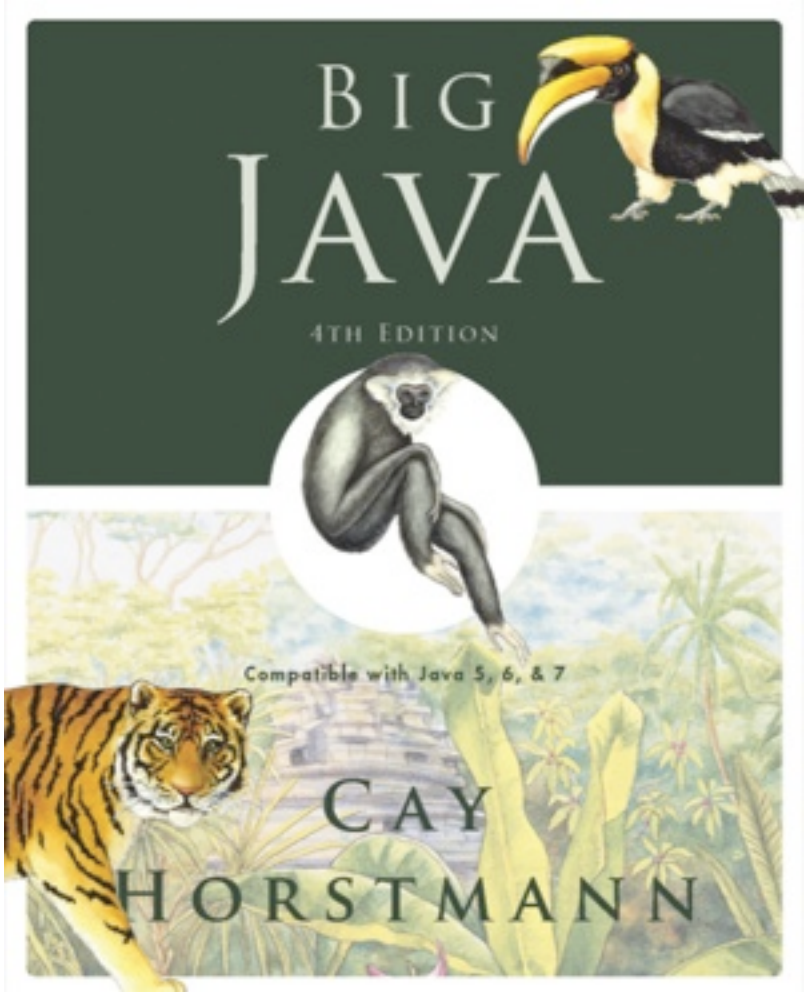
ICOM 4015: Advanced Programming

Lecture 3

Reading: Chapter Three: Implementing Classes



Chapter Three - Implementing Classes

Chapter Goals

- To become familiar with the process of implementing classes
- To be able to implement simple methods
- To understand the purpose and use of constructors
- To understand how to access instance variables and local variables
- To be able to write javadoc comments

G To implement classes for drawing graphical shapes

Key Concepts

- The Elements of a Java Class Declaration
- Modeling Objects Using Classes
- Separating the the API (WHAT) from the Implementation (HOW)

Anatomy of a Class Declaration

```
public class <NAME> {

   //Private instance variables

   //Constructors

   //Public Getters and other Accessor Instance Methods

   //Public Setters and other Modifier Instance Methods

   //Private Instance Methods

}
```

Instance Variables

- **Example:** tally counter
- Simulator statements:

```
Counter tally = new Counter();
tally.count();
tally.count();
int result = tally.getValue(); // Sets result to 2
```

Figure 1 A Tally Counter

- Each counter needs to store a variable that keeps track of how many times the counter has been advanced

Instance Variables

- **Instance variables** store the data of an object
- **Instance of a class:** an object of the class
- The class declaration specifies the instance variables:

```
public class Counter
{
   private int value;
   …
}
```

Instance Variables

- An instance variable declaration consists of the following parts:
  - *access specifier (such as* `public`*)*
  - *type of variable (such as* `int`*)*
  - *name of variable (such as* `value`*)*
- Each object of a class has its own set of instance variables
- You should declare all instance variables as private

Instance Variables

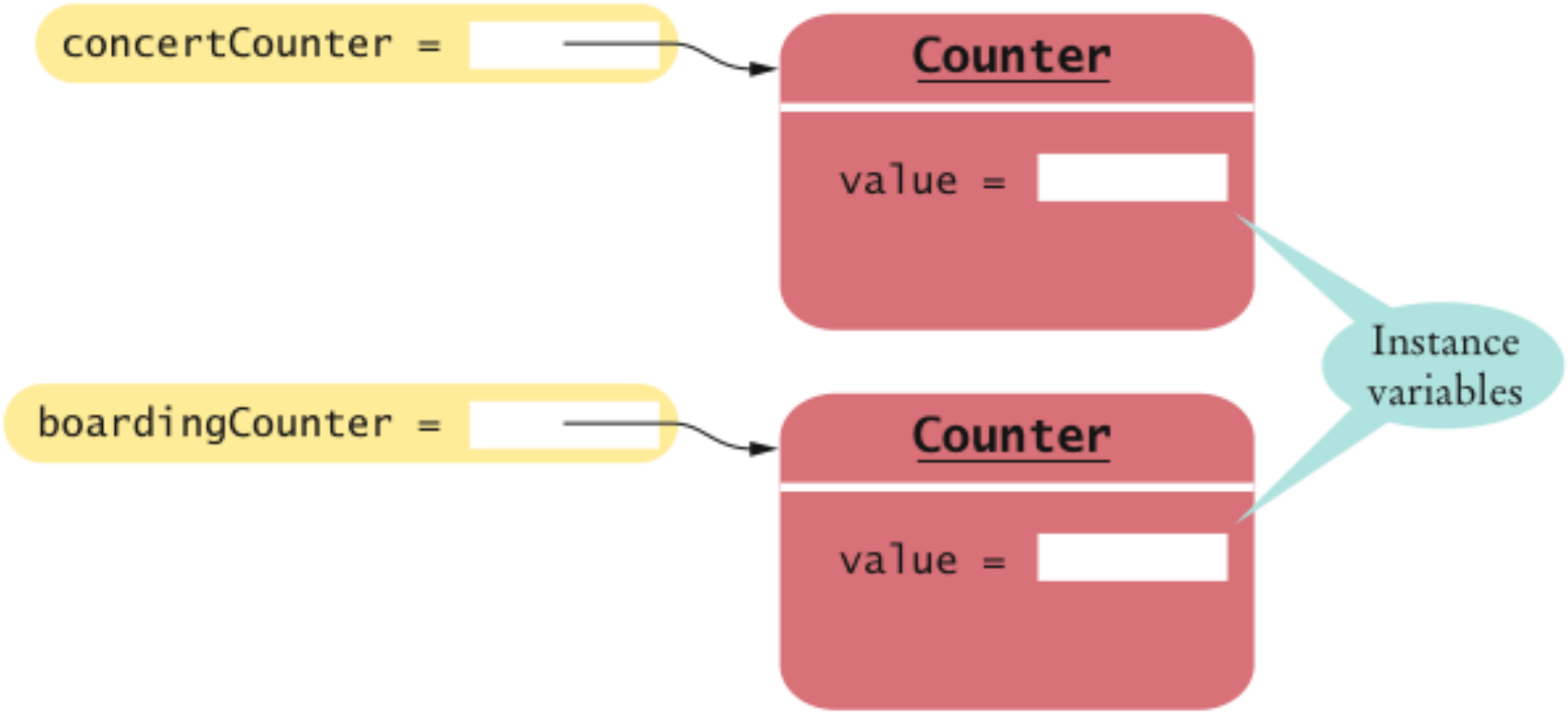


Figure 2 Instance Variables

Syntax 3.1 Instance Variable Declaration

Syntax

```
accessSpecifier class ClassName
{
   accessSpecifier typeName variableName;
   . . .
}
```

Example

```
public class Counter
{
   private int value;
   . . .
}
```

Instance variables should always be private.

Each object of this class has a separate copy of this instance variable.

Type of the variable

Accessing Instance Variables

- The `count` method advances the counter value by 1:

```
public void count()
{
   value = value + 1;
}
```

- The `getValue` method returns the current value:

```
public int getValue()
{
   return value;
}
```

- Private instance variables can only be accessed by methods of the same class

The Complete `Counter` Class

```
public class Counter
{
   // private instance variables
   private int value;

   // Constructors
   public Counter()
   {
      value= 0;
   }

   public void count() {
   {
      value = value + 1;
   }
   public int getValue() {
      return value;
   }
}
```

Self Check 3.1

Supply the body of a method `public void reset()` that resets the counter back to zero.

Answer:

```
public void reset()
{
   value = 0;
}
```

Self Check 3.2

Suppose you use a class `Clock` with private instance variables `hours` and `minutes`. How can you access these variables in your program?

Answer: You can only access them by invoking the methods of the `Clock` class.

Encapsulation: Separation of API from Its Implementation

- **Encapsulation** is the process of hiding object data and providing methods for data access
- To encapsulate data, declare instance variables as `private` and declare `public` methods that access the variables
- Encapsulation allows a programmer to use a class without having to know its implementation
- Information hiding makes it simpler for the implementer of a class to change implementations without affecting users of the API

Every Problem in Computer Science can be Solved …
by Another Level of Indirection

-David John Wheeler

Syntax 3.2 Class Declaration

Syntax

accessSpecifier `class` *ClassName*
{
    instance variables
    constructors
    methods
}

Example

```
public class Counter
{
   private int value;

   public Counter(double initialValue) { value = initialValue; }

   public void count() { value = value + 1; }
   public int getValue() { return value; }
}
```

Public interface

Private implementation

Self Check 3.3

Consider the `Counter` class. A counter's value starts at 0 and is advanced by the `count` method, so it should never be negative. Suppose you found a negative `value` variable during testing. Where would you look for the error?

Answer: In one of the methods of the `Counter` class.

Self Check 3.4

In Chapters 1 and 2, you used `System.out` as a black box to cause output to appear on the screen. Who designed and implemented `System.out`?

Answer: The programmers who designed and implemented the Java library.

Self Check 3.5

Suppose you are working in a company that produces personal finance software. You are asked to design and implement a class for representing bank accounts. Who will be the users of your class?

Answer: Other programmers who work on the personal finance application.

Specifying the Public Interface of a Class

Behavior of bank account (abstraction):

- deposit money
- withdraw money
- get balance

Specifying the Public Interface of a Class: Methods

- Methods of `BankAccount` class:
  - `deposit`
  - `withdraw`
  - `getBalance`

- We want to support method calls such as the following:

```
harrysChecking.deposit(2000);
harrysChecking.withdraw(500);
System.out.println(harrysChecking.getBalance());
```

Specifying the Public Interface of a Class: Constructor Declaration

- A constructor initializes the instance variables
- Constructor name = class name

```
public BankAccount()
{
   // body--filled in later
}
```

- Constructor body is executed when new object is created
- Statements in constructor body will set the internal data of the object that is being constructed
- All constructors of a class have the same name
- Compiler can tell constructors apart because they take different parameters

Specifying the Public Interface of a Class: Method Declaration

Parts of a Method Declaration

- access specifier (such as `public`)
- return type (such as `String` or `void`)
- method name (such as `deposit`)
- list of parameters (`double amount` for `deposit`)
- method body in `{ }`

Examples:

- `public void deposit(double amount) { . . . }`
- `public void withdraw(double amount) { . . . }`
- `public double getBalance() { . . . }`

Specifying the Public Interface of a Class: Method Header

- access specifier (such as `public`)
- return type (such as `void` or `double`)
- method name (such as `deposit`)
- list of parameter variables (such as `double amount`)

Examples:

- `public void deposit(double amount)`
- `public void withdraw(double amount)`
- `public double getBalance()`

`BankAccount` Public Interface

The public constructors and methods of a class form the *public interface* of the class:

```
public class BankAccount
{
   // private instance variables--filled in later

   // Constructors
   public BankAccount()
   {
      // body--filled in later
   }

   public BankAccount(double initialBalance)
   {
      // body--filled in later
   }
```

Continued

`BankAccount` Public Interface (cont.)

```
   // Public Instance Methods
   public void deposit(double amount)
   {
      // body--filled in later
   }
   public void withdraw(double amount)
   {
      // body--filled in later
   }
   public double getBalance()
   {
      // body--filled in later
   }
}
```

Self Check 3.6

How can you use the methods of the public interface to *empty* the `harrysChecking` bank account?

Answer:

```
harrysChecking.withdraw(harrysChecking.getBalance())
```

Self Check 3.7

What is wrong with this sequence of statements?

```
BankAccount harrysChecking = new BankAccount(10000);
System.out.println(harrysChecking.withdraw(500));
```

Answer: The `withdraw` method has return type `void`. It doesn't return a value. Use the `getBalance` method to obtain the balance after the withdrawal.

Self Check 3.8

Suppose you want a more powerful bank account abstraction that keeps track of an *account number* in addition to the balance. How would you change the public interface to accommodate this enhancement?

Answer: Add an `accountNumber` parameter to the constructors, and add a `getAccountNumber` method. There is no need for a `setAccountNumber` method – the account number never changes after construction.

Commenting the Public Interface

```
/**
   Withdraws money from the bank account.
   @param amount the amount to withdraw
*/
public void withdraw(double amount)
{
   //implementation filled in later
}
/**
   Gets the current balance of the bank account.
   @return the current balance
*/
public double getBalance()
{
   //implementation filled in later
}
```

Class Comment

```
/**
   A bank account has a balance that can be changed by
   deposits and withdrawals.
*/
public class BankAccount
{
   . . .
}
```

- Provide documentation comments for
  - *every class*
  - *every method*
  - *every parameter*
  - *every return value*

Javadoc Method Summary

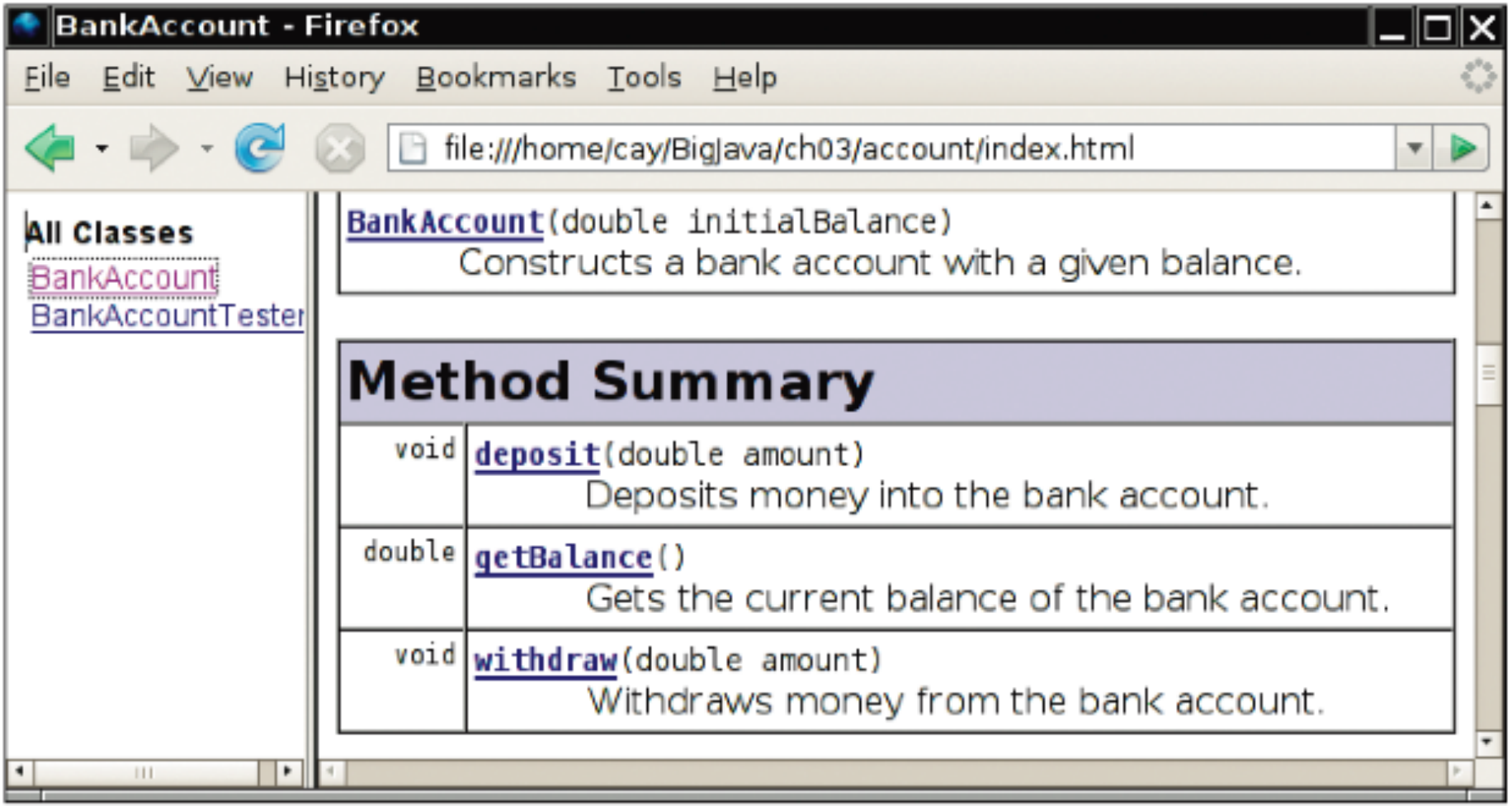


Figure 3 A Method Summary Generated by javadoc

Javadoc Method Detail

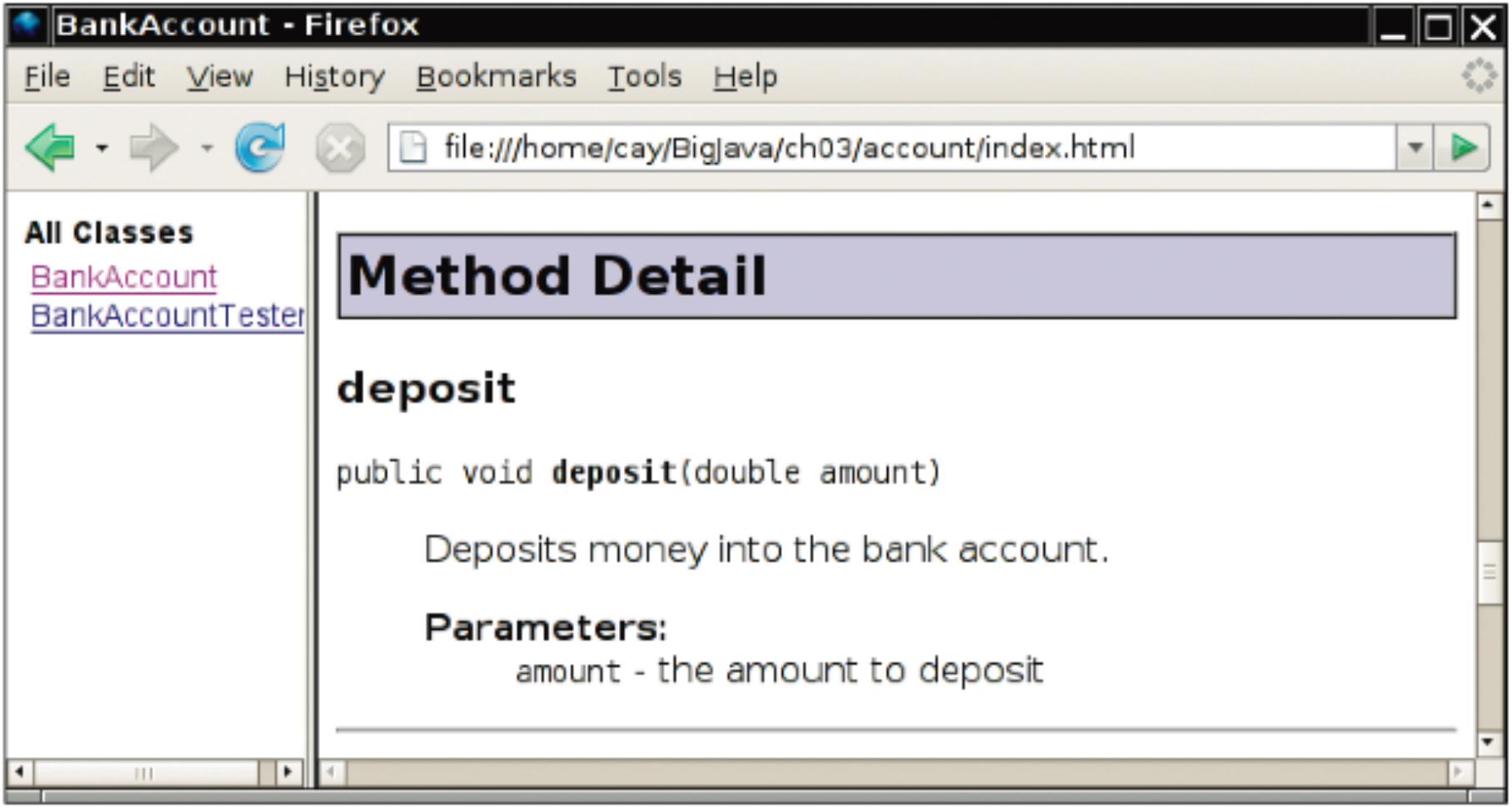


Figure 4 Method Detail Generated by javadoc

Self Check 3.9

Provide documentation comments for the `Counter` class of Section 3.1.

Answer:

```
/**
   This class models a tally counter.
*/
public class Counter
{
   private int value;
   /**
      Gets the current value of this counter.
      @return the current value
   */
   public int getValue()
   {
      return value;
   }
```

Continued

Self Check 3.9 (cont.)

```
   /**
      Advances the value of this counter by 1.
   */
   public void count()
   {
      value = value + 1;
   }
}
```

Self Check 3.10

Suppose we enhance the `BankAccount` class so that each account has an account number. Supply a documentation comment for the constructor

```
public BankAccount(int accountNumber, double initialBalance)
```

Answer:

```
/**
   Constructs a new bank account with a given initial balance.
   @param accountNumber the account number for this account
   @param initialBalance the initial balance for this account
*/
```

Self Check 3.11

Why is the following documentation comment questionable?

```
/**
   Each account has an account number.
   @return the account number of this account
*/
public int getAccountNumber()
```

Answer: The first sentence of the method description should describe the method – it is displayed in isolation in the summary table.

Implementing Constructors

- Constructors contain instructions to initialize the instance variables of an object:

```
public BankAccount()
{
   balance = 0;
}

public BankAccount(double initialBalance)
{
   balance = initialBalance;
}
```

Constructor Call Example

- Statement:

```
BankAccount harrysChecking = new BankAccount(1000);
```

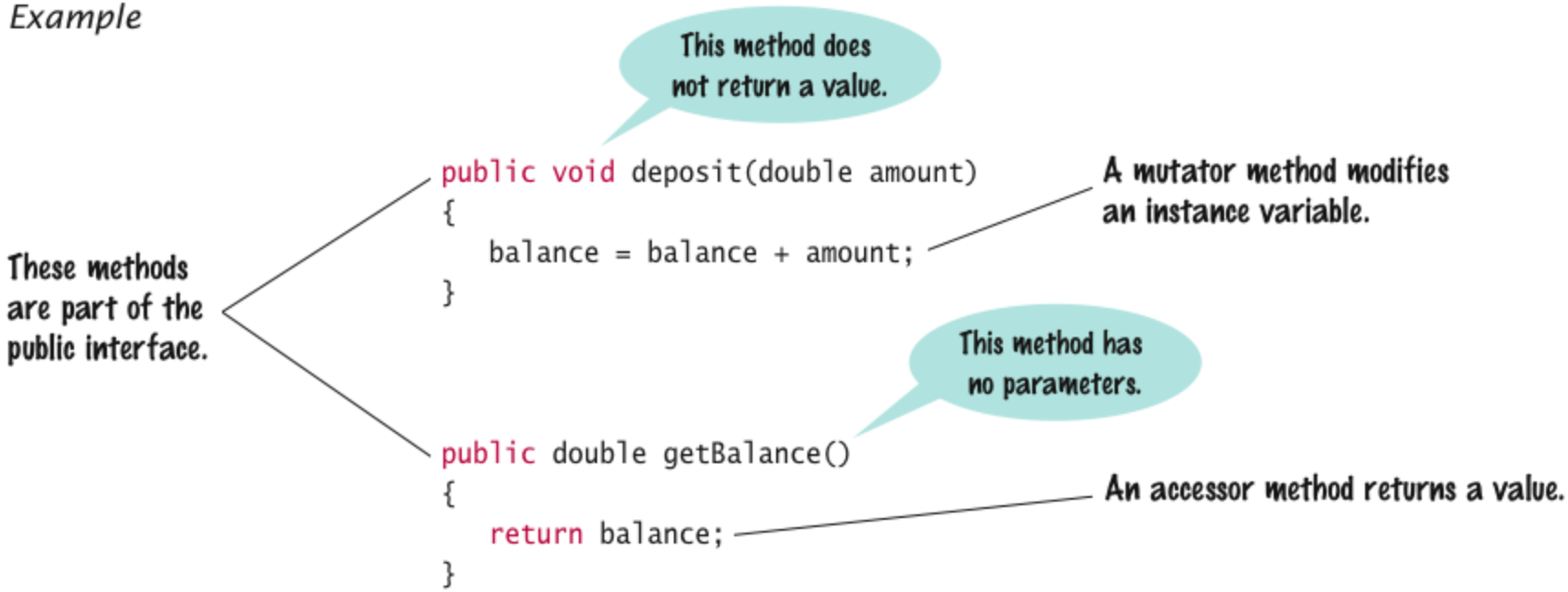
  - *Create a new object of type* `BankAccount`
  - *Call the second constructor (because a construction parameter is supplied in the constructor call)*
  - *Set the parameter variable* `initialBalance` *to* `1000`
  - *Set the* `balance` *instance variable of the newly created object to* `initialBalance`
  - *Return an object reference, that is, the memory location of the object, as the value of the* `new` *expression*
  - *Store that object reference in the* `harrysChecking` *variable*

Syntax 3.3 Method Declaration

Syntax

```
accessSpecifier returnType methodName(parameterType parameterName, . . . )
{
   method body
}
```

Example



Implementing Methods

- `deposit` method:

```
public void deposit(double amount)
{
   balance = balance + amount;
}
```

Method Call Example

- Statement:

```
harrysChecking.deposit(500);
```

  - *Set the parameter variable* `amount` *to 500*
  - *Fetch the* `balance` *variable of the object whose location is stored in* `harrysChecking`
  - *Add the value of* `amount` *to* `balance`
  - *Store the sum in the* `balance` *instance variable, overwriting the old value*

Implementing Methods

- ```
  public void withdraw(double amount)
  {
     balance = balance - amount;
  }
  ```
- ```
  public double getBalance()
  {
     return balance;
  }
  ```

The Complete Bank Account Class Declaration

```
/**
   A bank account has a balance that can be changed by
   deposits and withdrawals.
*/
public class BankAccount
{
   private double balance;

   /**
      Constructs a bank account with a zero balance.
   */
   public BankAccount()
   {
      balance = 0;
   }

   /**
      Constructs a bank account with a given balance.
      @param initialBalance the initial balance
   */
   public BankAccount(double initialBalance)
   {
      balance = initialBalance;
   }
```

Continued

The Complete Bank Account Class Declaration

```
/**
   Deposits money into the bank account.
   @param amount the amount to deposit
*/
public void deposit(double amount)
{
   balance = balance + amount;
}
/**
   Withdraws money from the bank account.
   @param amount the amount to withdraw
*/
public void withdraw(double amount)
{
   balance = balance - amount;
}
```

Continued

The Complete Bank Account Class Declaration

```
   /**
      Gets the current balance of the bank account.
      @return the current balance
   */
   public double getBalance()
   {
      return balance;
   }
}
```

Self Check 3.12

Suppose we modify the `BankAccount` class so that each bank account has an account number. How does this change affect the instance variables?

Answer:

An instance variable

```
private int accountNumber;
```

needs to be added to the class.

Self Check 3.13

Why does the following code not succeed in robbing mom's bank account?

```
public class BankRobber
{
   public static void main(String[] args)
   {
      BankAccount momsSavings = new BankAccount(1000);
      momsSavings.balance = 0;
   }
}
```

Answer: Because the `balance` instance variable is accessed from the `main` method of `BankRobber`. The compiler will report an error because `balance` has private access in `BankAccount`.

Self Check 3.14

The `Rectangle` class has four instance variables: `x`, `y`, `width`, and `height`. Give a possible implementation of the `getWidth` method.

Answer:

```
public int getWidth()
{
   return width;
}
```

Self Check 3.15

Give a possible implementation of the `translate` method of the `Rectangle` class.

Answer: There is more than one correct answer. One possible implementation is as follows:

```
public void translate(int dx, int dy)
{
   int newx = x + dx;
   x = newx;
   int newy = y + dy;
   y = newy;
}
```

Unit Testing

- *Unit test*: Verifies that a class works correctly in isolation, outside a complete program
- To test a class, use an environment for interactive testing, or write a tester class
- *Tester class*: A class with a main method that contains statements to test another class
- Typically carries out the following steps:
  1. *Construct one or more objects of the class that is being tested*
  2. *Invoke one or more methods*
  3. *Print out one or more results*
  4. *Print the expected results*

Continued

ch03/account/BankAccountTester.java

```
/**
   A class to test the BankAccount class.
*/
public class BankAccountTester
{
   /**
      Tests the methods of the BankAccount class.
      @param args not used
   */
   public static void main(String[] args)
   {
      BankAccount harrysChecking = new BankAccount();
      harrysChecking.deposit(2000);
      harrysChecking.withdraw(500);
      System.out.println(harrysChecking.getBalance());
      System.out.println("Expected: 1500");
   }
}
```

Program Run:

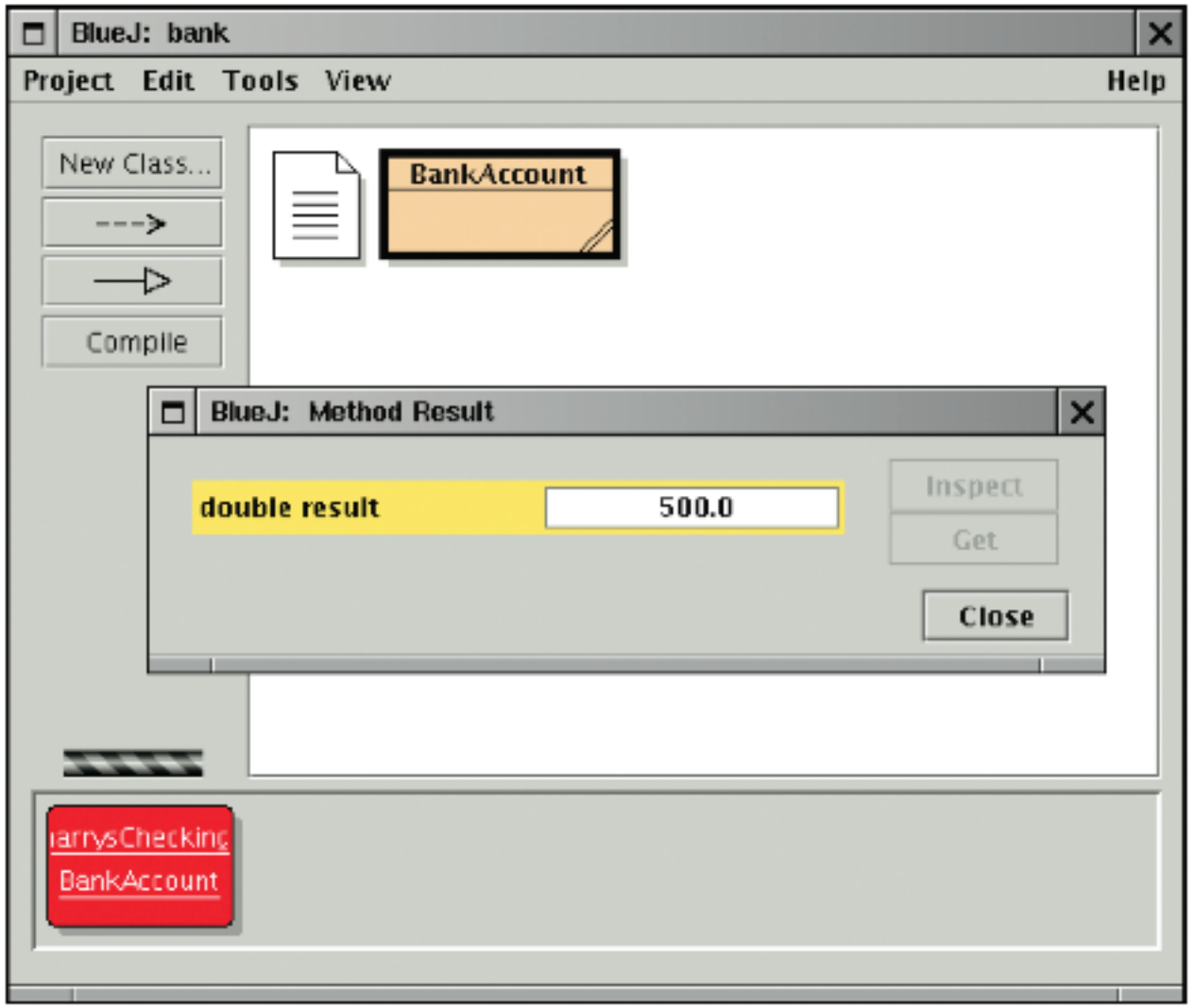
```
1500
Expected: 1500
```

Unit Testing (cont.)

- Details for building the program vary. In most environments, you need to carry out these steps:
  1. *Make a new subfolder for your program*
  2. *Make two files, one for each class*
  3. *Compile both files*
  4. *Run the test program*

Testing With BlueJ

Figure 5
The Return Value of the `getBalance` Method in BlueJ



Self Check 3.16

When you run the `BankAccountTester` program, how many objects of class `BankAccount` are constructed? How many objects of type `BankAccountTester`?

Answer: One `BankAccount` object, no `BankAccountTester` object. The purpose of the `BankAccountTester` class is merely to hold the `main` method.

Self Check 3.17

Why is the `BankAccountTester` class unnecessary in development environments that allow interactive testing, such as BlueJ?

Answer: In those environments, you can issue interactive commands to construct `BankAccount` objects, invoke methods, and display their return values.

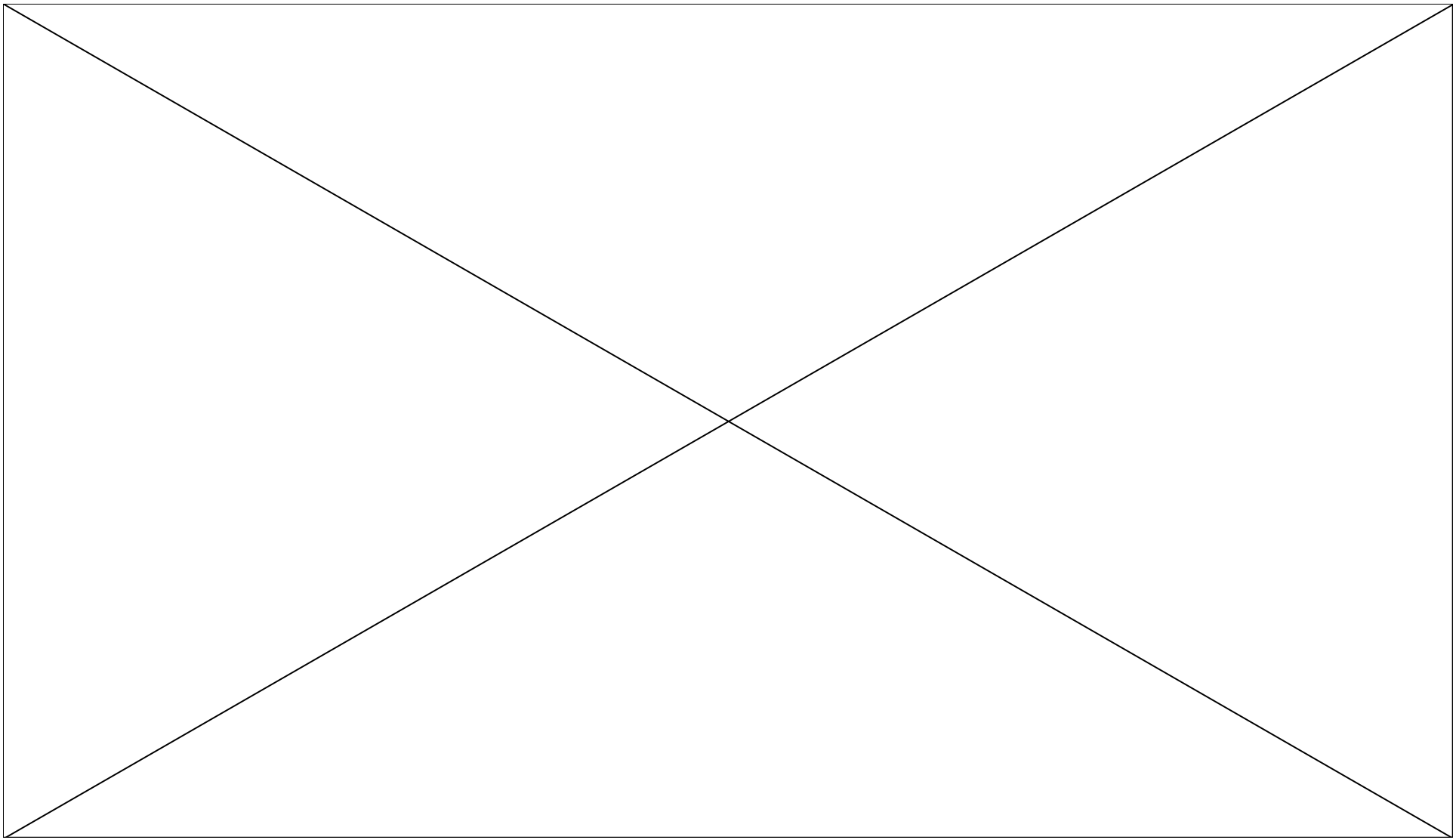
Local Variables vs. Instance Variables

- Local and parameter variables belong to a method
  - *When a method or constructor runs, its local and parameter variables come to life*
  - *When the method or constructor exits, they are removed immediately*
- Instance variables belongs to an objects, not methods
  - *When an object is constructed, its instance variables are created*
  - *The instance variables stay alive until no method uses the object any longer*
- Instance variables are initialized to a default value, but you must initialize local variables

Local Variables

- In Java, the *garbage collector* periodically reclaims objects when they are no longer used
- Instance variables are initialized to a default value, but you must initialize local variables

Animation 3.1: Lifetime of Variables



Self Check 3.18

What do local variables and parameter variables have in common? In which essential aspect do they differ?

Answer: Variables of both categories belong to methods – they come alive when the method is called, and they die when the method exits. They differ in their initialization. Parameter variables are initialized with the call values; local variables must be explicitly initialized.

Self Check 3.19

Why was it necessary to introduce the local variable `change` in the `giveChange` method? That is, why didn't the method simply end with the statement

```
return payment - purchase;
```

Answer: After computing the change due, `payment` and `purchase` were set to zero. If the method returned `payment - purchase`, it would always return zero.

Accessing Target Object Via Implicit Parameter

- The **implicit parameter** of a method is the object on which the method is invoked

```
public void deposit(double amount)
{
   balance = balance + amount;
}
```

- In the call

```
momsSavings.deposit(500)
```

The implicit parameter is `momsSavings` and the explicit parameter is `500`

- When you refer to an instance variable inside a method, it means the instance variable of the implicit parameter

Implicit Parameters and `this`

- The `this` reference denotes the implicit parameter
- `balance = balance + amount;`

  actually means

  `this.balance = this.balance + amount;`
- When you refer to an instance variable in a method, the compiler automatically applies it to the `this` reference

Implicit Parameters and `this`

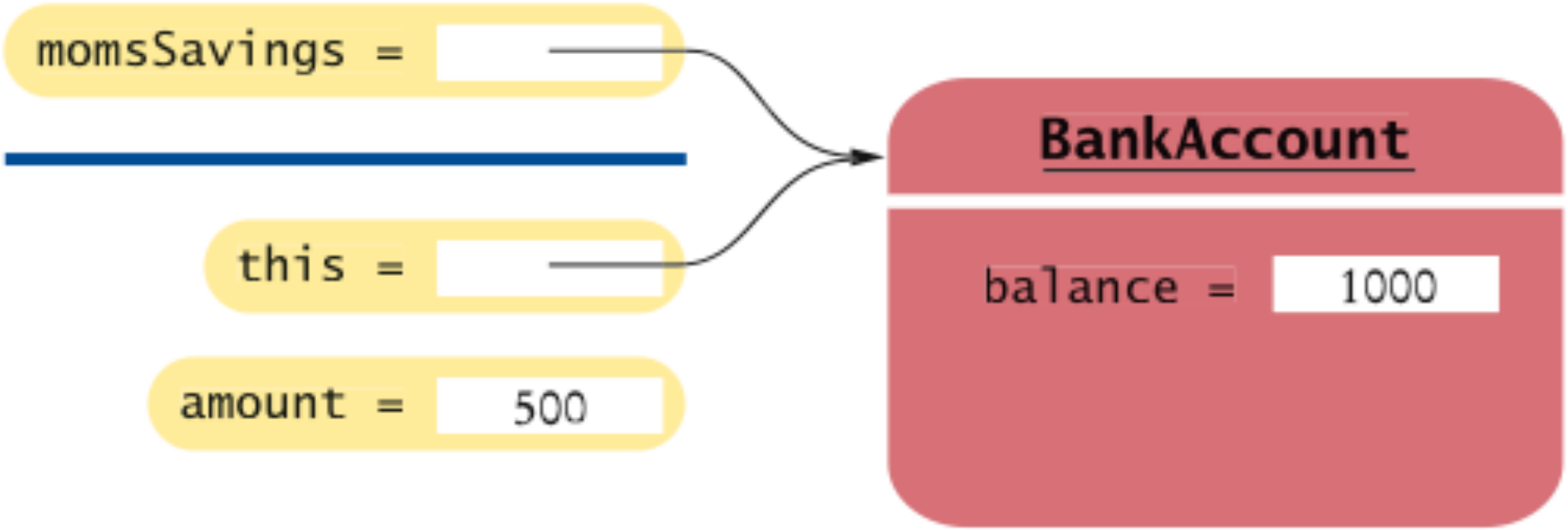


Figure 6 The Implicit Parameter of a Method Call

Implicit Parameters and `this`

- Some programmers feel that manually inserting the `this` reference before every instance variable reference makes the code clearer:

```
public BankAccount(double initialBalance)
{
   this.balance = initialBalance;
}
```

Implicit Parameters and `this`

- A method call without an implicit parameter is applied to the same object
- Example:

```
public class BankAccount
{
   . . .
   public void monthlyFee()
   {
      withdraw(10); // Withdraw $10 from this account
   }
}
```

- The implicit parameter of the `withdraw` method is the (invisible) implicit parameter of the `monthlyFee` method

Implicit Parameters and `this`

- You can use the `this` reference to make the method easier to read:

```
public class BankAccount
{
   . . .
   public void monthlyFee()
   {
      this.withdraw(10); // Withdraw $10 from this account
   }
}
```

Self Check 3.20

How many implicit and explicit parameters does the `withdraw` method of the `BankAccount` class have, and what are their names and types?

Answer: One implicit parameter, called `this`, of type `BankAccount`, and one explicit parameter, called `amount`, of type `double`.

Self Check 3.21

In the `deposit` method, what is the meaning of `this.amount`? Or, if the expression has no meaning, why not?

Answer: It is not a legal expression. `this` is of type `BankAccount` and the `BankAccount` class has no variable named `amount`. s

Self Check 3.22

How many implicit and explicit parameters does the `main` method of the `BankAccountTester` class have, and what are they called?

Answer: No implicit parameter – the main method is not ivoked on any object – and one explicit parameter, called `args`.

Shape Classes

- Good practice: Make a class for each graphical shape

```
public class Car
{
   public Car(int x, int y)
   {
      // Remember position
      . . .
   }
      public void draw(Graphics2D g2)
   {
      // Drawing instructions
      . . .
   }
}
```

Drawing Cars

- Draw two cars: one in top-left corner of window, and another in the bottom right
- Compute bottom right position, inside `paintComponent` method:

```
int x = getWidth() - 60;
int y = getHeight() - 30;
Car car2 = new Car(x, y);
```

- `getWidth` and `getHeight` are applied to object that executes `paintComponent`
- If window is resized `paintComponent` is called and car position recomputed

Continued

Drawing Cars (The Goal)

Figure 7
The Car Component Draws Two Car Shapes

Plan Complex Shapes on Graph Paper

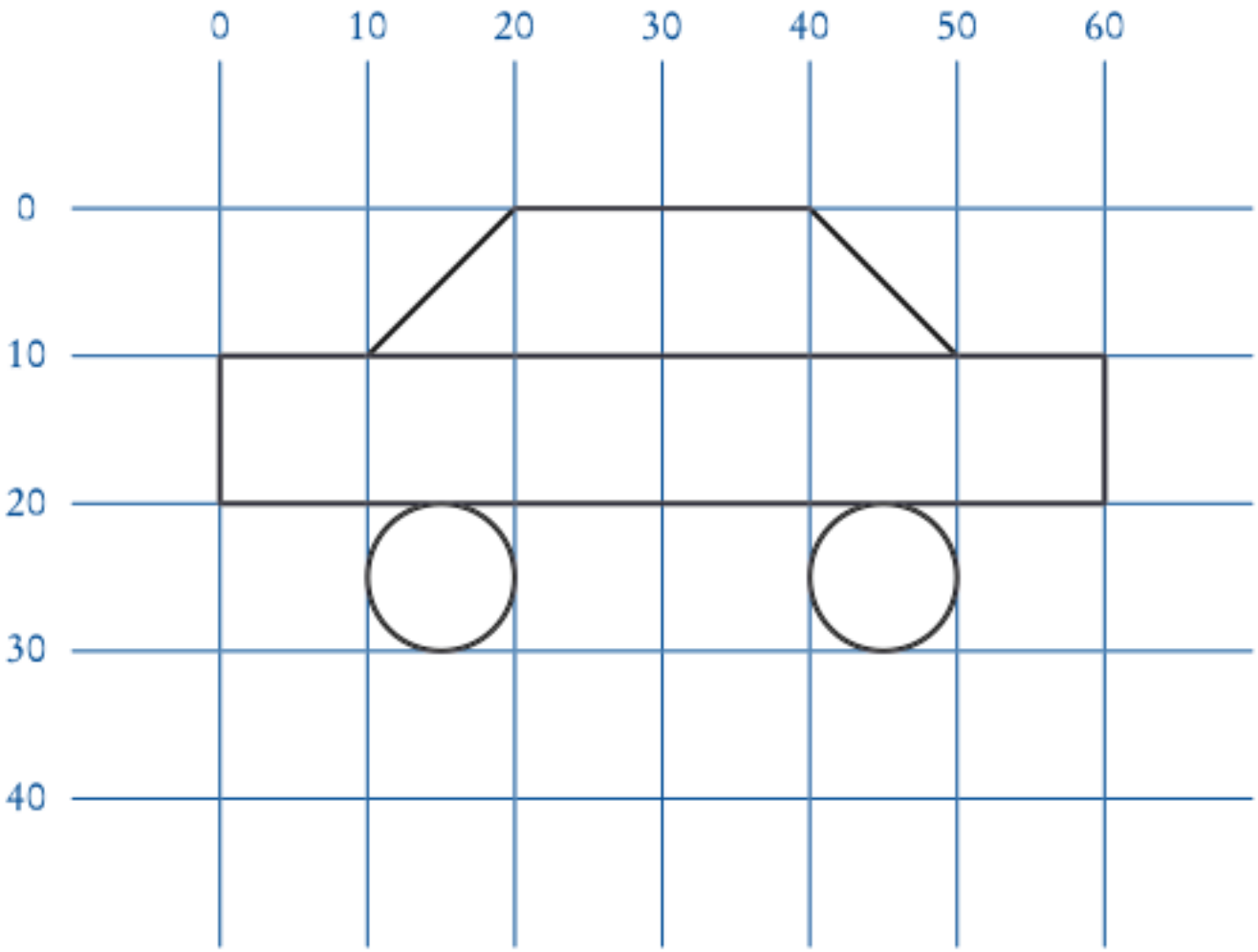


Figure 8 Using Graph Paper to Find Shape Coordinates

Classes of Car Drawing Program

- `Car`: responsible for drawing a single car
  - *Two objects of this class are constructed, one for each car*
- `CarComponent`: displays the drawing
- `CarViewer`: shows a frame that contains two `CarComponent`s

ch03/car/Car.java

```
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;

/**
   A car shape that can be positioned anywhere on the screen.
*/
public class Car
{
   private int xLeft;
   private int yTop;

   /**
      Constructs a car with a given top left corner.
      @param x the x coordinate of the top left corner
      @param y the y coordinate of the top left corner
   */
   public Car(int x, int y)
   {
      xLeft = x;
      yTop = y;
   }
```

Continued

ch03/car/Car.java (cont.)

```
/**
   Draws the car.
   @param g2 the graphics context
*/
public void draw(Graphics2D g2)
{
   Rectangle body
         = new Rectangle(xLeft, yTop + 10, 60, 10);
   Ellipse2D.Double frontTire
         = new Ellipse2D.Double(xLeft + 10, yTop + 20, 10, 10);
   Ellipse2D.Double rearTire
         = new Ellipse2D.Double(xLeft + 40, yTop + 20, 10, 10);

   // The bottom of the front windshield
   Point2D.Double r1
         = new Point2D.Double(xLeft + 10, yTop + 10);
   // The front of the roof
   Point2D.Double r2
         = new Point2D.Double(xLeft + 20, yTop);
   // The rear of the roof
   Point2D.Double r3
         = new Point2D.Double(xLeft + 40, yTop);
```

Continued

ch03/car/Car.java (cont.)

```
        // The bottom of the rear windshield
        Point2D.Double r4
              = new Point2D.Double(xLeft + 50, yTop + 10);

        Line2D.Double frontWindshield
              = new Line2D.Double(r1, r2);
        Line2D.Double roofTop
              = new Line2D.Double(r2, r3);
        Line2D.Double rearWindshield
              = new Line2D.Double(r3, r4);

        g2.draw(body);
        g2.draw(frontTire);
        g2.draw(rearTire);
        g2.draw(frontWindshield);
        g2.draw(roofTop);
        g2.draw(rearWindshield);
    }
}
```

ch03/car/CarComponent.java

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JComponent;

/**
   This component draws two car shapes.
*/
public class CarComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      Graphics2D g2 = (Graphics2D) g;

      Car car1 = new Car(0, 0);

      int x = getWidth() - 60;
      int y = getHeight() - 30;

      Car car2 = new Car(x, y);

      car1.draw(g2);
      car2.draw(g2);
   }
}
```

ch03/car/CarViewer.java

```
import javax.swing.JFrame;

public class CarViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      frame.setSize(300, 400);
      frame.setTitle("Two cars");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      CarComponent component = new CarComponent();
      frame.add(component);

      frame.setVisible(true);
   }
}
```

Self Check 3.23

Which class needs to be modified to have the two cars positioned next to each other?

Answer: `CarComponent`

Self Check 3.24

Which class needs to be modified to have the car tires painted in black, and what modification do you need to make?

Answer: In the `draw` method of the `Car` class, call

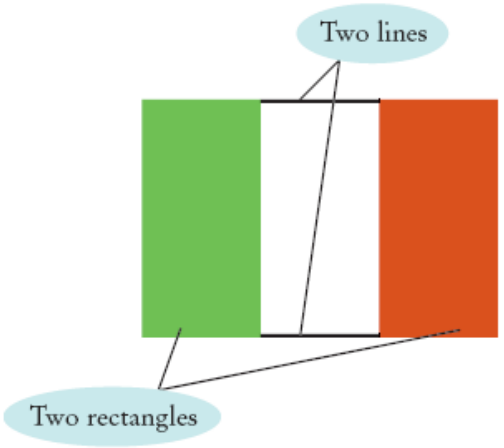
```
g2.fill(frontTire);
g2.fill(rearTire);
```

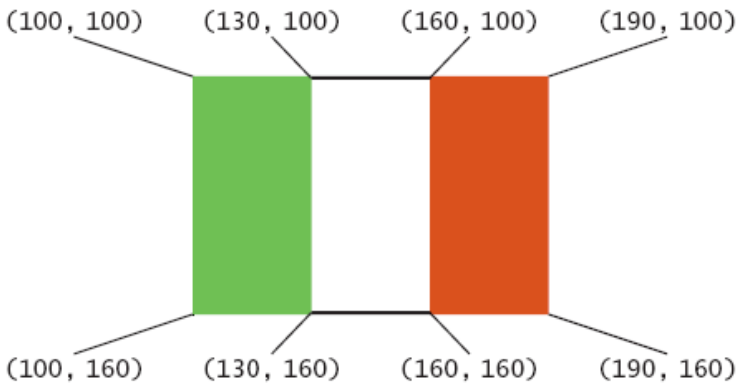
Self Check 3.25

How do you make the cars twice as big?

Answer: Double all measurements in the `draw` method of the `Car` class.

Drawing Graphical Shapes






```
Rectangle leftRectangle = new Rectangle(100, 100, 30, 60);
Rectangle rightRectangle = new Rectangle(160, 100, 30, 60);
Line2D.Double topLine = new Line2D.Double(130, 100, 160, 100);
Line2D.Double bottomLine = new Line2D.Double(130, 160, 160, 160);
```