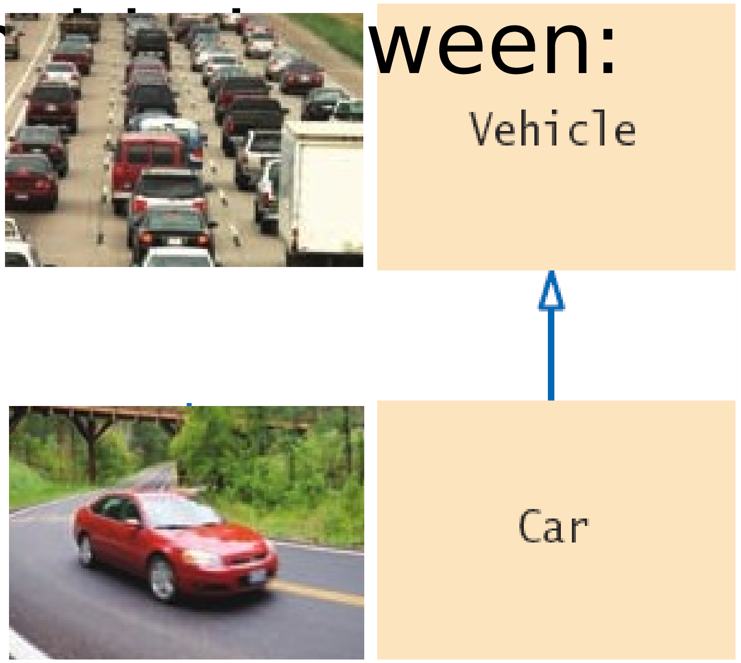
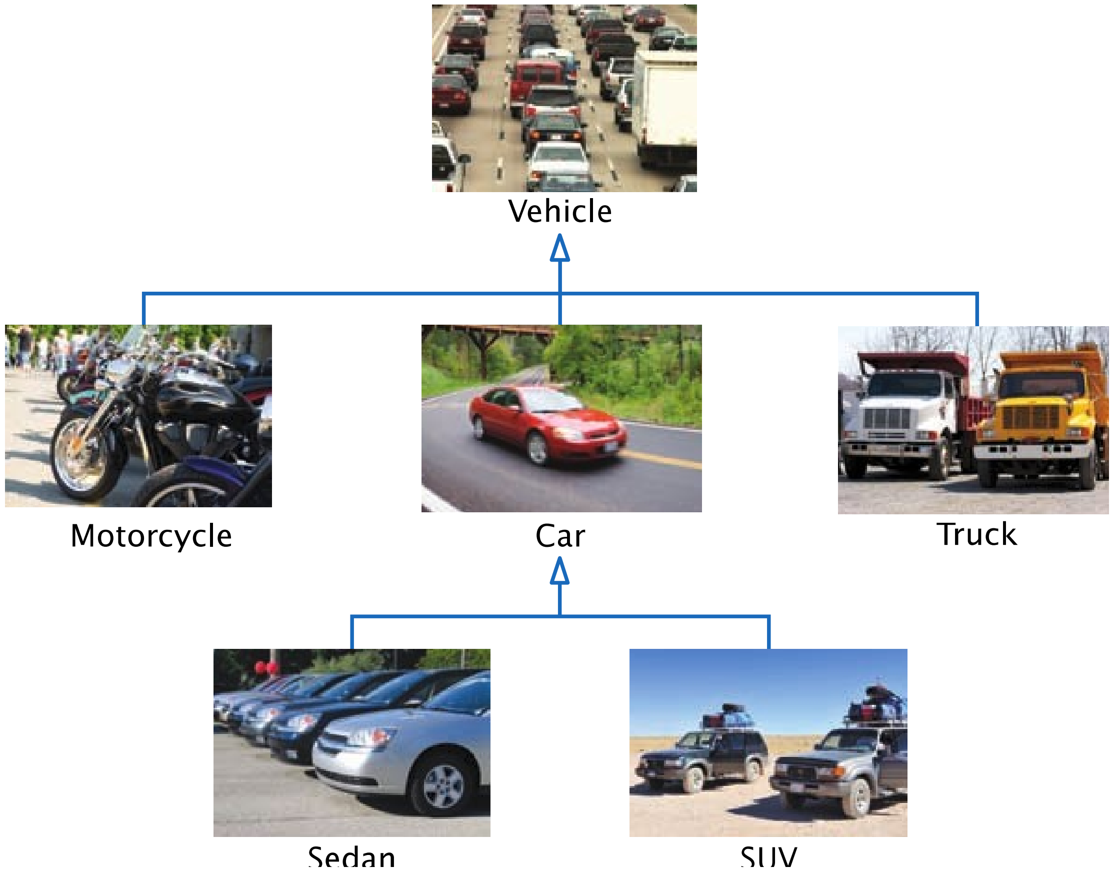
Inheritance Hierarchies

- In object-oriented programming, inheritance is a relation ween:
  - A *superclass*: a more generalized class
  - A *subclass*: a more specialized class
- The subclass ‘inherits’ data (variables) and behavior (methods) from the superclass



A Vehicle Class Hierarchy

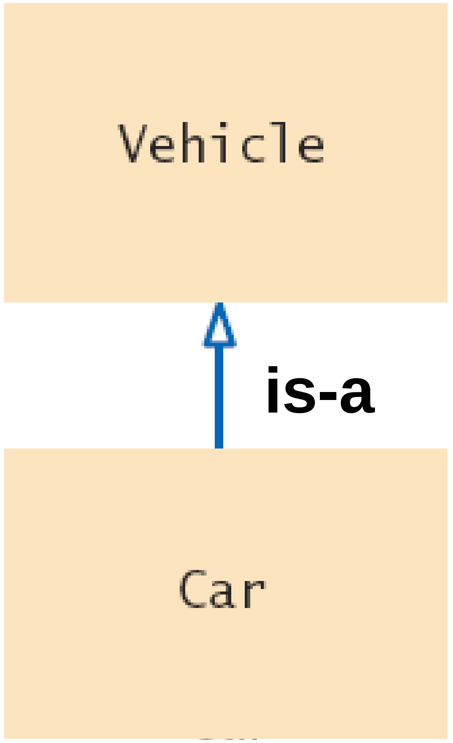
- General
- Specialized
- More Specific



The Substitution Principle

- Since the subclass Car “**is-a**” Vehicle
  - Car shares common traits with Vehicle
  - You can substitute a Car object in an algorithm that expects a Vehicle object

```
Car myCar = new Car(. . .);
processVehicle(myCar);
```



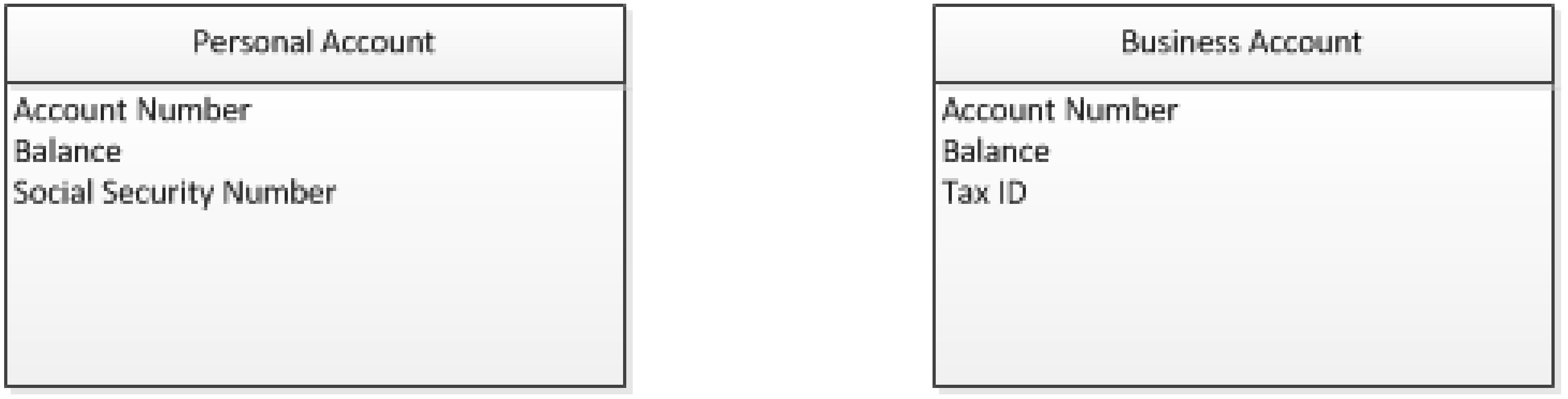
The ‘is-a’ relationship is represented by an arrow in a class diagram and means that the subclass can behave as an object of the superclass.

Identify Inheritance Relationships

- Inheritance describes an “IS A” relationship
  - A car ***is a*** vehicle
  - A bus ***is a*** vehicle
  - A car ***is a*** bus (Not inheritance relationship)
  - An employee ***is a*** person
  - A customer ***is a*** person
  - A checking account ***is a kind of*** bank account
  - A savings account ***is a type of*** bank account
  - A Bentley ***is a*** car which ***is a*** vehicle

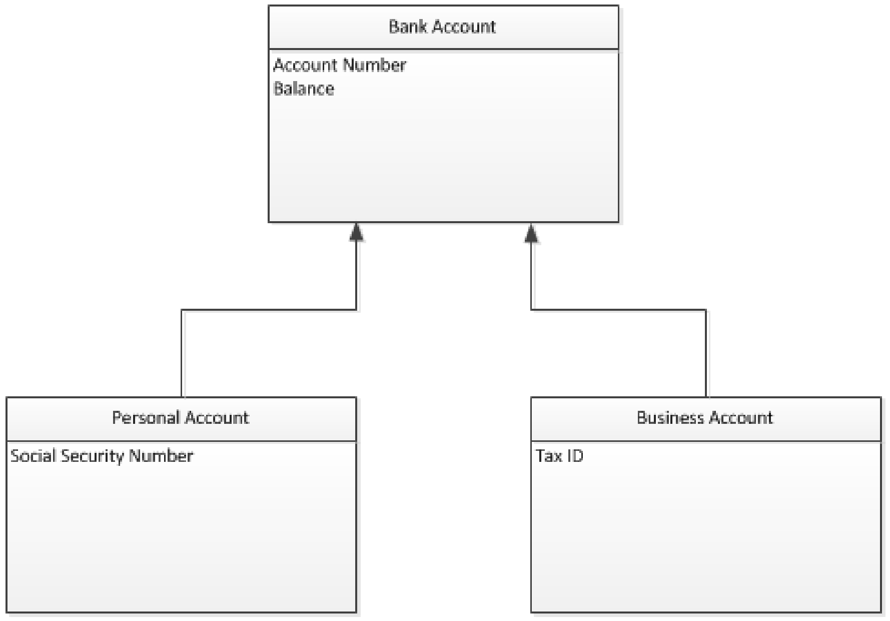
Bank Account Example

- Assume in an application you need to define two types of Bank Accounts: Business Account and Personal Account
- The two types of accounts share some common attributes: Account Number and Balance
- But they differ in some others which are specific to each type of account



Bank Account Example

- Using inheritance, you can define a Bank Account class that contain the general attributes that exist in both type of accounts (Account Number and Balance)
- Personal Account and Business Account inherit from Bank Account
- Each subclass defines the specific attributes that are applicable to themselves.

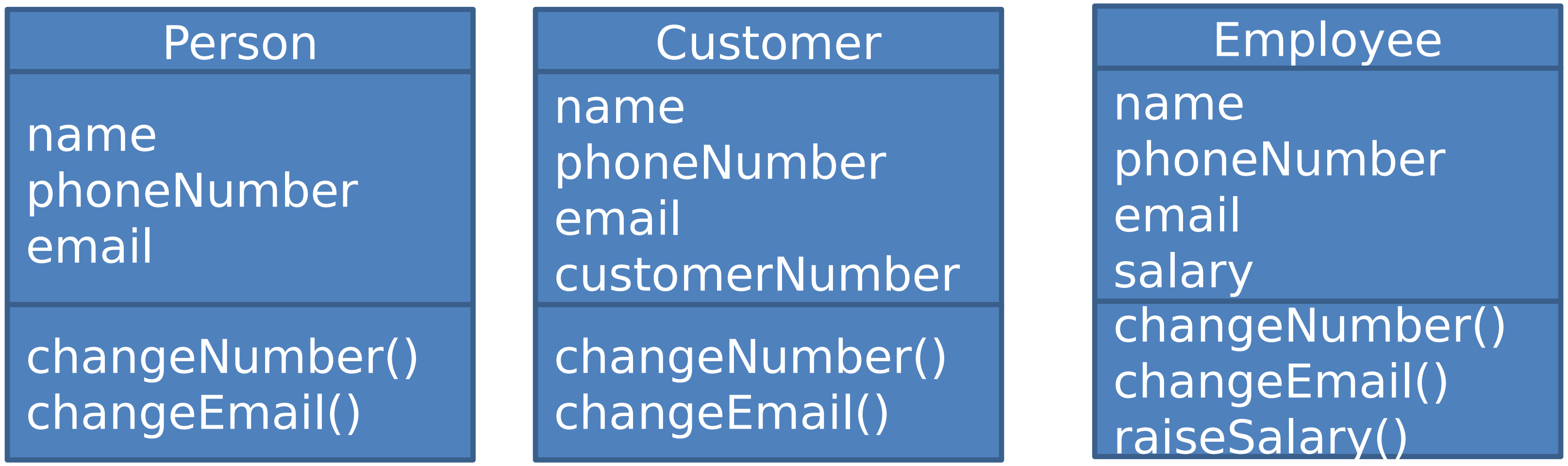


Advantages of Inheritance

- Inheritance is a form of ***abstraction.***
- It is used to save duplication of information in different classes.
- Inheritance is used to define a new class based on another class.
- It is also a form of code reuse because you don’t have to define classes from scratch. You can base it on an existing class.

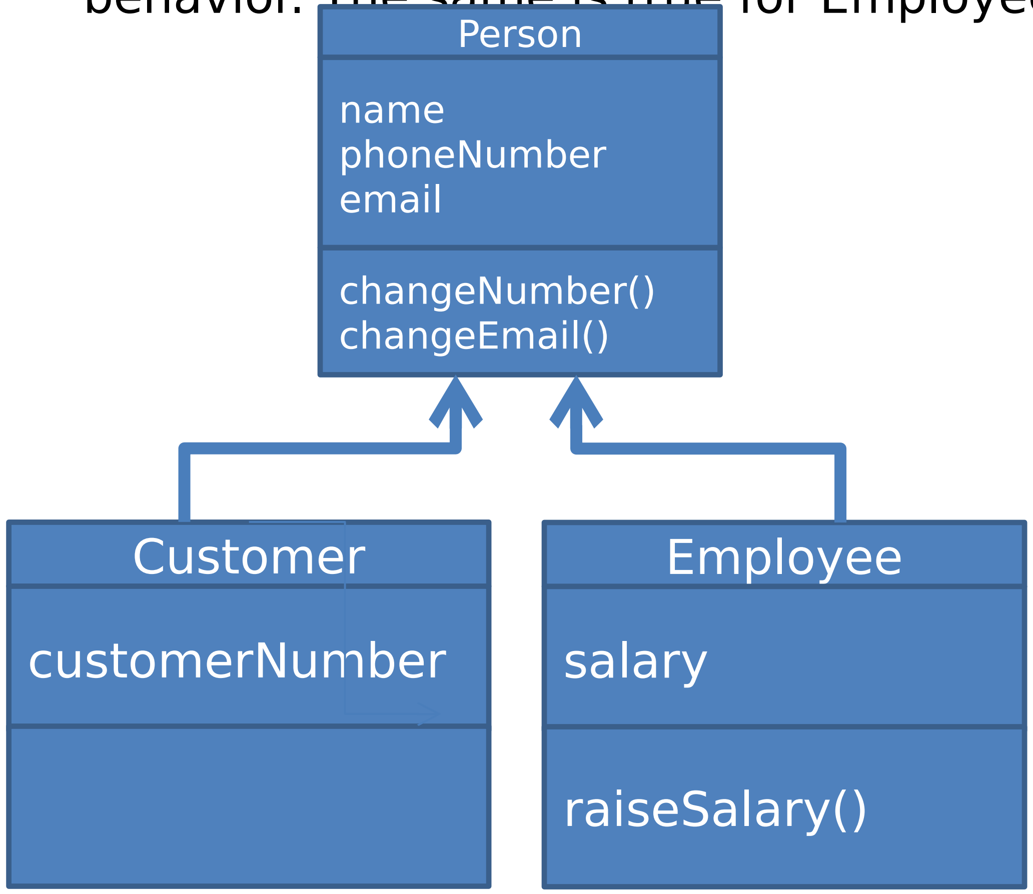
Inheritance Example

- Think of how you can apply the inheritance principle to the classes below:



Inheritance Example

Person class contains the general attributes and behaviors that is shared among all three classes. Customer is a type of Person; therefore, it can inherit the person attributes and behavior. The same is true for Employee.



Superclass
(Parent class)

Subclass
(Child class)

Customer inherits from Person or
Customer is derived from Person

Employee inherits from Person
Employee is derived from Person

Inheritance

- A subclass is created using the keyword **extends** as in:

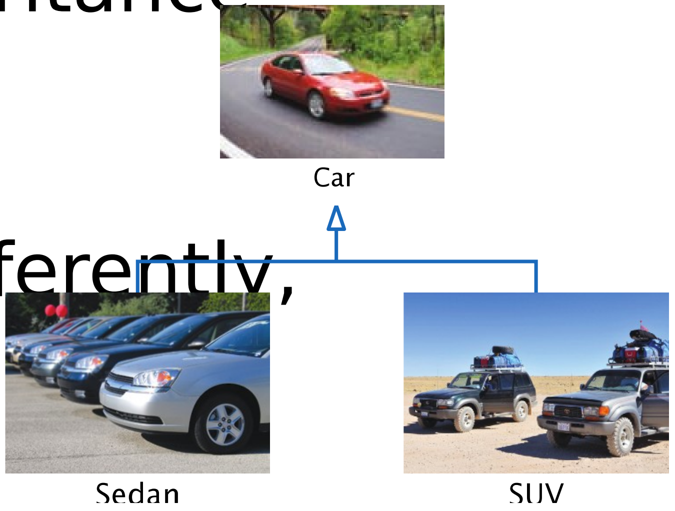
  Note : Subclasses should in some way modify the behavior of the class they are extending using additional variables and methods

```
class Customer extends Person
{

}
```

Programming Tip

- Use a Single Class for Variation in Values, Inheritance for Variation in Behavior
  - If two vehicles only vary by fuel efficiency, use an instance variable for the variation, not inheritance

```
// Car instance variable
double milesPerGallon;
```



  - If two vehicles behave differently, use inheritance

Be careful not to over-use inheritance

Inheriting from the Superclass

- Subclasses inherit from the superclass:
  - All public methods that it does not override
  - All instance variables
- The Subclass can
  - Add new instance variables
  - Add new methods
  - Change the implementation of inherited methods

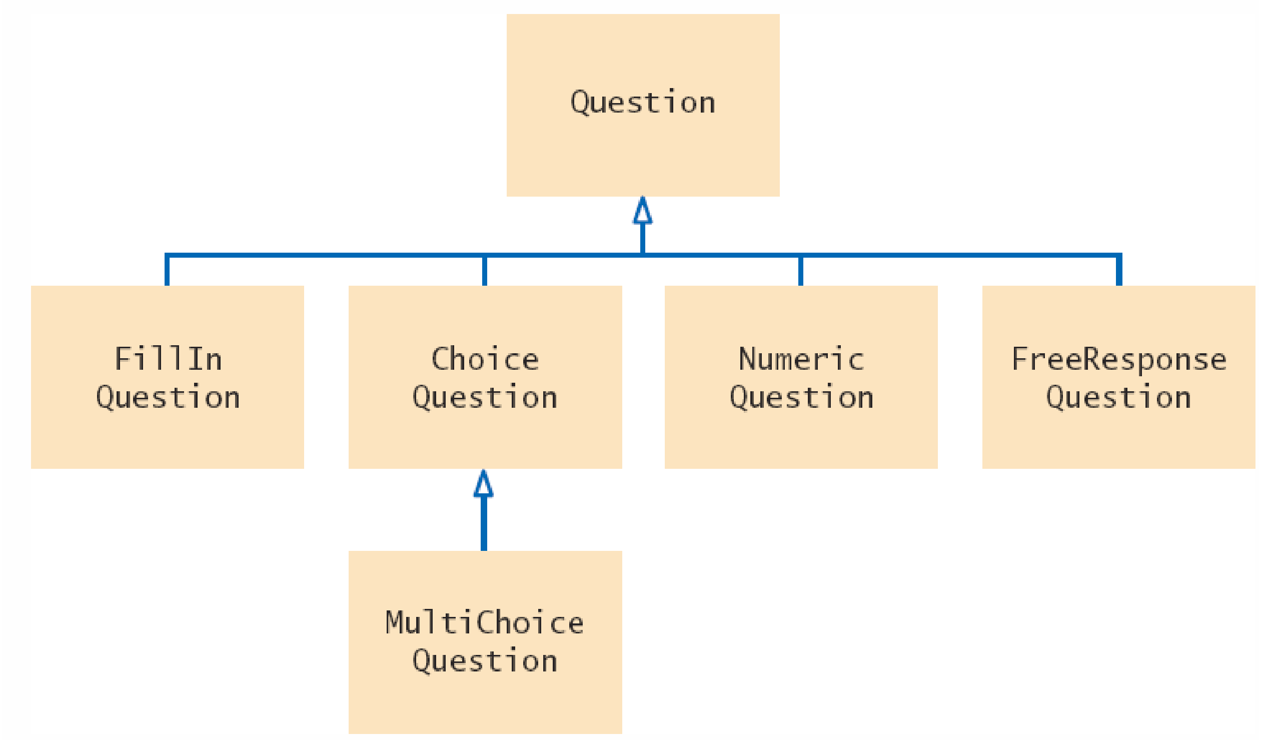
Form a subclass by specifying what is different from the superclass.

Quiz Question Hierarchy

- There are different types of quiz questions:
  1) Fill-in-the-blank
  2) Single answer choice
  3) Multiple answer choice
  4) Numeric answer
  5) Free Response

The ‘root’ of the hierarchy is shown at the top.

- A question can:
  - Display it’s text
  - Check for correct answer



Question.java (1)

```java
/**
    A question with a text and an answer.
*/
public class Question
{
    private String text;
    private String answer;

    /**
        Constructs a question with empty question and answer.
    */
    public Question()
    {
        text = "";
        answer = "";
    }

    /**
        Sets the question text.
        @param questionText the text of this question
    */
    public void setText(String questionText)
    {
        text = questionText;
    }
```

The class `Question` is the 'root' of the hierarchy, also known as the superclass

- Only handles Strings
- No support for:
  - Approximate values
  - Multiple answer choice

Question.java (2)

```java
    /**
       Sets the answer for this question.
       @param correctResponse the answer
    */
    public void setAnswer(String correctResponse)
    {
       answer = correctResponse;
    }

    /**
       Checks a given response for correctness.
       @param response the response to check
       @return true if the response was correct, false otherwise
    */
    public boolean checkAnswer(String response)
    {
       return response.equals(answer);
    }

    /**
       Displays this question.
    */
    public void display()
    {
       System.out.println(text);
    }
}
```

QuestionDemo1.java

```java
import java.util.ArrayList;
import java.util.Scanner;

/**
   This program shows a simple quiz with one question.
*/
public class QuestionDemo1
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      Question q = new Question();
      q.setText("Who was the inventor of Java?");
      q.setAnswer("James Gosling");

      q.display();
      System.out.print("Your answer: ");
      String response = in.nextLine();
      System.out.println(q.checkAnswer(response));
   }
}
```

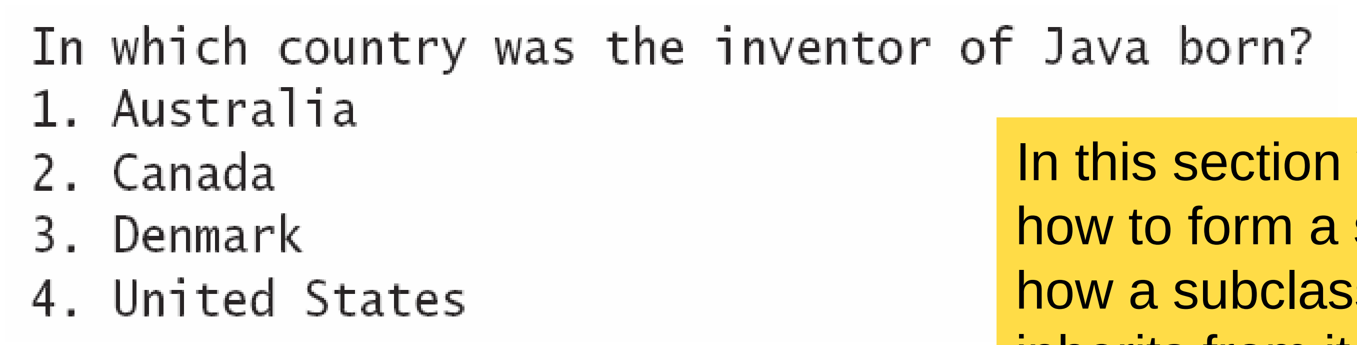
Program Run

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

Creates an object of the `Question` class and uses methods.

Implementing Subclasses

- Consider implementing `ChoiceQuestion` to handle:



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

In this section you will see how to form a subclass and how a subclass automatically inherits from its superclass

- How does `ChoiceQuestion` differ from `Question`?
  - It stores choices (1,2,3 and 4) in addition to the question
  - There must be a method for adding multiple choices
    - The display method will show these choices below the question, numbered appropriately

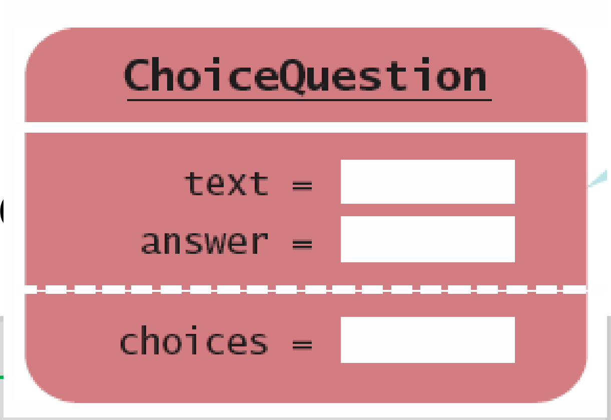
Overriding Superclass Methods

- Can you re-use any methods of the `Question` class?
  - Inherited methods perform exactly the same
  - If you need to change how a method works:
    - Write a new more specialized method in the subclass
    - Use the same method name as the superclass method you want to replace
    - It must take all of the same parameters
  - This will ***override*** the superclass method
- The new method will be invoked with the same method name when it [illegible] ject

A subclass can override a method of the superclass by providing a new implementation.

Planning the subclass

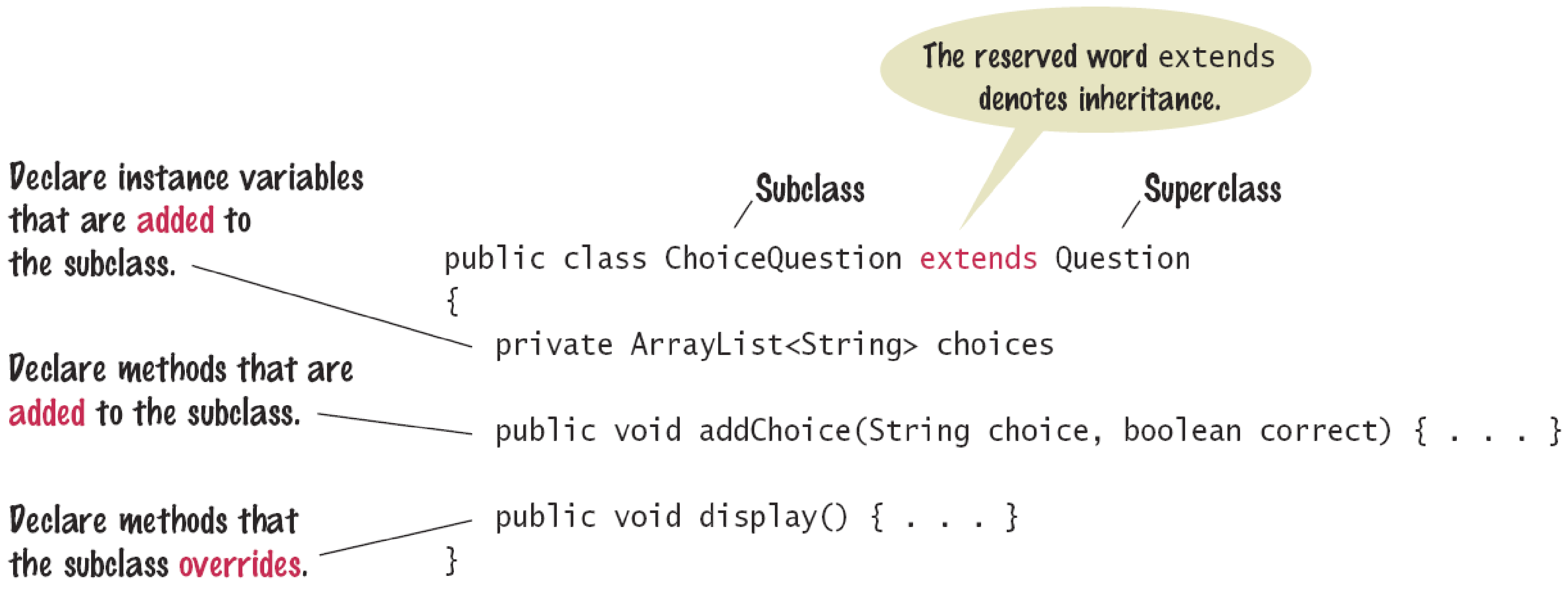
- Use the reserved word extends to inherit from Question
  - Inherits text and answer variab
  - Add new instance variable choi



```
public class ChoiceQuestion extends Questi
{
   // 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() { . . . }
}
```

Syntax : Subclass Declaration

- The subclass inherits from the superclass and ‘extends’ the functionality of the superclass



```
public class ChoiceQuestion extends Question
{
   private ArrayList<String> choices

   public void addChoice(String choice, boolean correct) { . . . }

   public void display() { . . . }
}
```

The reserved word extends denotes inheritance.
Subclass
Superclass
Declare instance variables that are added to the subclass.
Declare methods that are added to the subclass.
Declare methods that the subclass overrides.

Implementing addChoice

- The method will receive two parameters
  - The text for the choice
  - A boolean denoting if it is the correct choice or not
- It adds text as a choice, adds choice number to the text and calls the inherited `setAnswer` method

```
public void addChoice(String choice, boolean correct)
{
  choices.add(choice);
  if (correct)
  {
    // Convert choices.size() to string
    String choiceString = "" + choices.size();
    setAnswer(choiceString);
  }
}
```

`setAnswer()` is the same as calling `this.setAnswer()`

Common Error

- Replicating Instance Variables from the Superclass
  - A subclass cannot directly access private instance variables of the superclass

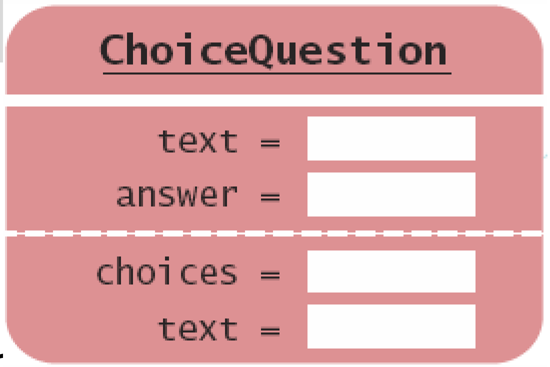
```
public class Question
{
   private String text;
   private String
    answer;
   . . .
```

```
public class ChoiceQuestion extends
   Question
{
   . . .
   text = questionText;    // Complier
    Error!
```

Common Error

- Do not try to fix the compiler error with a new instance variable of the same name

```
public class ChoiceQuestion extends
   Question
{
  private String text;  // Second copy
```



- The constructor sets one text variabl
- The display method outputs the other

Common Error

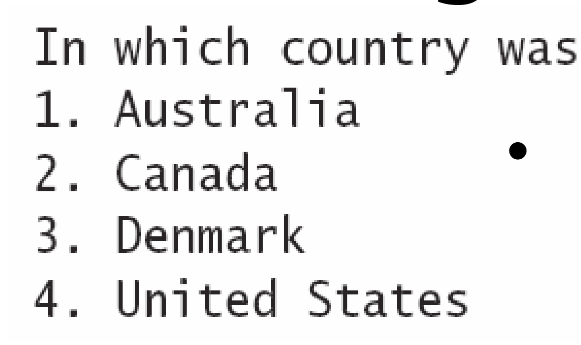
- Confusing *Super*- and *Sub*classes
  - The use of the terminology super and sub may be confusing
  - The Subclass `ChoiceQuestion` is an ‘extended’ and more powerful version of `Question`
    - Is it a ‘super’ version of `Question`?... NO.
- Super and Subclass terminology comes from set theory

  `ChoiceQuestion` is one of a **subset** of all objects that inherit from `Question`
  - The set of `Question` objects is a **superset** of `ChoiceQuestion` objects

Overriding Methods

- The `ChoiceQuestion` class needs a `display` method that overrides the `display` method of the `Question` class
- They are two different method implementations
- The two methods named `display` are:
  - `Question display`
    - Displays the instance variable text String
  - `ChoiceQuestion display`
    - Overrides `Question display` method
    - Displays the instance variable text String
    - Displays the local list of choices

Calling Superclass Methods



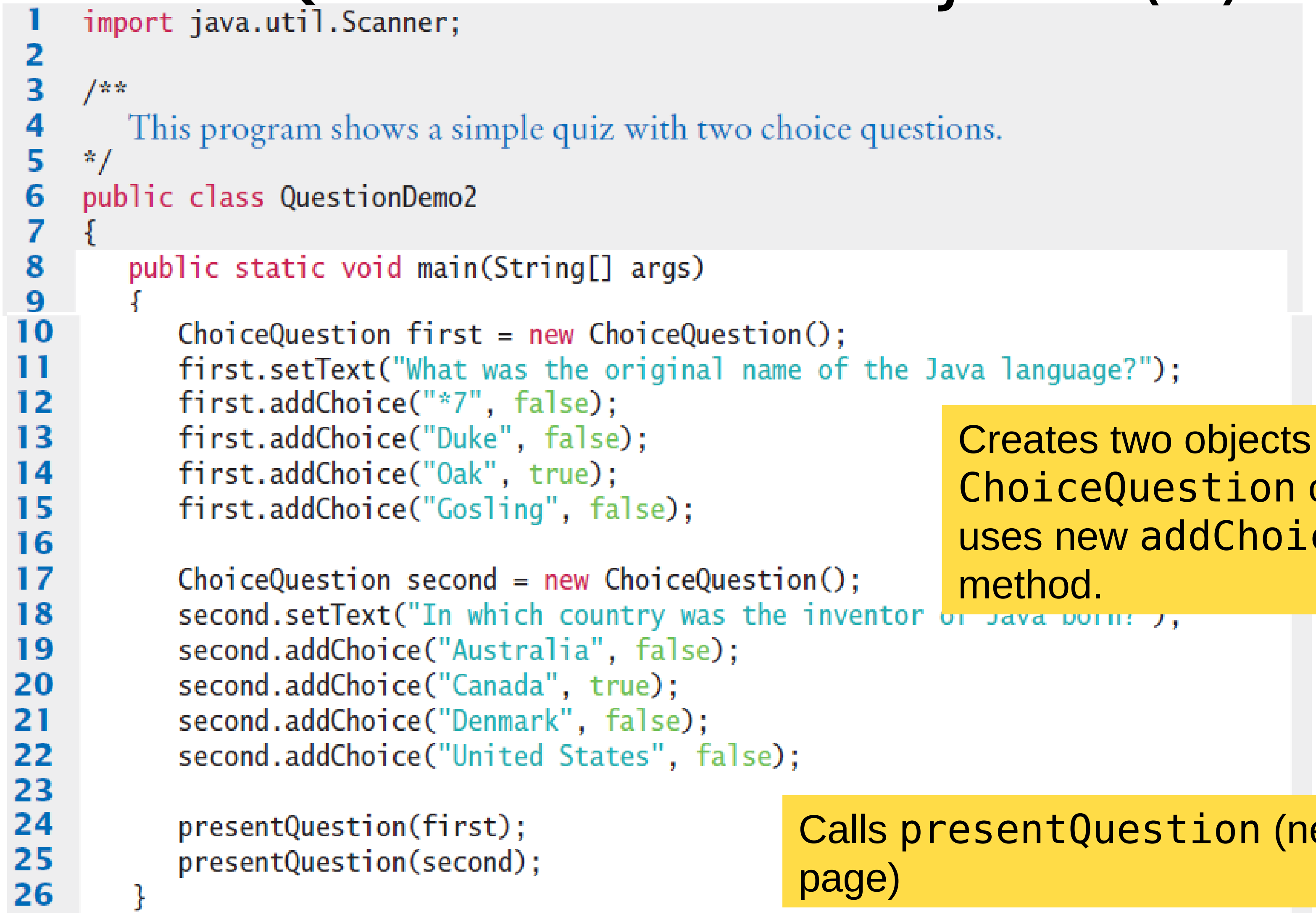
```
In which country was the inventor of Java born?
1. Australia
2. Canada
3. Denmark
4. United States
```

- Consider the display method of the `ChoiceQuestion` class
  - It needs to display the question AND the list of choices

- `text` is a private instance variable of the superclass
  - How do you get access to it to print the question?
  - Call the `display` me `Question`!
    - From a subclass, preface the metho name with:
    - `super.`

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

QuestionDemo2.java (1)

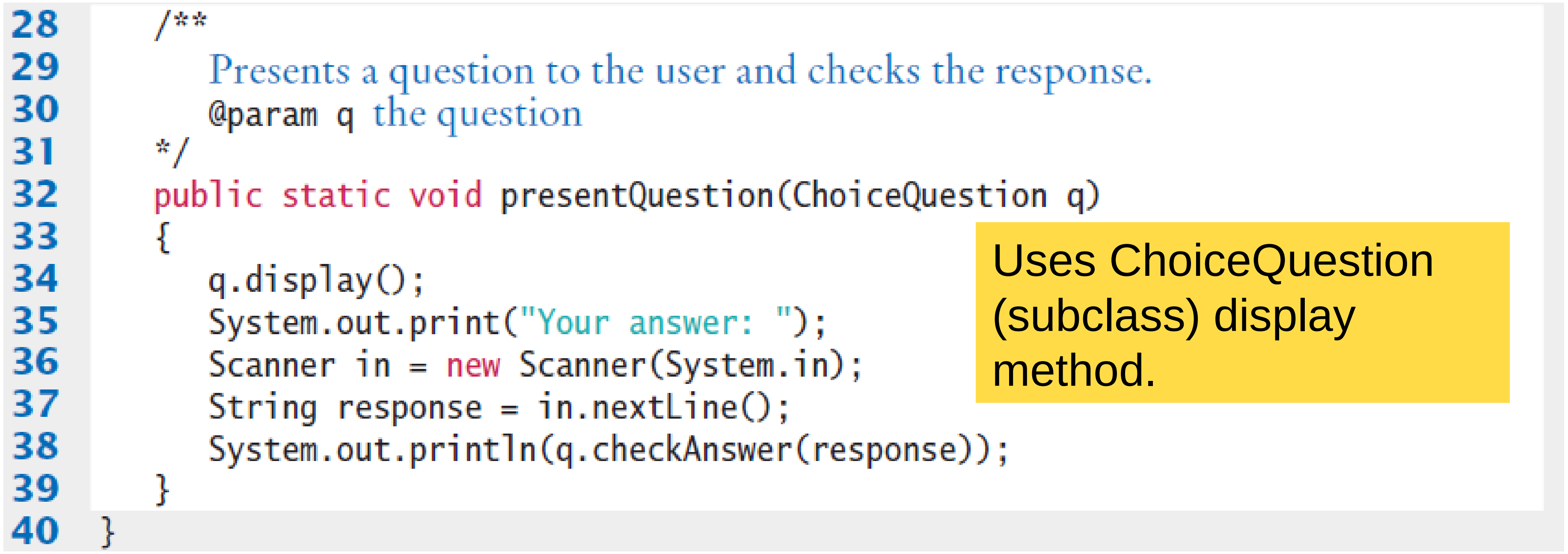


```
import java.util.Scanner;

/**
   This program shows a simple quiz with two choice questions.
*/
public class QuestionDemo2
{
   public static void main(String[] args)
   {
      ChoiceQuestion first = new ChoiceQuestion();
      first.setText("What was the original name of the Java language?");
      first.addChoice("*7", false);
      first.addChoice("Duke", false);
      first.addChoice("Oak", true);
      first.addChoice("Gosling", false);

      ChoiceQuestion second = new ChoiceQuestion();
      second.setText("In which country was the inventor of Java born?");
      second.addChoice("Australia", false);
      second.addChoice("Canada", true);
      second.addChoice("Denmark", false);
      second.addChoice("United States", false);

      presentQuestion(first);
      presentQuestion(second);
   }
```

Creates two objects of the `ChoiceQuestion` class, uses new `addChoice` method.

Calls `presentQuestion` (next page)

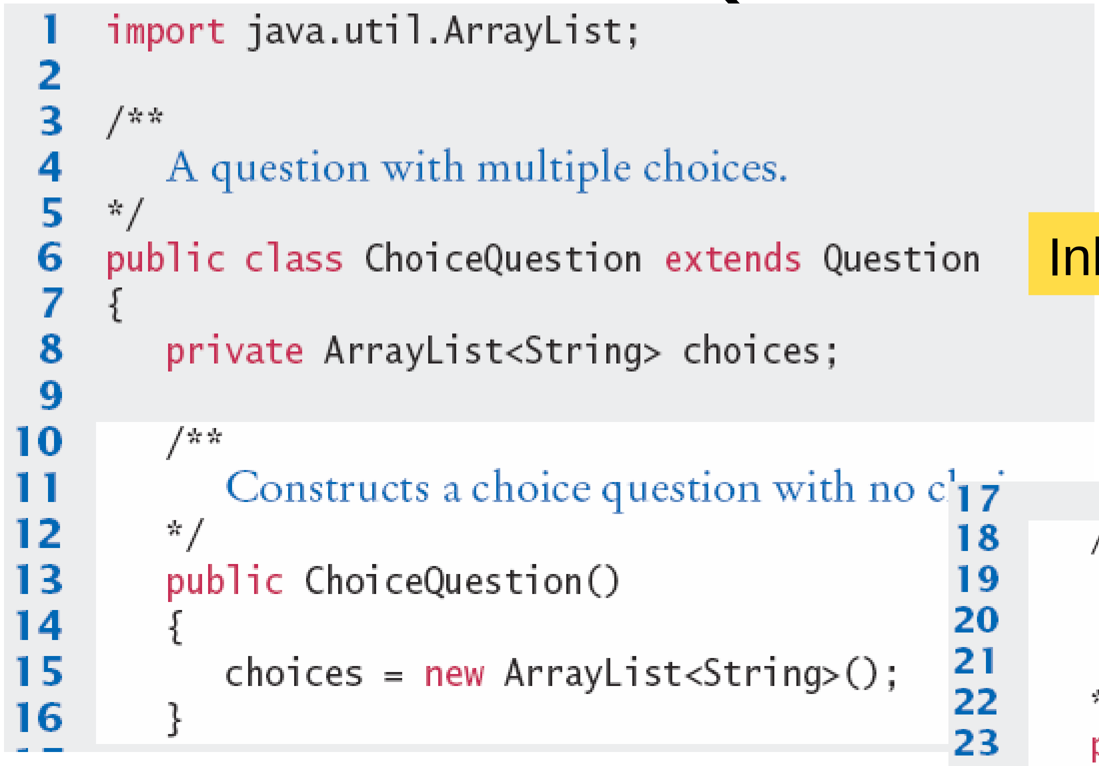
QuestionDemo2.java (2)



```java
    /**
       Presents a question to the user and checks the response.
       @param q the question
    */
    public static void presentQuestion(ChoiceQuestion q)
    {
       q.display();
       System.out.print("Your answer: ");
       Scanner in = new Scanner(System.in);
       String response = in.nextLine();
       System.out.println(q.checkAnswer(response));
    }
}
```

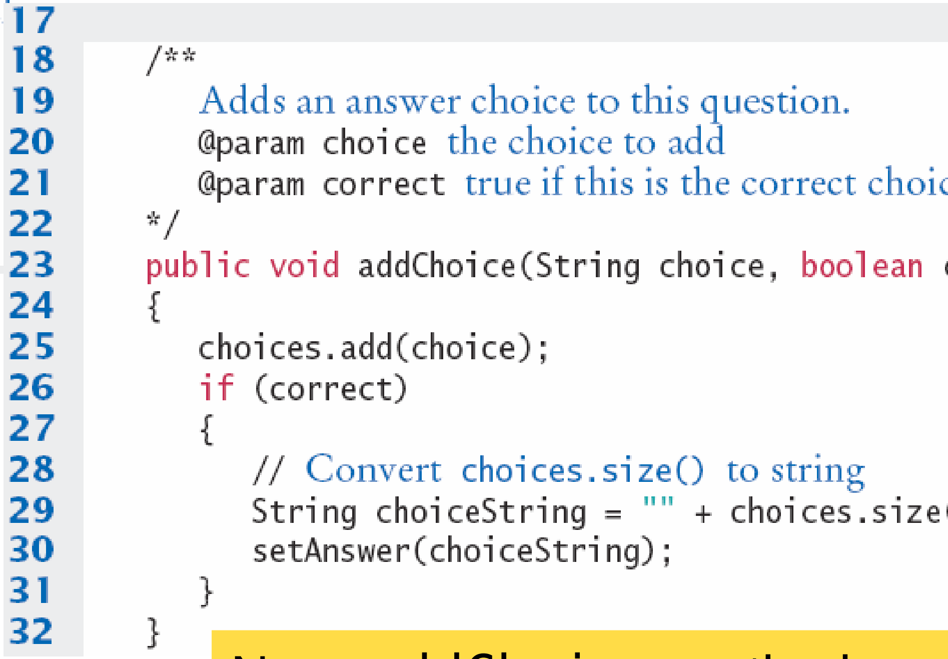
Uses ChoiceQuestion (subclass) display method.

ChoiceQuestion.java (1)



```
import java.util.ArrayList;

/**
   A question with multiple choices.
*/
public class ChoiceQuestion extends Question
{
   private ArrayList<String> choices;

   /**
      Constructs a choice question with no c
   */
   public ChoiceQuestion()
   {
      choices = new ArrayList<String>();
   }
```

Inherits from `Question` class.



```
   /**
      Adds an answer choice to this question.
      @param choice the choice to add
      @param correct true if this is the correct choic
   */
   public void addChoice(String choice, boolean c
   {
      choices.add(choice);
      if (correct)
      {
         // Convert choices.size() to string
         String choiceString = "" + choices.size
         setAnswer(choiceString);
      }
   }
```

New `addChoice` method.

ChoiceQuestion.java (2)

```java

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

Overridden `display` method.

Program Run

```
Who was the inventor of Java?
Your answer: Bjarne Stroustrup
false
In which country was the inventor of Java born?
1: Australia
2: Canada
3: Denmark
4: United States
Your answer: 2
true
```

Common Error

- Accidental Overloading

```
println(int x);
println(String s);  // Overloaded
```

  - Remember that **overloading** is when two methods share the same name but have different parameters
  - **Overriding** is where a subclass defines a method with the same name and exactly the same parameters as the superclass method
    - `Question display() method`
    - `ChoiceQuestion display() method`
  - If you intend to **override**, but change parameters, you will be **overloading** the inherited method, not **overriding** it
    - `ChoiceQuestion display(printStream out) method`

Common Error

- Forgetting to use super when invoking a Superclass method
  - Assume that Manager inherits from Employee
    - getSalary is an overridden method of Employee

```
public class Manager extends Employee
{
   . . .
   public double getSalary()
   {
     double baseSalary = getSalary();   // Manager.getSalary
     //  should be super.getSalary();   // Employee.getSalary
     return baseSalary + bonus;
   }
}
```

ChoiceQuestion Demo Constructor

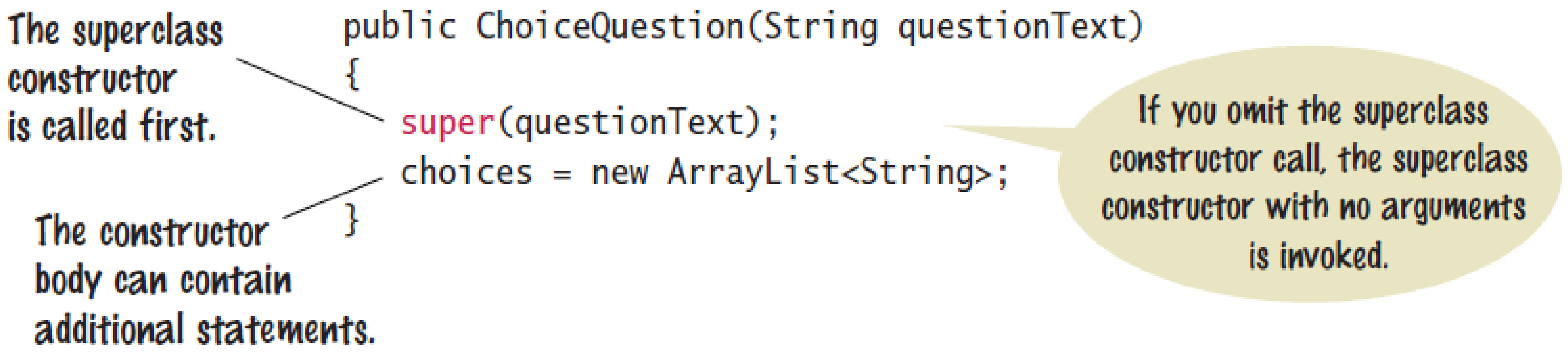
- Calling the Superclass Constructor
  - When a subclass is instantiated, it will call the superclass constructor with no arguments
  - If you prefer to call a more specific constructor, you can invoke it by using replacing the superclass name with the reserved word super follo

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

  - **It must be the first statement in your constructor**

Constructor with Superclass

- To initialize private instance variables ir superclass, invoke a specific constructor



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

The superclass constructor is called first.
The constructor body can contain additional statements.
If you omit the superclass constructor call, the superclass constructor with no arguments is invoked.

Special

- Final Methods and Classes
  - You can also ***prevent*** programmers from creating subclasses
    and override methods using `final`.
  - The ample:

```
public final class String { . . . }
```

  –

```
public class SecureAccount extends BankAccount
{
   . . .
   public final boolean checkPassword(String password)
   {
      . . .
   }
}
```

Special Topic

- protected Access
  - When trying to implement the display method of the ChoiceQuestion class, the display method wanted to access the instance variable text of the superclass, but it was private.
  - We chose to use a method of the superclass to display the text.
- Java provides a more elegant solution
  - The superclass can declare an instance variable as protected instead of private
  - protected data in an object can accessed by the methods of the object's class and all its subclass
  - But it can also be accessed by all

o

```
public class Question
{
  protected String
   text;
  . . .
}
```

If you want to grant access to the data to subclass methods only, consider making the accessor method protected.

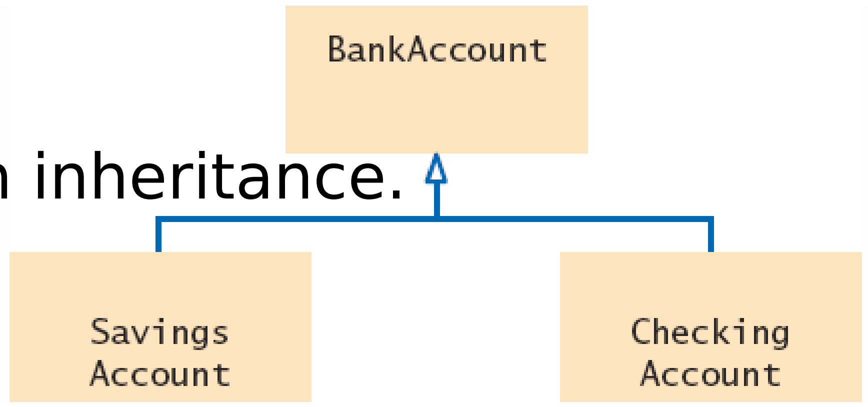
Steps to Using Inheritance

- As an example, we will consider a bank that offers customers the following account types:
  1) A savings account that earns interest. The interest compounds monthly and is based on the minimum monthly balance.
  2) A checking account that has no interest, gives you three free withdrawals per month, and charges a $1 transaction fee for each additional withdrawal.
- The program will manage a set of accounts of both types
  - It should be structured so that other account types can be added without affecting the main processing loop.
- The menu: `D)eposit W)ithdraw M)onth end Q)uit`
  - For deposits and withdrawals, query the account number and amount. Print the balance of the account after each transaction.
  - In the “Month end” command, accumulate interest or clear the transaction counter, depending on the type of the bank

Steps to Using Inheritance

1) List the classes that are part of the hierarchy.
   SavingsAccount
   CheckingAccount

2) Organize the classes into an inheritance. hierarchy
   Base on superclass BankAccount



3) Determine the common responsibilities.
   a. Write Pseudocode for each task
   b. Find common tasks

Using Inheritance

```
For each user command
    If it is a deposit or withdrawal
        Deposit or withdraw the amount from the specified account.
        Print the balance.
    If it is month end processing
        For each account
            Call month end processing.
            Print the balance.
```

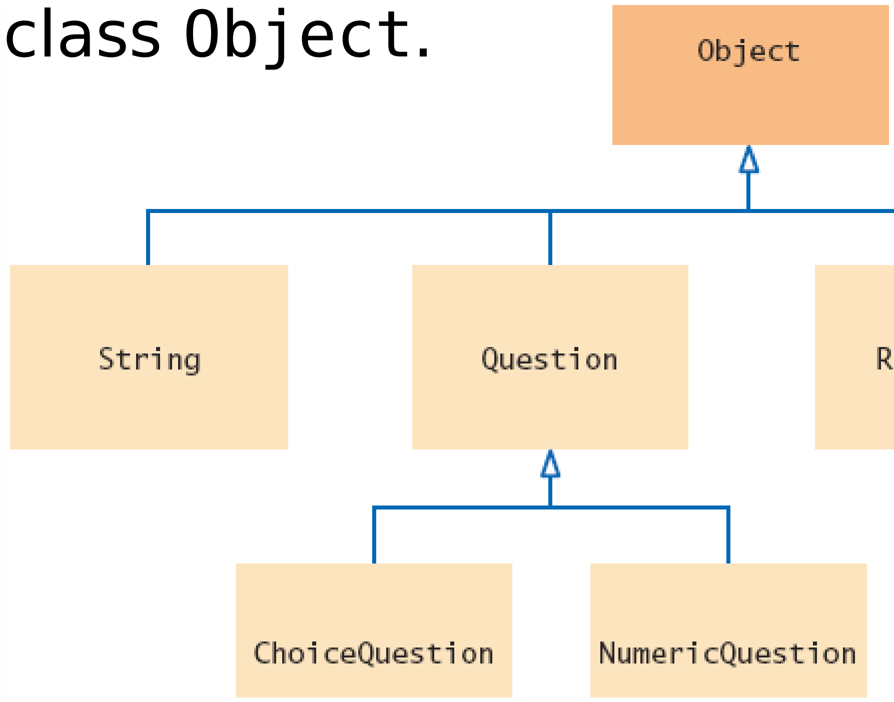
Deposit money.
Withdraw money.
Get the balance.
Carry out month end processing.

Steps to Using Inheritance

4) Decide which methods are overridden in subclasses.
   - For each subclass and each of the common responsibilities, decide whether the behavior can be inherited or whether it needs to be overridden
5) Declare the public interface of each subclass.
   - Typically, subclasses have responsibilities other than those of the superclass. List those, as well as the methods that need to be overridden.
   - You also need to specify how the objects of the subclasses should be constructed.
6) Identify instance variables.
   - List the instance variables for each class. Place instance variables that are common to all classes in the base of the hierarchy.
7) Implement constructors and methods.
8) Construct objects of different subclasses and process them.

`Object`: The Cosmic Superclass

- In Java, every class that is declared without an explicit extends clause automatically extends the class `Object`.



The methods of the `Object` class are very general. You will learn to override the `toString` method.

Writing a toString method

- The toString method returns a String representation for each object.
- The Rectangle class (java.awt) has a toString method

```
Rectangle box = new Rectangle(5, 10, 20, 30);
String s = box.toString();          // Call toString
   directly
// Sets s to
```

  – "java.awt.Rectangle[x=5,y=10,width=20,height=30]"

  – The toString method can also be invoked implicitly

```
System.out.println("box=" + box);   // Call toString
   implicitly
```

- The compiler can invoke the toString method, because it knows that *every object* has a toString method:
  – Every class extends the Object class, and can override

Overriding the toString method

- Example: Override the toString method for the BankAccount class

```
BankAccount momsSavings = new BankAccount(5000);
String s = momsSavings.toString();
// Sets s to something like "BankAccount@d24606bf"
```

  – All that is printed is the name of the class, followed by the hash code which can be used to tell objects (Chapter 10)

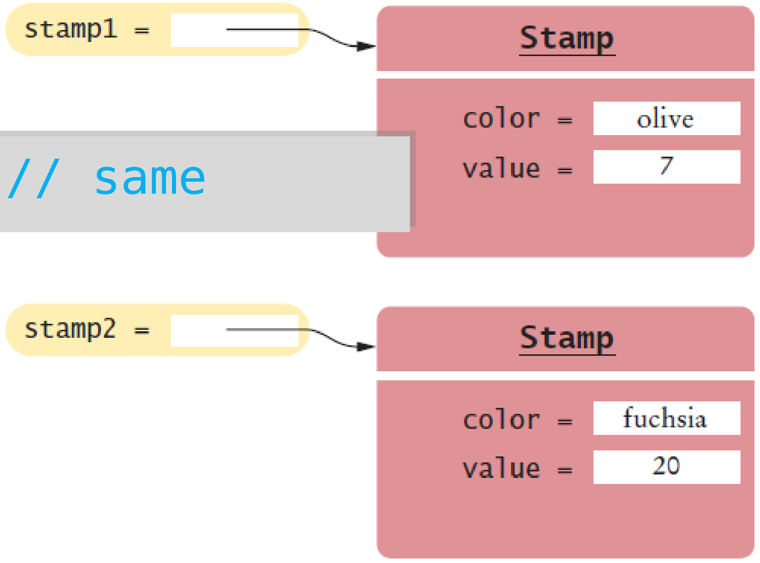
- We want to know what is inside the object

```
public class BankAccount
{
  public String toString()
  {
    // returns "BankAccount[balance=5000]"
    return "BankAccount[balance=" + balance + "]";
  }
}
```

Override the toString method to yield a string that describes the object’s state.

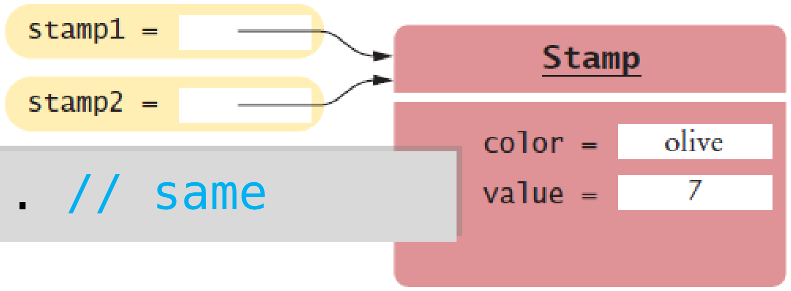
Overriding the `equals` method

- In addition to the `toString` method, the Object class `equals` method checks whether two objects have the same contents:



```
if (stamp1.equals(stamp2)) . . . // same
   Contents?
```

- This is different from the == operator which compares the two references



```
if (stamp1 == stamp2) . . . // same
   Objects?
```

Overriding the equals method

- The Object class specifies the type of parameter as

```
public class Stamp
{
  private String color;
  private int value;
  . . .
  public boolean equals(Object
   otherObject)
  {
   . . .
  }
  . .
}
```

The Stamp equals method must declare the same type of parameter as the Object equals method to override it.

```
public boolean equals(Object otherObject)
{
  Stamp other = (Stamp) otherObject;
  return color.equals(other.color)
    && value == other.value;
}
```

Cast the parameter variable to the class Stamp

Shadowing / Hiding

Shadowing or hiding can occur in one of the following cases:

1. A parameter name is the same as a class or instance variable. The parameter name hides the instance or class variable.
2. A class or instance variable in the subclass has the same name as the class or instance variable in the super class.
3. A class method in the subclass has the same signature as a class method in the super class. (Overriding occurs only for instance methods).

Shadowing / Hiding

In this example, try to figure out the output of the assign method:

```
class SuperClass {
    int j = 1;
}



class Derived extends SuperClass {
     int j = 2;
     void assign(int j) {
      System.out.println(j);
      System.out.println(super.j);
      System.out.println(this.j);
      }
}
```

```
class SuperThis {

  static public void main  (String [] args) {
      Derived d1 = new Derived ();

      d1.assign(4);
  }
}
```

Shadowing / Hiding

```
class SuperClass {
    int j = 1;
}
class Derived extends SuperClass {
     int j = 2;
     void assign(int j) {
     System.out.println(j);


    System.out.println(super.j);


    System.out.println(this.j);
    }
}
```

j is the parameter which is 4

super.j is the SuperClass instance variable which is 1

this.j is the Drived class instance variable which is 2