

Calls the method of the superclass instead of the method of the current class.

Keyword super

```
public void display()
{
  // Display the question text
  super.display(); // OK
  // Display the answer choices
  . . .
}
```

```
public void display()
{
  // Display the question text
  display(); // Errorinvokes this.display()
  . . .
}
```

Constructor with Superclass Initializer

Syntax

```
public ClassName(parameterType parameterName, . . .)
{
   super(arguments);
   . . .
}
```

The superclass constructor is called first.

The constructor body can contain additional statements.

```
public ChoiceQuestion(String questionText)
{
   super(questionText);
   choices = new ArrayList<String>;
}
```

If you omit the superclass constructor call, the superclass constructor with no arguments is invoked.

Review

1. Consider classes Manager and Employee . Which should be the superclass and which should be the subclass?
2. What are the inheritance relationships between classes BankAccount , Checking Account , and SavingsAccount?
3. Should a class Quiz inherit from the class Question ? Why or why not?
4. Suppose q is an object of the class Question and cq an object of the class Choice Question . Which of the following calls are legal?
   a. q.setAnswer(response)
   b. cq.setAnswer(response)
   c. q.addChoice(choice, true)
   d. cq.addChoice(choice, true)

Review

5. Identify the superclass and subclass in each of the following pairs of classes.
   a. Employee , Manager
   b. GraduateStudent , Student
   c. Person , Student
   d. Employee , Professor
   e. BankAccount , CheckingAccount
   f. Vehicle , Car
   g. Vehicle , Minivan
   h. Car , Minivan
   i. Truck , Vehicle
6. Consider a program for managing inventory in a small appliance store. Why isnt it useful to have a superclass SmallAppliance and subclasses Toaster , CarVacuum , TravelIron , and so on?

Review

7. Draw an inheritance diagram that shows the inheritance relationships between these classes.
   - Person
   - Employee
   - Student
   - Instructor
   - Classroom
   - Object
8. What inheritance relationships would you establish among the following classes?

- Student
- Employee
- Janitor
- Course
- ComputerLab
- Professor
- Secretary
- SeminarSpeaker
- Seminar
- TeachingAssistant
- DepartmentChair
- Person
- Lecture

Review

9. What is wrong with the following implementation of the display method?

```
public class ChoiceQuestion{
  . . .
  public void display(){
    System.out.println(text);
    for (int i = 0; i < choices.size(); i++){
      int choiceNumber = i + 1;
      System.out.println(choiceNumber + ": " +
          choices.get(i));
    }
  }
}
```

Review

10. What is wrong with the following implementation of the display method?

```
public class ChoiceQuestion{
  . . .
  public void display(){
    this.display();
    for (int i = 0; i < choices.size(); i++){
      int choiceNumber = i + 1;
      System.out.println(choiceNumber + ": " +
          choices.get(i));
    }
  }
}
```