ICOM 4015: Advanced Programming

Lecture 8

Chapter Eight: Designing Classes

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Chapter Eight: Designing Classes

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Chapter Goals

- To learn how to choose appropriate classes to implement
- To understand the concepts of cohesion and coupling
- To minimize the use of side effects
- To document the responsibilities of methods and their callers with preconditions and postconditions
- To understand the difference between instance methods and static methods
- To introduce the concept of static fields

Contined

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Chapter Goals (cont.)

- To understand the scope rules for local variables and instance fields
- To learn about packages
- To learn about unit testing frameworks

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Choosing Classes

- A class represents a single concept from the problem domain
- Name for a class should be a noun that describes concept
- Concepts from mathematics:

```
Point
Rectangle
Ellipse
```

- Concepts from real life:

```
BankAccount
CashRegister
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Choosing Classes

- Actors (end in -er, -or) – objects do some kinds of work for you

  ```
  Scanner
  Random // better name: RandomNumberGenerator
  ```

- Utility classes – no objects, only static methods and constants

  ```
  Math
  ```

- Program starters: only have a `main` method
- Don't turn actions into classes:

  `Paycheck` is a better name than `ComputePaycheck`

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.1

What is the rule of thumb for finding classes?

Answer: Look for nouns in the problem description.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.2

Your job is to write a program that plays chess. Might `ChessBoard` be an appropriate class? How about `MovePiece`?

Answer: Yes (`ChessBoard`) and no (`MovePiece`).

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Cohesion

- A class should represent a single concept
- The public interface of a class is cohesive if all of its features are related to the concept that the class represents
- This class lacks cohesion:

```
public class CashRegister
{
   public void enterPayment(int dollars, int quarters,
      int dimes, int nickels, int pennies)
   . . .
   public static final double NICKEL_VALUE = 0.05;
   public static final double DIME_VALUE = 0.1;
   public static final double QUARTER_VALUE = 0.25;
   . . .
}
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Cohesion

`CashRegister`, as described above, involves two concepts: *cash register* and *coin*

Solution: Make two classes:

```
public class Coin
{
   public Coin(double aValue, String aName) { . . . }
   public double getValue() { . . . }
   . . .
}
public class CashRegister
{
   public void enterPayment(int coinCount, Coin coinType)
      { . . . }
   . . .
}
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Coupling

- A class *depends* on another if it uses objects of that class
- `CashRegister` depends on `Coin` to determine the value of the payment
- `Coin` does not depend on `CashRegister`
- High Coupling = many class dependencies
- Minimize coupling to minimize the impact of interface changes
- To visualize relationships draw class diagrams
- UML: Unified Modeling Language. Notation for object-oriented analysis and design

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Coupling

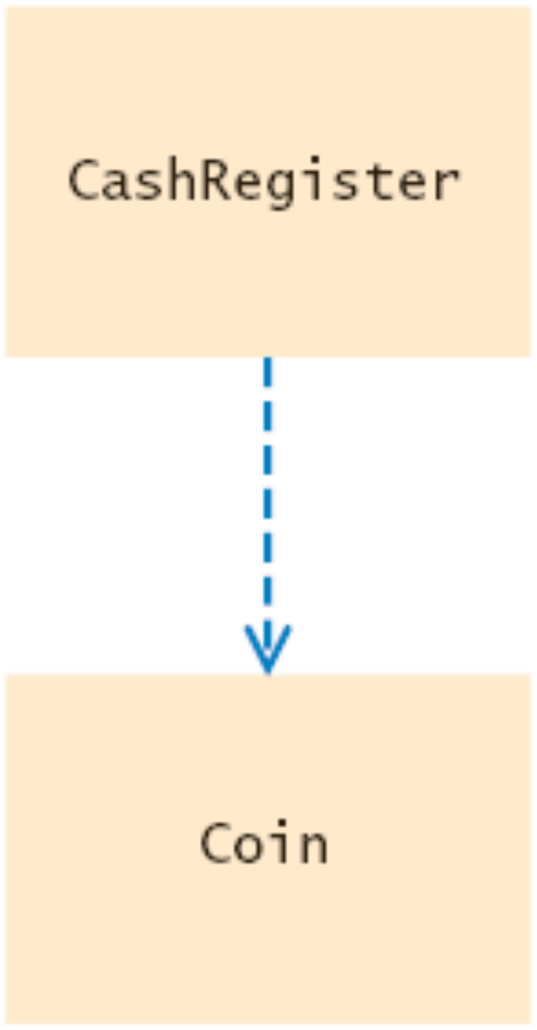


Figure 1
Dependency Relationship Between the CashRegister and Coin Classes

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

High and Low Coupling Between Classes

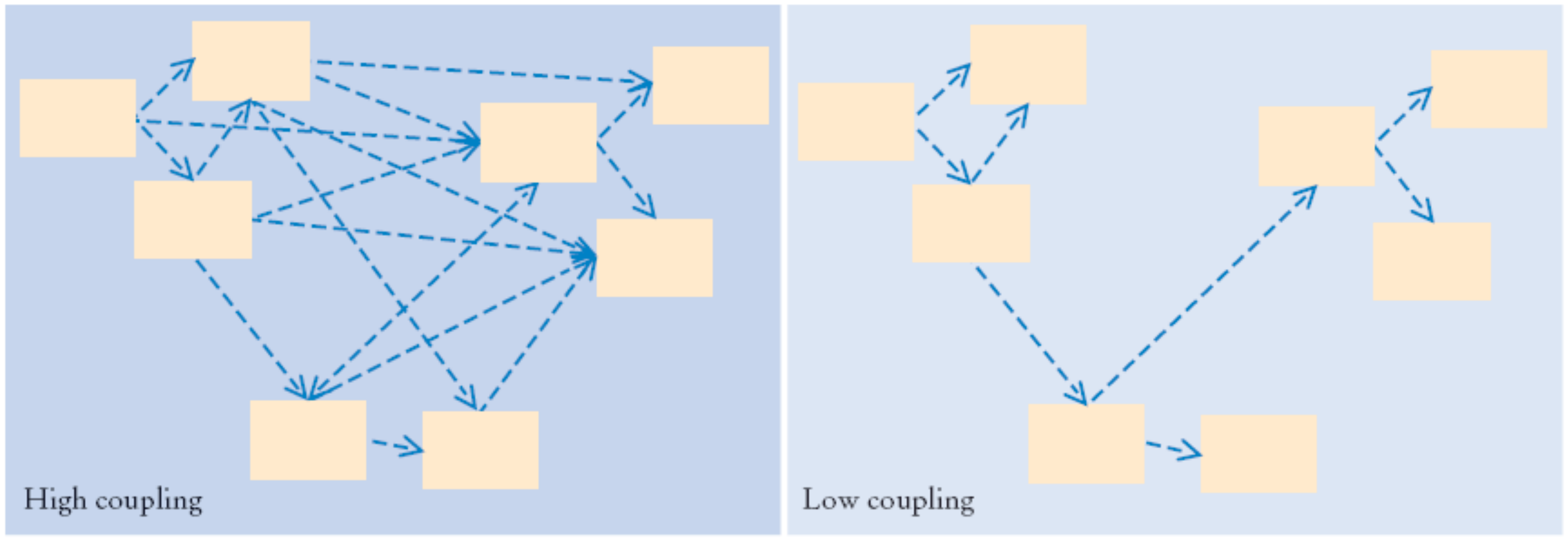


Figure 2 High and Low Coupling Between Classes

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.3

Why is the `CashRegister` class from Chapter 4 not cohesive?

Answer: Some of its features deal with payments, others with coin values.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.4

Why does the `Coin` class not depend on the `CashRegister` class?

> **Answer:** None of the coin operations require the `CashRegister` class.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.5

Why should coupling be minimized between classes?

Answer: If a class doesn't depend on another, it is not affected by interface changes in the other class.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Accessors, Mutators and Immutable Classes

- Accessor: does not change the state of the implicit parameter

```
double balance = account.getBalance();
```

- Mutator: modifies the object on which it is invoked

```
account.deposit(1000);
```

- Immutable class: has no mutator methods (e.g., `String`)

```
String name = "John Q. Public";
String uppercased = name.toUpperCase(); // name is not
   changed
```

- It is safe to give out references to objects of immutable classes; no code can modify the object at an unexpected time

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.6

Is the `substring` method of the `String` class an accessor or a mutator?

Answer: It is an accessor – calling `substring` doesn't modify the string on which the method is invoked. In fact, all methods of the `String` class are accessors.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.7

Is the `Rectangle` class immutable?

Answer: No – `translate` is a mutator.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Side Effects

- Side effect of a method: any externally observable data modification

```
public void transfer(double amount, BankAccount other)
{
   balance = balance - amount;
   other.balance = other.balance + amount; // Modifies
      explicit parameter
}
```

- Updating explicit parameter can be surprising to programmers; it is best to avoid it if possible

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Side Effects

- Another example of a side effect is output

  ```
  public void printBalance() // Not recommended
  {
     System.out.println("The balance is now $" +
  balance); }
  ```

  Bad idea: message is in English, and relies on `System.out`
  It is best to decouple input/output from the actual work of your classes

- You should minimize side effects that go beyond modification of the implicit parameter

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.8

If a refers to a bank account, then the call `a.deposit(100)` modifies the bank account object. Is that a side effect?

> **Answer:** No – a side effect of a method is any change outside the implicit parameter.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.9

Consider the `DataSet` class of Chapter 6. Suppose we add a method

```
void read(Scanner in)
{
   while (in.hasNextDouble())
      add(in.nextDouble());
}
```

Does this method have a side effect?

Answer: Yes – the method affects the state of the Scanner parameter.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

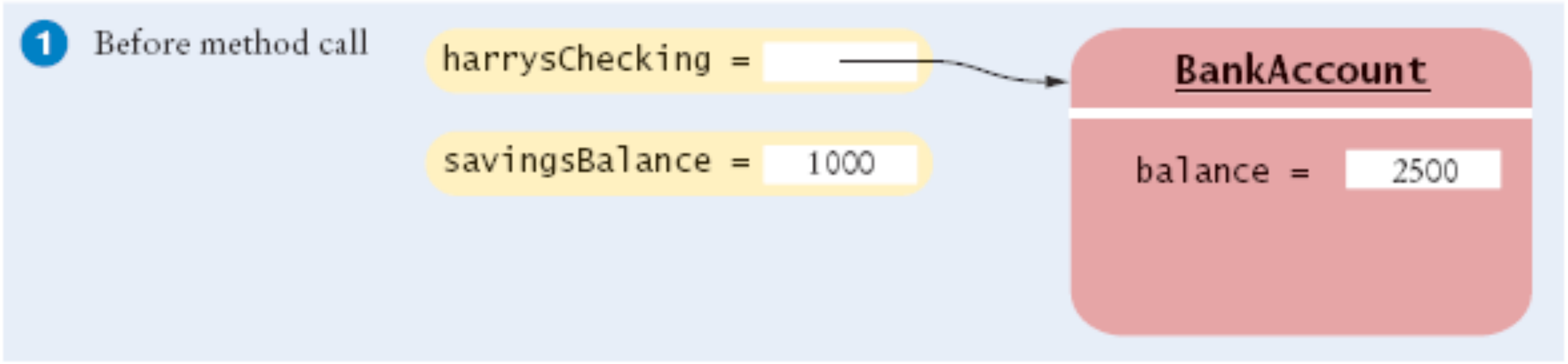
Common Error: Trying to Modify Primitive Type Parameters

-
  ```
  void transfer(double amount, double otherBalance)
   {
      balance = balance - amount;
      otherBalance = otherBalance + amount;
   }
  ```
- Won't work
- Scenario:
  ```
  double savingsBalance = 1000;
  harrysChecking.transfer(500, savingsBalance);
  System.out.println(savingsBalance);
  ```
- In Java, a method can never change parameters of primitive type

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Common Error: Trying to Modify Primitive Type Parameters

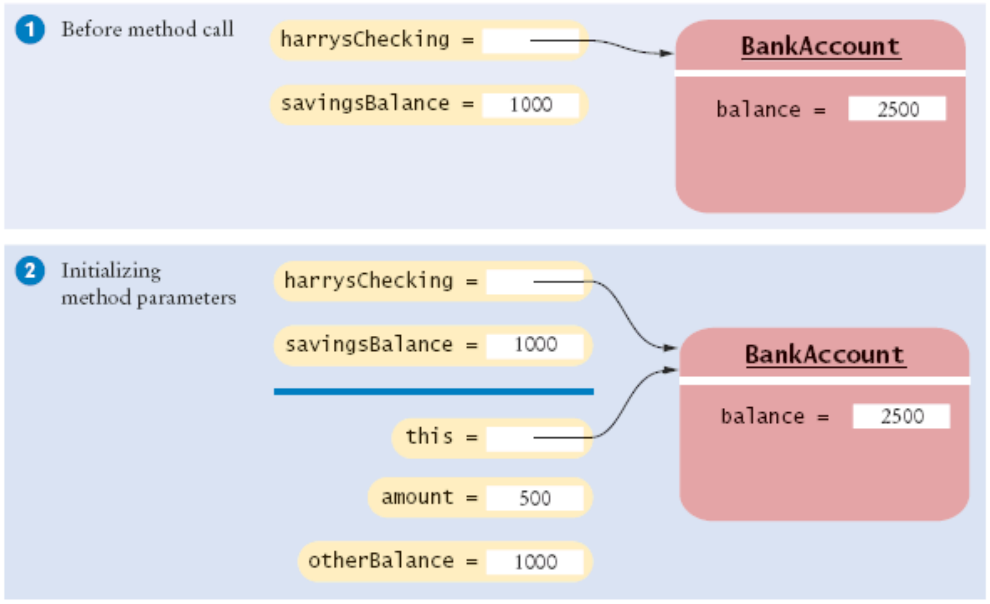
```
double savingsBalance = 1000;
harrysChecking.transfer(500, savingsBalance); 1
System.out.println(savingsBalance);
...
void transfer(double amount, double otherBalance)
{
   balance = balance - amount;
   otherBalance = otherBalance + amount;
}
```



Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Common Error: Trying to Modify Primitive Type Parameters

```
double savingsBalance = 1000;
harrysChecking.transfer(500, savingsBalance);   ❶
System.out.println(savingsBalance);
...
void transfer(double amount, double otherBalance)   ❷
{
   balance = balance - amount;
   otherBalance = otherBalance + amount;
}
```



Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

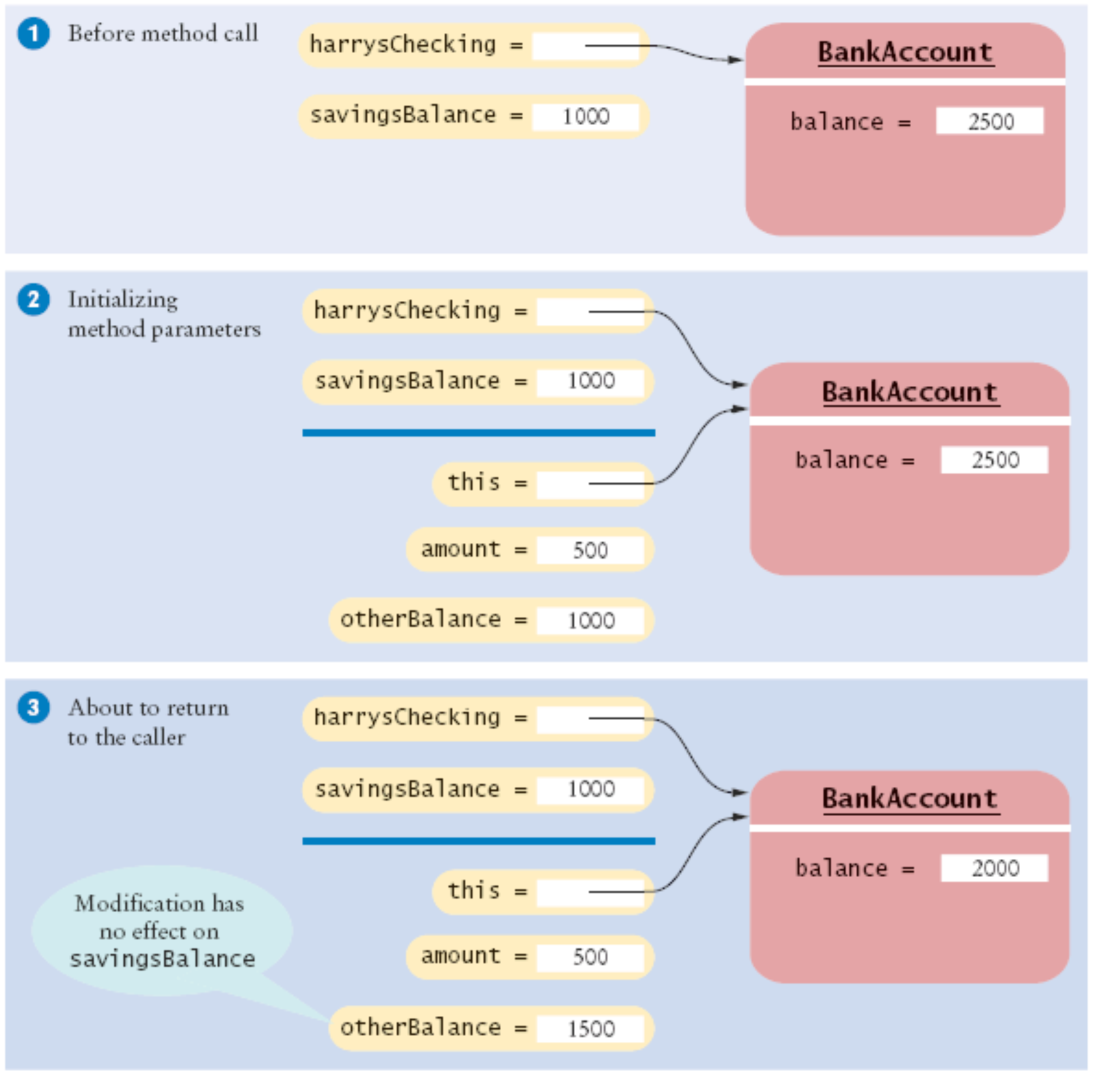
Common Error: Trying to Modify Primitive Type Parameters

```
double savingsBalance = 1000;
harrysChecking.transfer(500, savingsBalance); ❶
System.out.println(savingsBalance);
...
void transfer(double amount, double otherBalance) ❷
{
   balance = balance - amount;
   otherBalance = otherBalance + amount;
} ❸
```

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Common Error: Trying to Modify Primitive Type Parameters (cont.)



Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Common Error: Trying to Modify Primitive Type Parameters

```
double savingsBalance = 1000;
harrysChecking.transfer(500, savingsBalance); ❶
System.out.println(savingsBalance); ❹
...
void transfer(double amount, double otherBalance) ❷
{
   balance = balance - amount;
   otherBalance = otherBalance + amount;
} ❸
```

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Common Error: Trying to Modify Primitive Type Parameters (cont.)

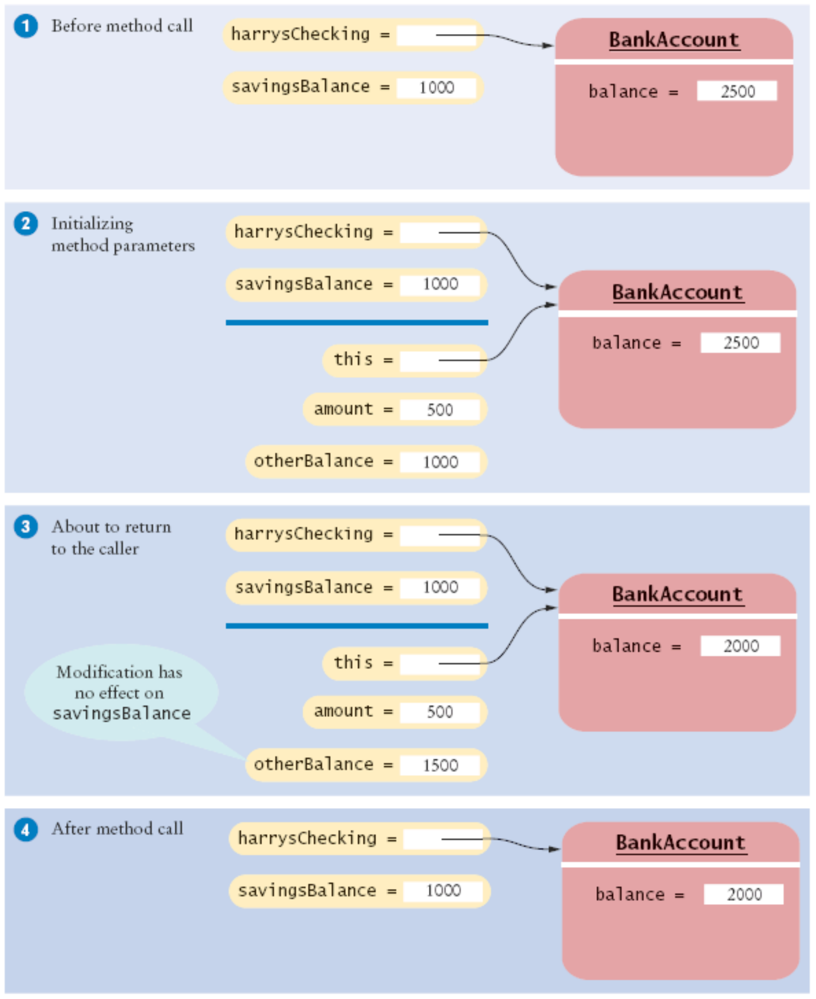
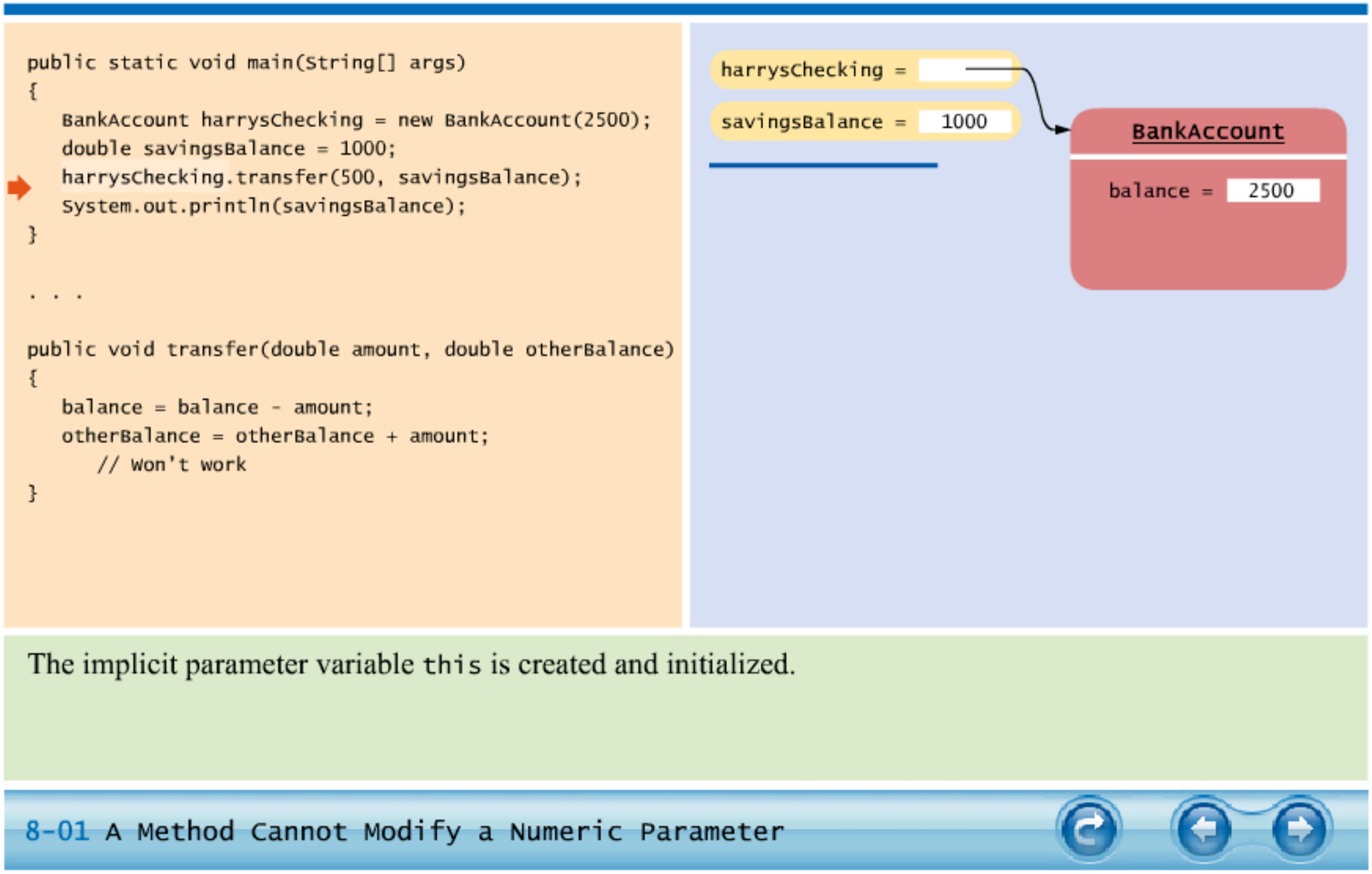


Figure 3 Modifying a Numeric Parameter Has No Effect on Caller

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Animation 8.1 – Trying to Modify Primitive Type Parameters



```
public static void main(String[] args)
{
   BankAccount harrysChecking = new BankAccount(2500);
   double savingsBalance = 1000;
   harrysChecking.transfer(500, savingsBalance);
   System.out.println(savingsBalance);
}

. . .

public void transfer(double amount, double otherBalance)
{
   balance = balance - amount;
   otherBalance = otherBalance + amount;
      // Won't work
}
```

The implicit parameter variable this is created and initialized.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Call by Value and Call by Reference

- Call by value: Method parameters are copied into the parameter variables when a method starts
- Call by reference: Methods can modify parameters
- Java has call by value
- A method can change state of object reference parameters, but cannot replace an object reference with another

Continued

Copyright © 2008 by John Wiley & Sons. All rights reserved.

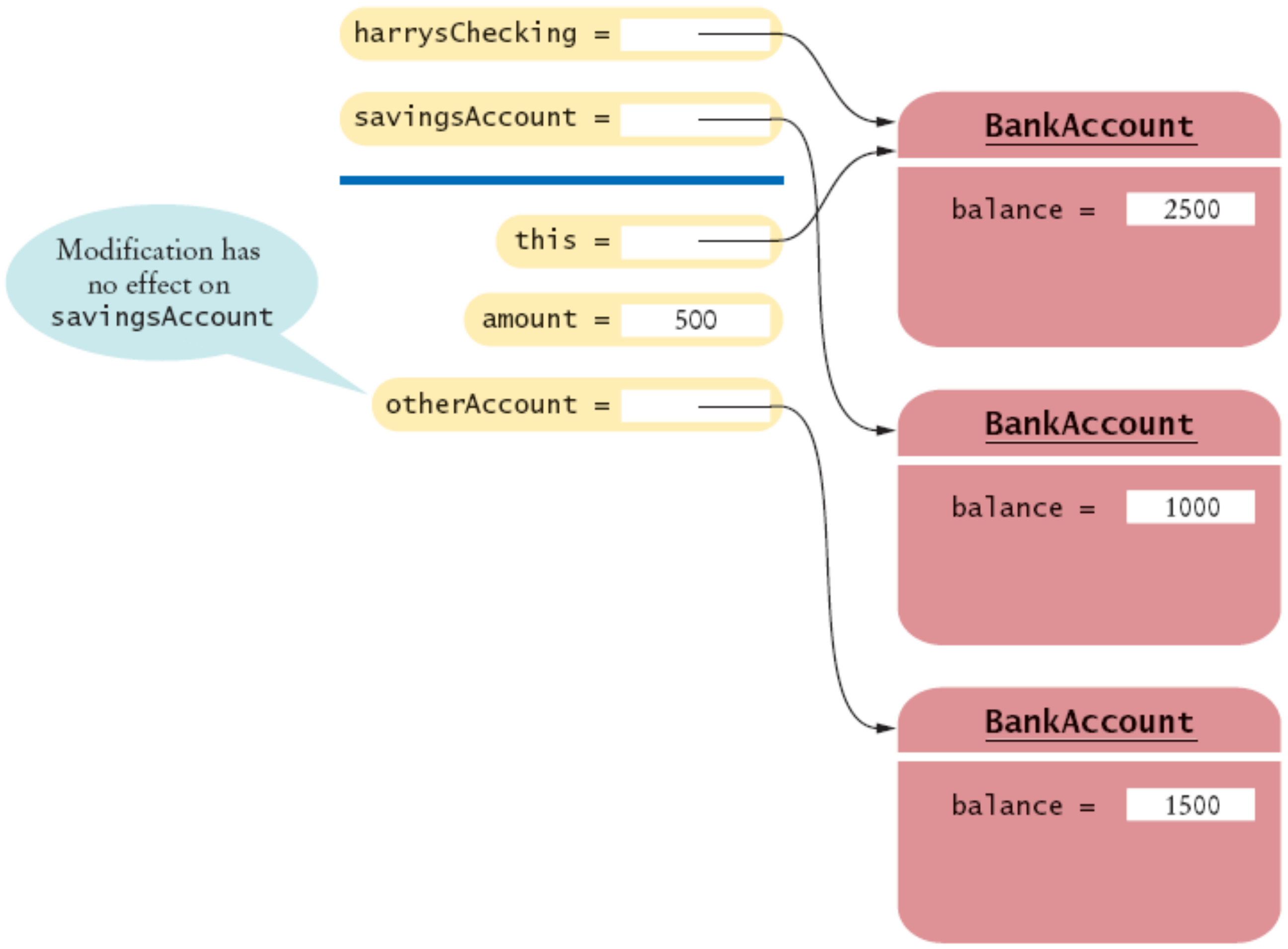
Call by Value and Call by Reference (cont.)

```
public class BankAccount
  {
     public void transfer(double amount, BankAccount
        otherAccount)
     {
        balance = balance - amount;
        double newBalance = otherAccount.balance + amount;
        otherAccount = new BankAccount(newBalance); //
           Won't work
     }
  }
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Call by Value Example

```
harrysChecking.transfer(500, savingsAccount);
```



Modifying an Object Reference Parameter Has No Effect on the Caller

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Preconditions

- Precondition: Requirement that the caller of a method must meet
- Publish preconditions so the caller won't call methods with bad parameters

-
  ```
  /**
     Deposits money into this account.
     @param amount the amount of money to deposit
     (Precondition: amount >= 0)
  */
  ```
- Typical use:
  - *To restrict the parameters of a method*
  - *To require that a method is only called when the object is in an appropriate state*

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Preconditions (cont.)

- If precondition is violated, method is not responsible for computing the correct result. It is free to do *anything*

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Preconditions

- Method may throw exception if precondition violated – more in Chapter 11

  ```
  if (amount < 0) throw new IllegalArgumentException();
  balance = balance + amount;
  ```

- Method doesn't have to test for precondition. (Test may be costly)

  ```
  // if this makes the balance negative, it's the caller's
     fault
  balance = balance + amount;
  ```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Preconditions

- Method can do an assertion check

```
assert amount >= 0;
balance = balance + amount;
```

- To enable assertion checking:

```
java -enableassertions MyProg
```

  You can turn assertions off after you have tested your program, so that it runs at maximum speed

- Many beginning programmers silently return to the caller

```
if (amount < 0)
   return; // Not recommended; hard to debug
balance = balance + amount;
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Syntax 8.1 Assertion

```
assert condition;
```

Example:

```
assert amount >= 0;
```

Purpose:

To assert that a condition is fulfilled. If assertion checking is enabled and the condition is false, an assertion error is thrown.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Postconditions

- Condition that is true after a method has completed
- If method call is in accordance with preconditions, it must ensure that postconditions are valid
- There are two kinds of postconditions:
  - *The return value is computed correctly*
  - *The object is in a certain state after the method call is completed*

```
/**
    Deposits money into this account.
    (Postcondition: getBalance() >= 0)
    @param amount the amount of money to deposit
    (Precondition: amount >= 0) */
```

- Don't document trivial postconditions that repeat the `@return` clause

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Postconditions (cont.)

```
amount <= getBalance() // this is the way to state a
   postcondition
amount <= balance // wrong postcondition formulation
```

- Contract: If caller fulfills precondition, method must fulfill postcondition

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.10

Why might you want to add a precondition to a method that you provide for other programmers?

Answer: Then you don't have to worry about checking for invalid values – it becomes the caller's responsibility.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.11

When you implement a method with a precondition and you notice that the caller did not fulfill the precondition, do you have to notify the caller?

Answer: No – you can take any action that is convenient for you.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Static Methods

- Every method must be in a class
- A static method is not invoked on an object
- Why write a method that does not operate on an object? Common reason: encapsulate some computation that involves only numbers. Numbers aren't objects, you can't invoke methods on them. E.g., `x.sqrt()` can never be legal in Java

-
  ```
  public class Financial
  {
     public static double percentOf(double p, double a)
     {
        return (p / 100) * a;
     }
     // More financial methods can be added here.
  }
  ```

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Static Methods (cont.)

- Call with class name instead of object:

```
double tax = Financial.percentOf(taxRate, total);
```

- `main` is static – there aren't any objects yet

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.12

Suppose Java had no static methods. Then all methods of the `Math` class would be instance methods. How would you compute the square root of *x*?

Answer:

```
Math m = new Math();
y = m.sqrt(x);
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.13

Harry turns in his homework assignment, a program that plays tic-tac-toe. His solution consists of a single class with many static methods. Why is this not an object-oriented solution?

Answer: In an object-oriented solution, the `main` method would construct objects of classes `Game`, `Player`, and the like. Most methods would be instance methods that depend on the state of these objects.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Static Fields

- A static field belongs to the class, not to any object of the class. Also called *class field*

-
  ```
  public class BankAccount
   {
      . . .
      private double balance;
      private int accountNumber;
      private static int lastAssignedNumber = 1000;
   }
  ```
- If `lastAssignedNumber` was not `static`, each instance of `BankAccount` would have its own value of `lastAssignedNumber`

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Static Fields (cont.)

-
```
public BankAccount()
 {
     // Generates next account number to be assigned
     lastAssignedNumber++; // Updates the static field
     // Assigns field to account number of this bank
        account
     accountNumber = lastAssignedNumber; // Sets the
        instance field }
```

- Minimize the use of static fields (static final fields are ok)

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Static Fields

- Three ways to initialize:
  1. *Do nothing. Field is initialized with 0 (for numbers), false (for boolean values), or null (for objects)*
  2. *Use an explicit initializer, such as*

     ```
     public class BankAccount
     {
        . . .
        private static int lastAssignedNumber = 1000;
           // Executed once,
           // when class is loaded }
     ```
  3. *Use a static initialization block*
- Static fields should always be declared as `private`

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Static Fields (cont.)

- Exception: Static constants, which may be either private or public

```
public class BankAccount
{
   . . .
   public static final double OVERDRAFT_FEE = 5; //
      Refer to it as
      // BankAccount.OVERDRAFT_FEE
}
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

A Static Field and Instance Fields

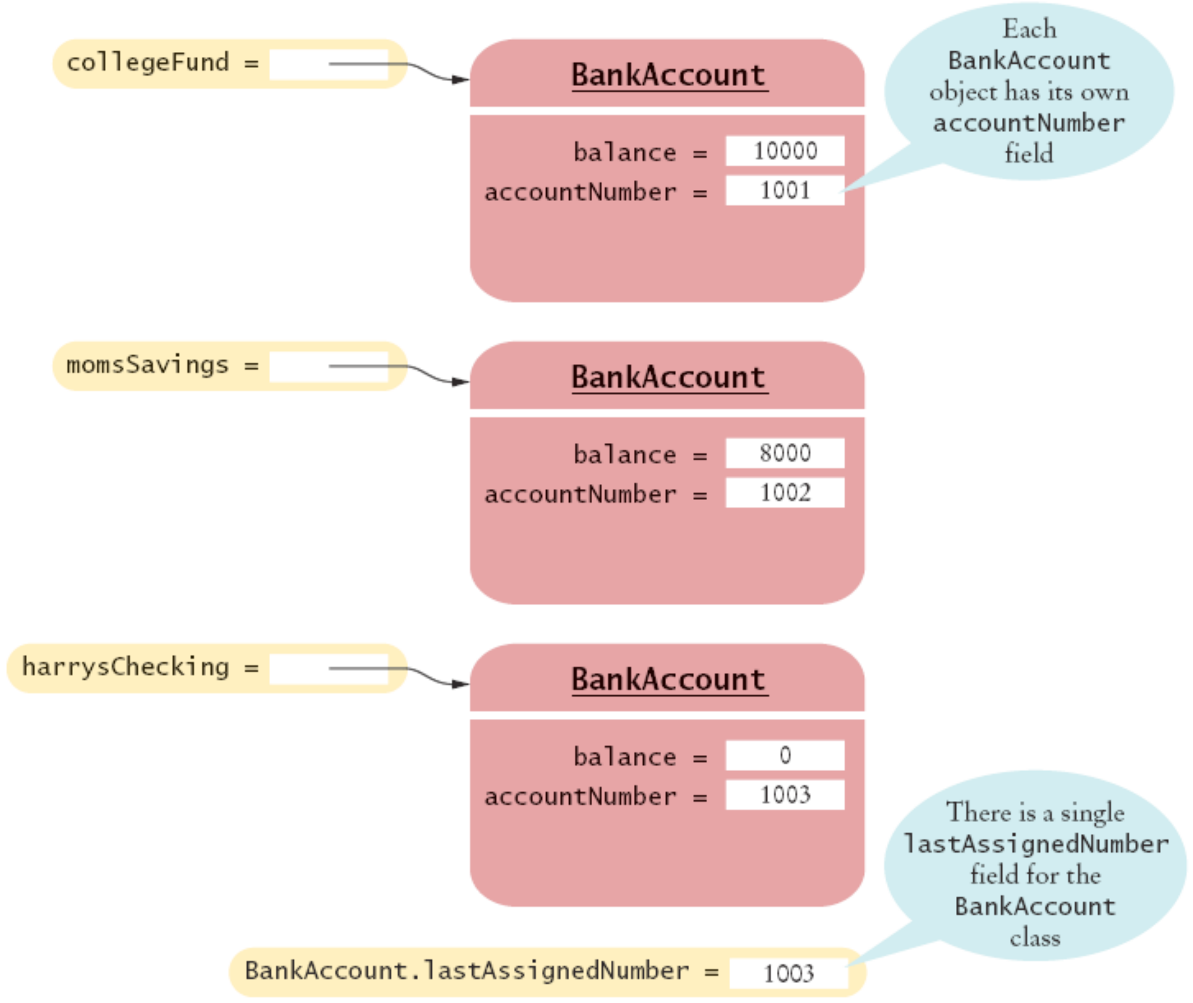


Figure 4 A Static Field and Instance Fields

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.14

Name two static fields of the `System` class.

Answer: `System.in` and `System.out`.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.15

Harry tells you that he has found a great way to avoid those pesky objects: Put all code into a single class and declare all methods and fields `static`. Then `main` can call the other static methods, and all of them can access the static fields. Will Harry's plan work? Is it a good idea?

Answer: Yes, it works. Static methods can access static fields of the same class. But it is a terrible idea. As your programming tasks get more complex, you will want to use objects and classes to organize your programs.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Scope of Local Variables

- Scope of variable: Region of program in which the variable can be accessed
- Scope of a local variable extends from its declaration to end of the block that encloses it

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Scope of Local Variables (cont.)

- Sometimes the same variable name is used in two methods:

```
public class RectangleTester
{
   public static double area(Rectangle rect)
   {
      double r = rect.getWidth() * rect.getHeight();
      return r;
   }
   public static void main(String[] args)
   {
      Rectangle r = new Rectangle(5, 10, 20, 30);
      double a = area(r);
      System.out.println(r);
   }
}
```

- These variables are independent from each other; their scopes are disjoint

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Scope of Local Variables

- Scope of a local variable cannot contain the definition of another variable with the same name

```
Rectangle r = new Rectangle(5, 10, 20, 30);
if (x >= 0)
{
   double r = Math.sqrt(x);
   // Error - can't declare another variable called r
      here
   . . .
}
```

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Scope of Local Variables (cont.)

- However, can have local variables with identical names if scopes do not overlap

```
if (x >= 0)
{
   double r = Math.sqrt(x);
   . . .
   } // Scope of r ends here
else
{
   Rectangle r = new Rectangle(5, 10, 20, 30);
   // OK - it is legal to declare another r here
   . . .
}
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Scope of Class Members

- Private members have class scope: You can access all members in any method of the class
- Must qualify public members outside scope

```
Math.sqrt
harrysChecking.getBalance
```

- Inside a method, no need to qualify fields or methods that belong to the same class

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Scope of Class Members (cont.)

- An unqualified instance field or method name refers to the `this` parameter

```
public class BankAccount
{
   public void transfer(double amount, BankAccount other)
   {
      withdraw(amount); // i.e., this.withdraw(amount);
      other.deposit(amount);
   }
   . . .
}
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Overlapping Scope

- A local variable can *shadow* a field with the same name
- Local scope wins over class scope

```
public class Coin
{
   . . .
   public double getExchangeValue(double exchangeRate)
   {
      double value; // Local variable
      . . .
      return value;
   }
   private String name;
   private double value; // Field with the same name
}
```

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Overlapping Scope (cont.)

- Access shadowed fields by qualifying them with the this reference

  ```
  value = this.value * exchangeRate;
  ```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.16

Consider the `deposit` method of the `BankAccount` class. What is the scope of the variables `amount` and `newBalance`?

Answer: The scope of `amount` is the entire `deposit` method. The scope of `newBalance` starts at the point at which the variable is defined and extends to the end of the method.

Self Check 8.17

What is the scope of the `balance` field of the `BankAccount` class?

Answer: It starts at the beginning of the class and ends at the end of the class.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Organizing Related Classes into Packages

- Package: Set of related classes
- To put classes in a package, you must place a line

  ```
  package packageName;
  ```

  as the first instruction in the source file containing the classes
- Package name consists of one or more identifiers separated by periods

Continued

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Organizing Related Classes into Packages (cont.)

- For example, to put the `Financial` class introduced into a package named `com.horstmann.bigjava, the Financial.java` file must start as follows:

```
package com.horstmann.bigjava;

public class Financial
{
   . . .
}
```

- Default package has no name, no `package` statement

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Important Packages in the Java Library

Package	Purpose	Sample Class
java.lang	Language support	Math
java.util	Utilities	Random
java.io	Input and output	PrintStream
java.awt	Abstract Windowing Toolkit	Color
java.applet	Applets	Applet
java.net	Networking	Socket
java.sql	Database Access	ResultSet
javax.swing	Swing user interface	JButton
org.omg.CORBA	Common Object Request Broker Architecture	IntHolder

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Syntax 8.2 Package Specification

```
package packageName;
```

Example:

```
package com.horstmann.bigjava;
```

Purpose:

To declare that all classes in this file belong to a particular package.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Importing Packages

- Can always use class without importing
  ```
  java.util.Scanner in = new java.util.Scanner(System.in);
  ```
- Tedious to use fully qualified name
- Import lets you use shorter class name
  ```
  import java.util.Scanner; . . .
  Scanner in = new Scanner(System.in)
  ```
- Can import all classes in a package
  ```
  import java.util.*;
  ```
- Never need to import `java.lang`
- You don't need to import other classes in the same package

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Package Names and Locating Classes

- Use packages to avoid name clashes

  java.util.Timer vs. javax.swing.Timer

- Package names should be unambiguous
- Recommendation: start with reversed domain name

  com.horstmann.bigjava

  edu.sjsu.cs.walters: for Bertha Walters' classes (walters@cs.sjsu.edu)

- Path name should match package name

  `com/horstmann/bigjava/Financial.java`

Continued

Copyright © 2008 by John Wiley & Sons. All rights reserved.

Package Names and Locating Classes (cont.)

- Path name starts with class path

```
export CLASSPATH=/home/walters/lib:.
set CLASSPATH=c:\home\walters\lib;.
```

- Class path contains the base directories that may contain package directories

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Base Directories and Subdirectories for Packages

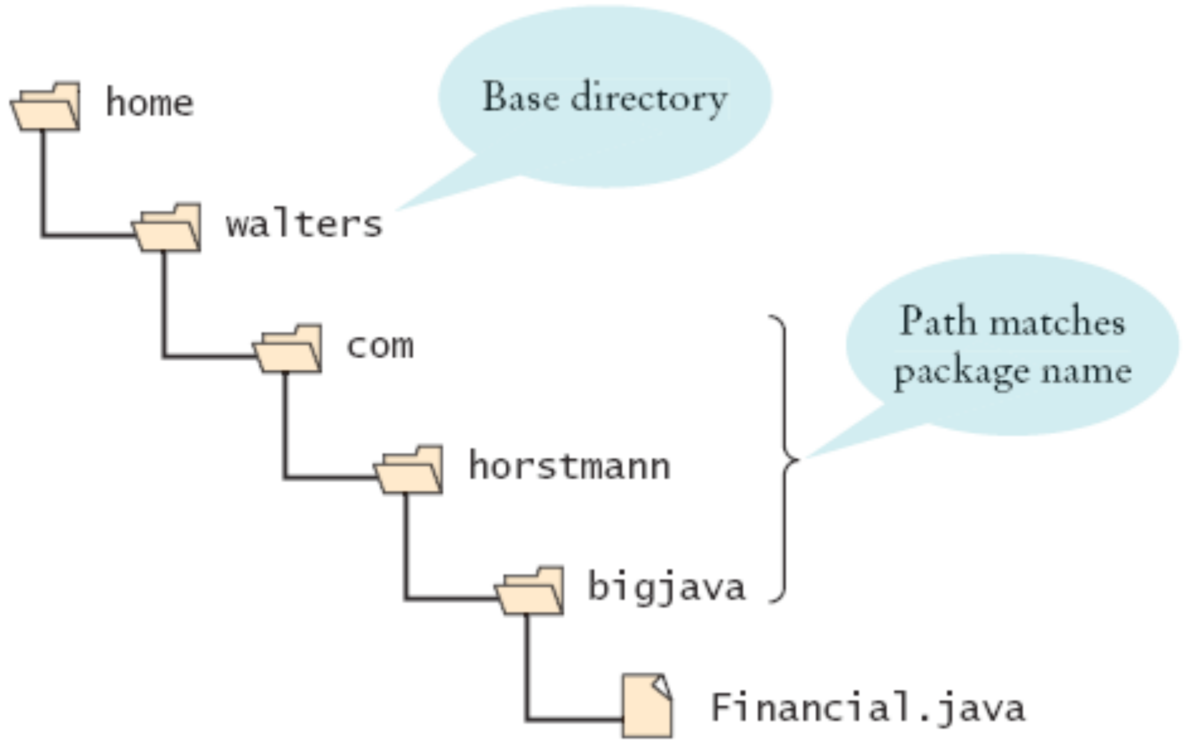


Figure 5
Base Directories and Subdirectories for Packages

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.18

Which of the following are packages?

a. `java`
b. `java.lang`
c. `java.util`
d. `java.lang.Math`

Answer:

a. No
b. Yes
c. Yes
d. No

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.19

Is a Java program without `import` statements limited to using the default and `java.lang` packages?

Answer: No – you simply use fully qualified names for all other classes, such as `java.util.Random` and `java.awt.Rectangle`.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

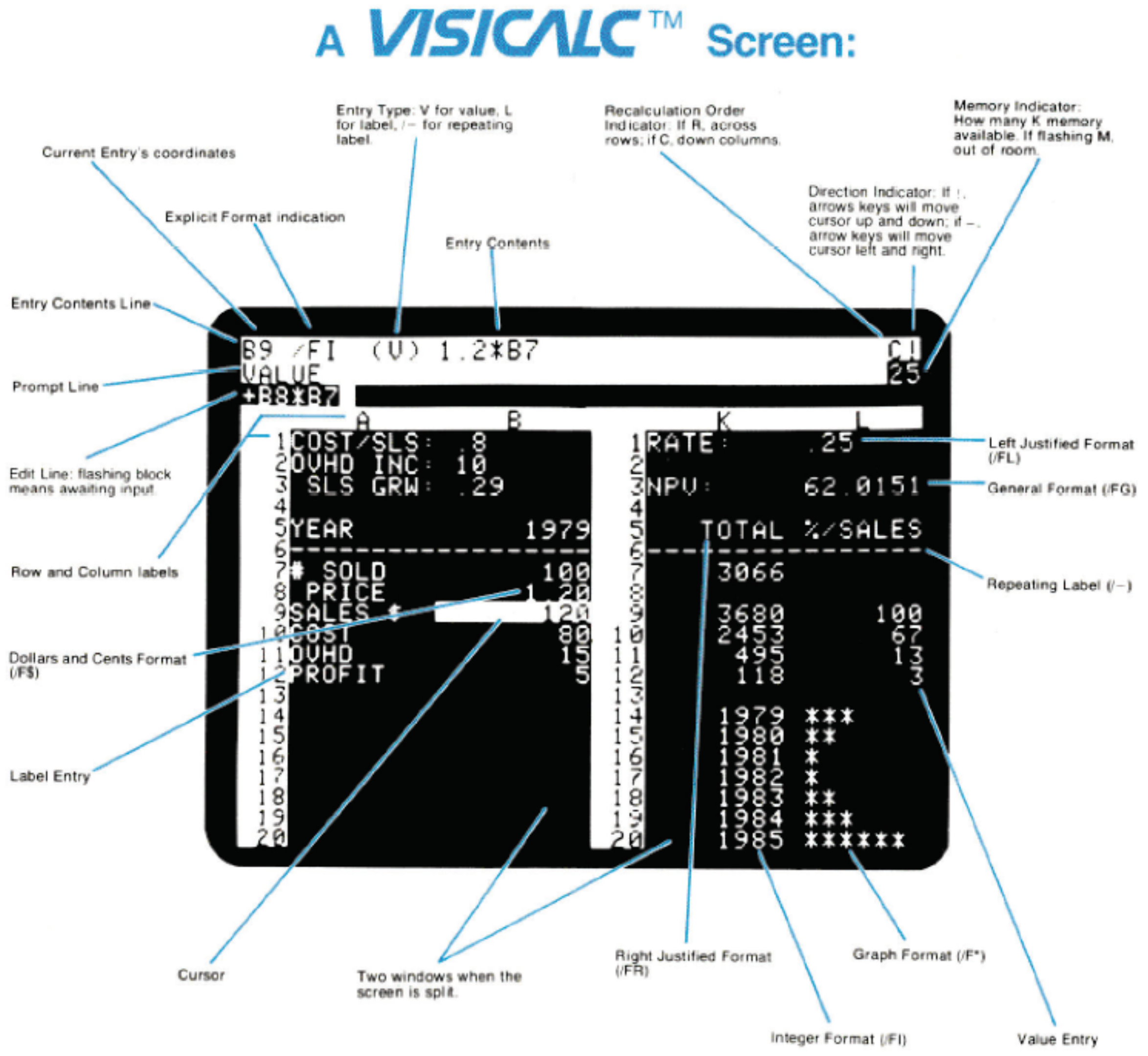
Self Check 8.20

Suppose your homework assignments are located in the directory `/home/me/cs101` (`c:\me\cs101` on Windows). Your instructor tells you to place your homework into packages. In which directory do you place the class `hw1.problem1.TicTacToeTester`?

Answer: `/home/me/cs101/hw1/problem1` or, on Windows, `c:\me\cs101\hw1\problem1`

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

The Explosive Growth of Personal Computers


A VISICALC™ Screen:
Current Entry's coordinates
Explicit Format indication
Entry Type: V for value, L for label, /– for repeating label.
Entry Contents
Recalculation Order Indicator: If R, across rows; if C, down columns.
Memory Indicator: How many K memory available. If flashing M, out of room.
Direction Indicator: If ↕, arrows keys will move cursor up and down; if ↔, arrow keys will move cursor left and right.
Entry Contents Line
Prompt Line
Edit Line: flashing block means awaiting input
Row and Column labels
Dollars and Cents Format (/F$)
Label Entry
Left Justified Format (/FL)
General Format (/FG)
Repeating Label (/–)
Cursor
Two windows when the screen is split.
Right Justified Format (/FR)
Graph Format (/F*)
Integer Format (/FI)
Value Entry

```
B9 /FI  (V) 1.2*B7                          C!
VALUE                                       25
+B8*B7
      A           B         K           L
 1COST/SLS:  .8          1RATE:      .25
 2OVHD INC:  10          2
 3 SLS GRW: .29          3NPV:     62.0151
 4                       4
 5YEAR         1979      5   TOTAL  %/SALES
 6-----------------      6-------------------
 7# SOLD        100      7   3066
 8 PRICE       1.20      8
 9SALES $       120      9   3680      100
10COST           80     10   2453       67
11OVHD           15     11    495       13
12PROFIT          5     12    118        3
13                      13
14                      14   1979 ***
15                      15   1980 **
16                      16   1981 *
17                      17   1982 *
18                      18   1983 **
19                      19   1984 ***
20                      20   1985 ******
```

The VisiCalc Spreadsheet Running on an Apple II

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Unit Testing Frameworks

- Unit test frameworks simplify the task of writing classes that contain many test cases
- JUnit: `http://junit.org`
  Built into some IDEs like BlueJ and Eclipse
- Philosophy: whenever you implement a class, also make a companion test class. Run all tests whenever you change your code

Continued

Copyright © 2008 by John Wiley & Sons. All rights reserved.

Unit Testing Frameworks

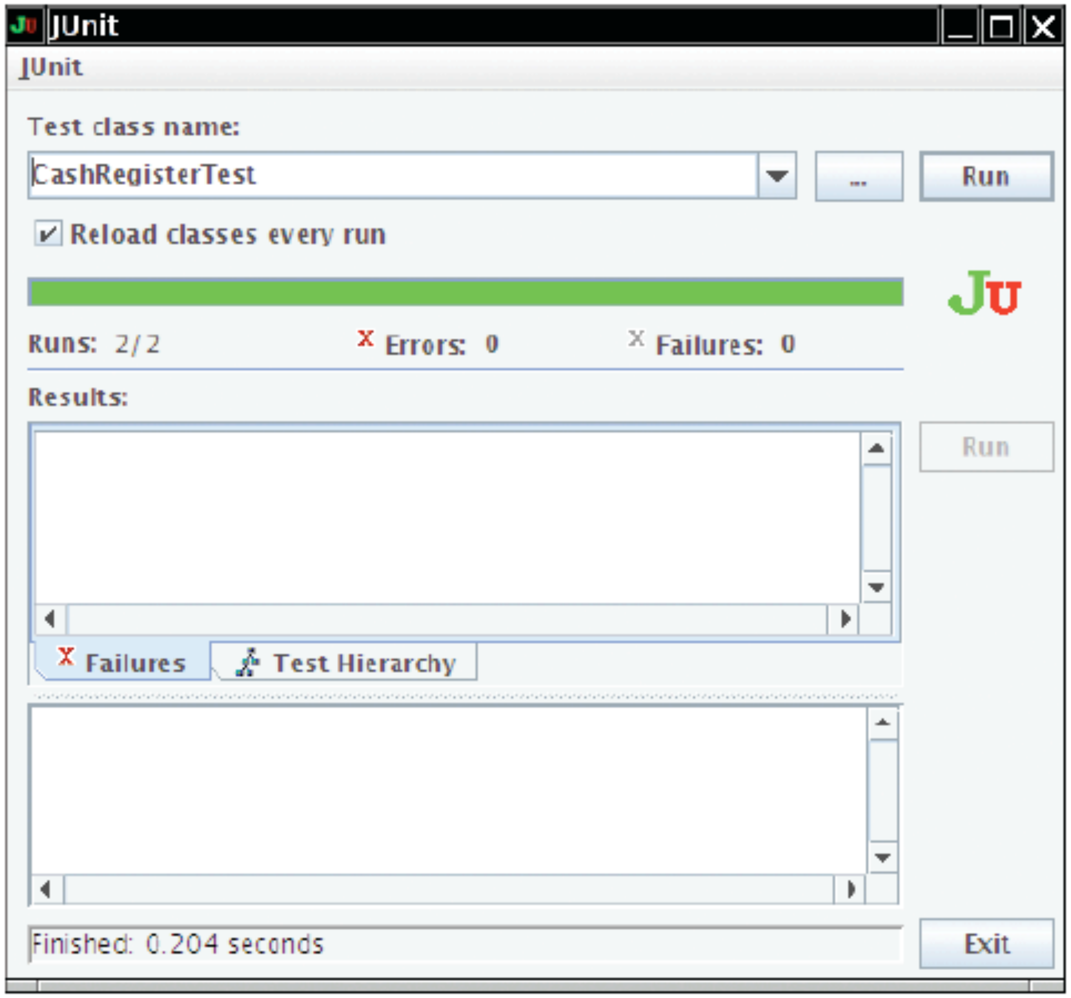


Figure 6 Unit Testing with JUnit

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.21

Provide a JUnit test class with one test case for the `Earthquake` class in Chapter 5.

Answer: Here is one possible answer, using the JUnit 4 style.

```
public class EarthquakeTest
{
   @Test public void testLevel4()
   {
      Earthquake quake = new Earthquake(4);
      Assert.assertEquals("Felt by many people, no
          destruction", quake.getDescription());
   }
}
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Self Check 8.22

What is the significance of the `EPSILON` parameter in the `assertEquals` method?

Answer: It is a tolerance threshold for comparing floating-point numbers. We want the equality test to pass if there is a small roundoff error.

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.