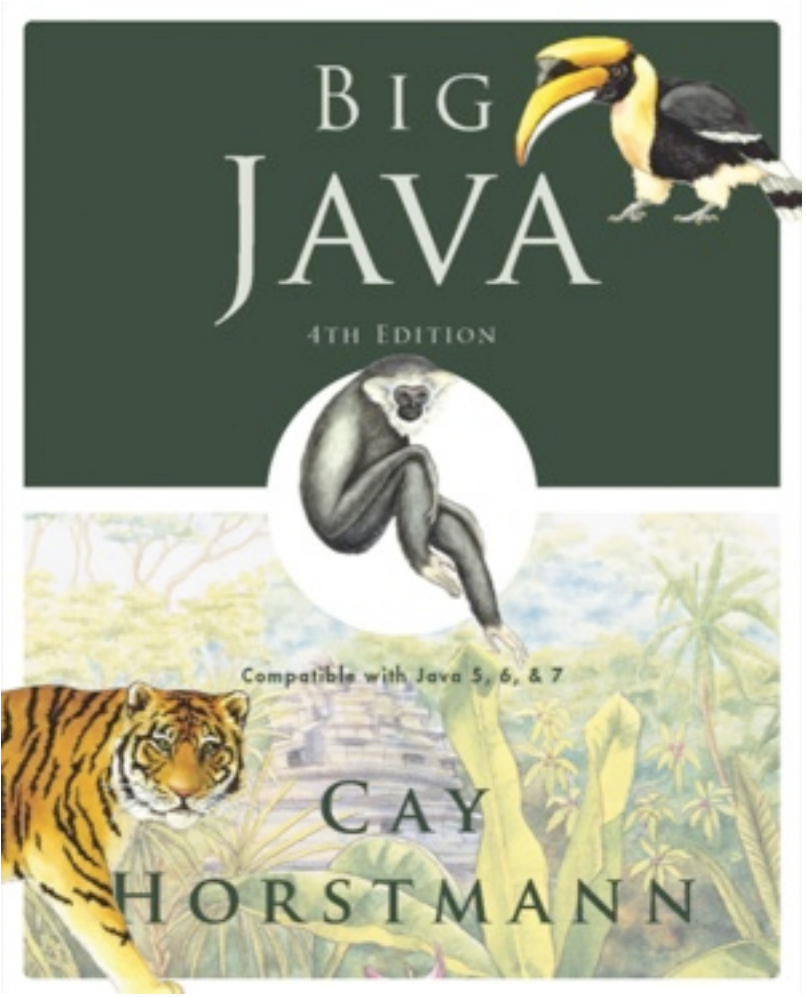


Chapter 3 – Implementing Classes

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

Instance Variables

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

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

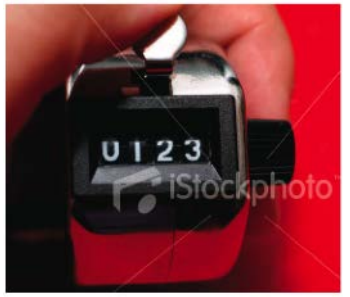


Figure 1 A Tally Counter

- 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 (*`private`*)*
  - *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

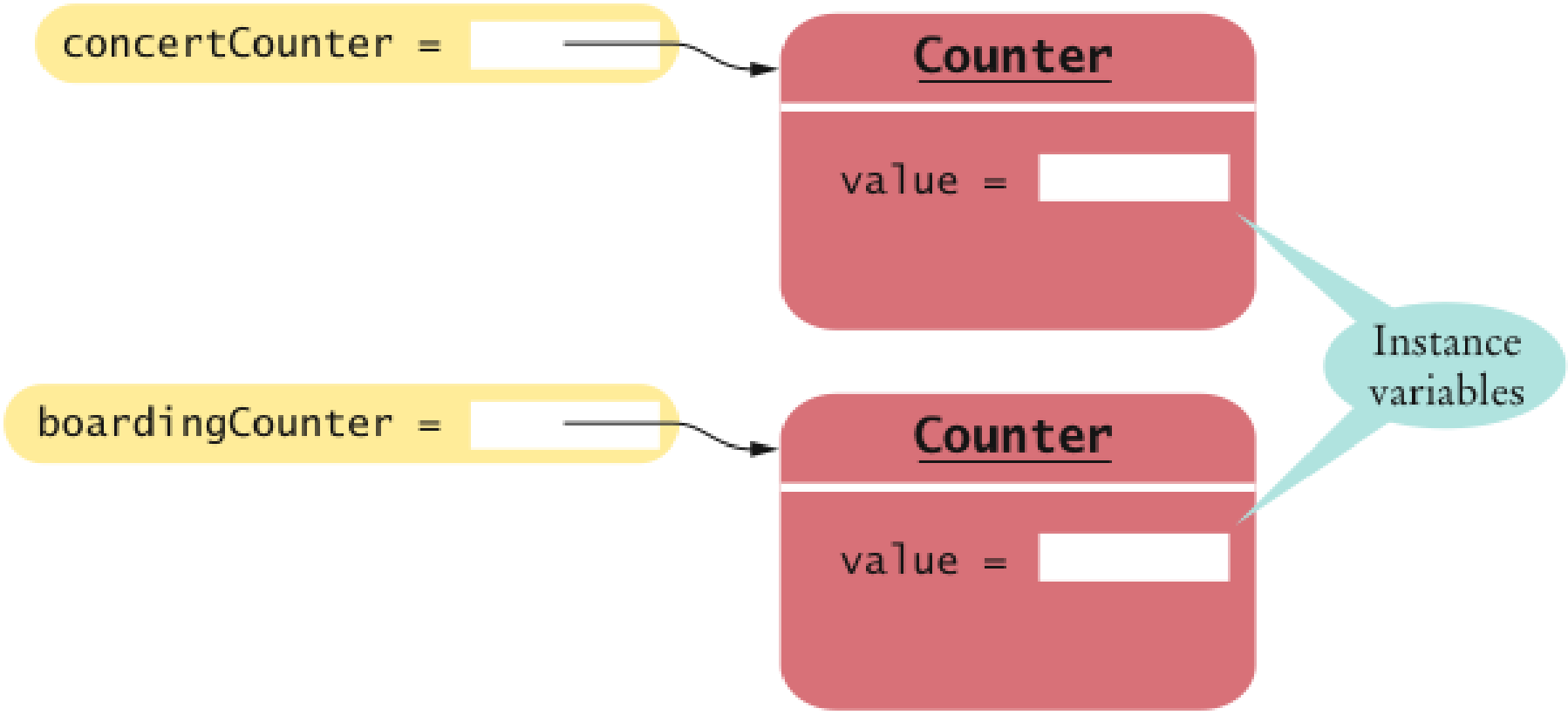


Figure 2 Instance Variables

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

Instance Variables

- **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 implementor of a class to locate errors and change implementations

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: 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()`

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

BankAccount Public Interface

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

```
public class BankAccount
{
   // private variables--filled in later
   // Constructors
   public BankAccount()
   {
      // body--filled in later
   }

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

Continued

BankAccount Public Interface (cont.)

```
   // 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
   }
}
```

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

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

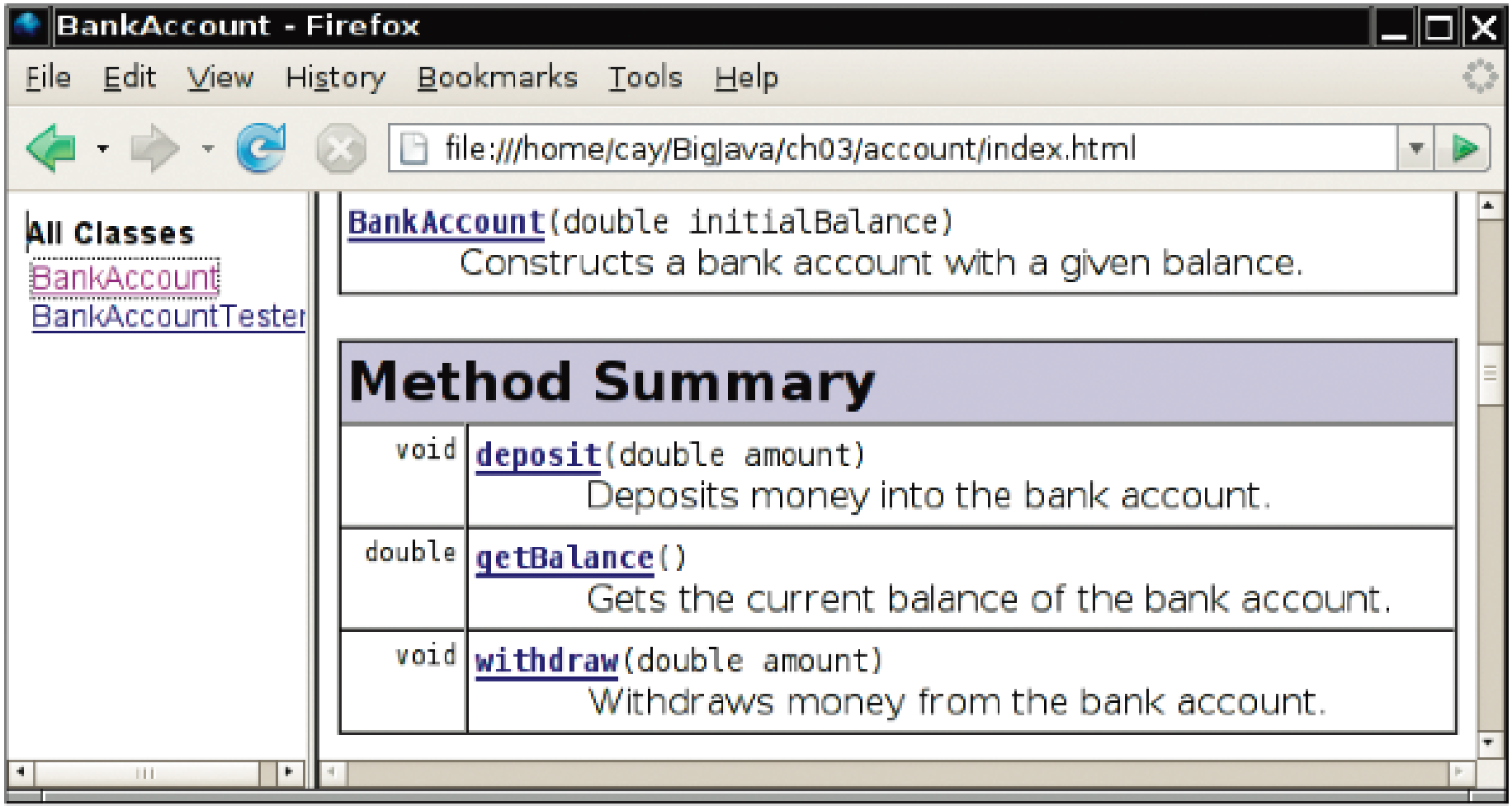


Figure 3 A Method Summary Generated by javadoc

Javadoc Method Detail

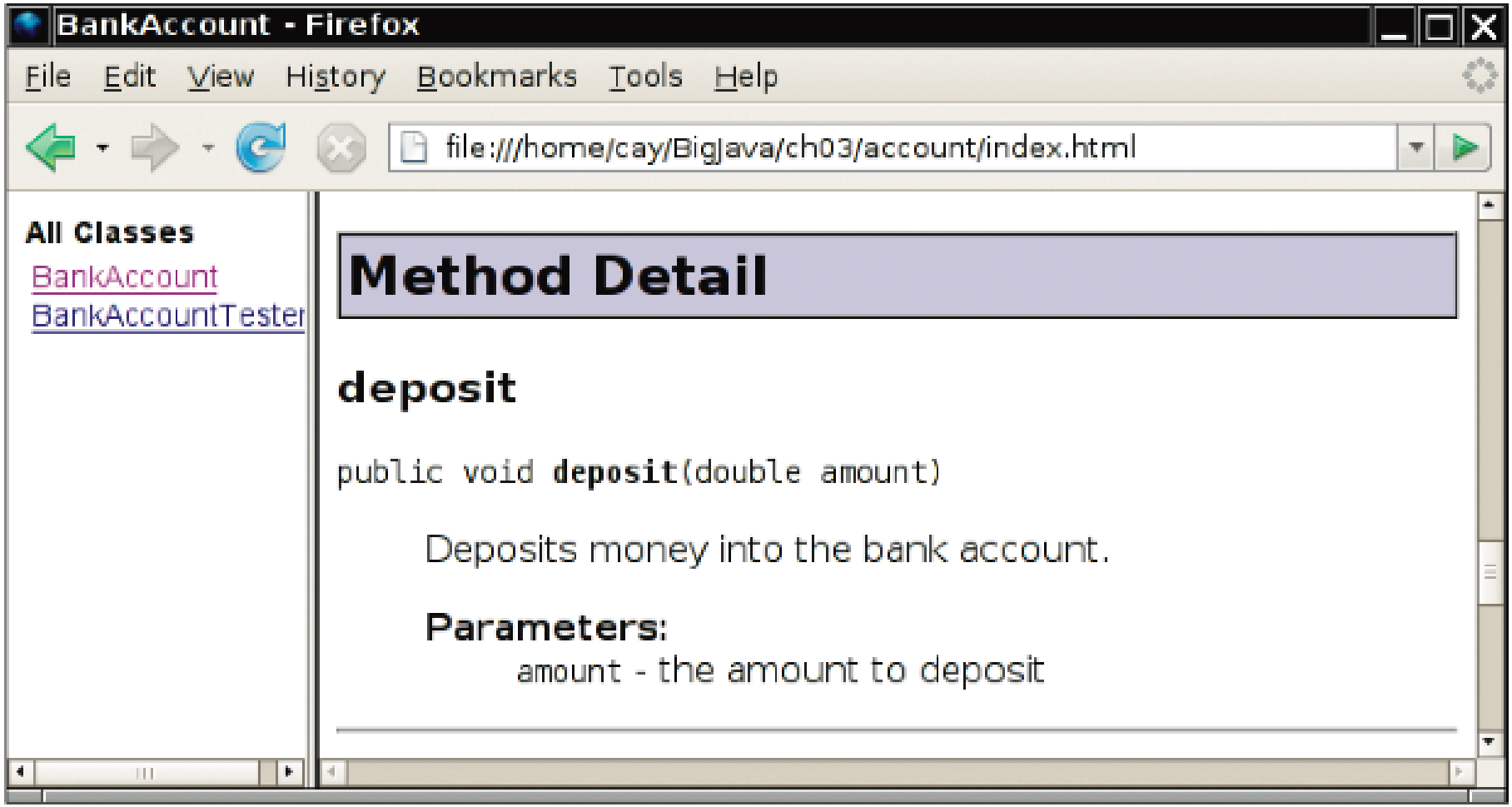


Figure 4 Method Detail Generated by javadoc

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);
```

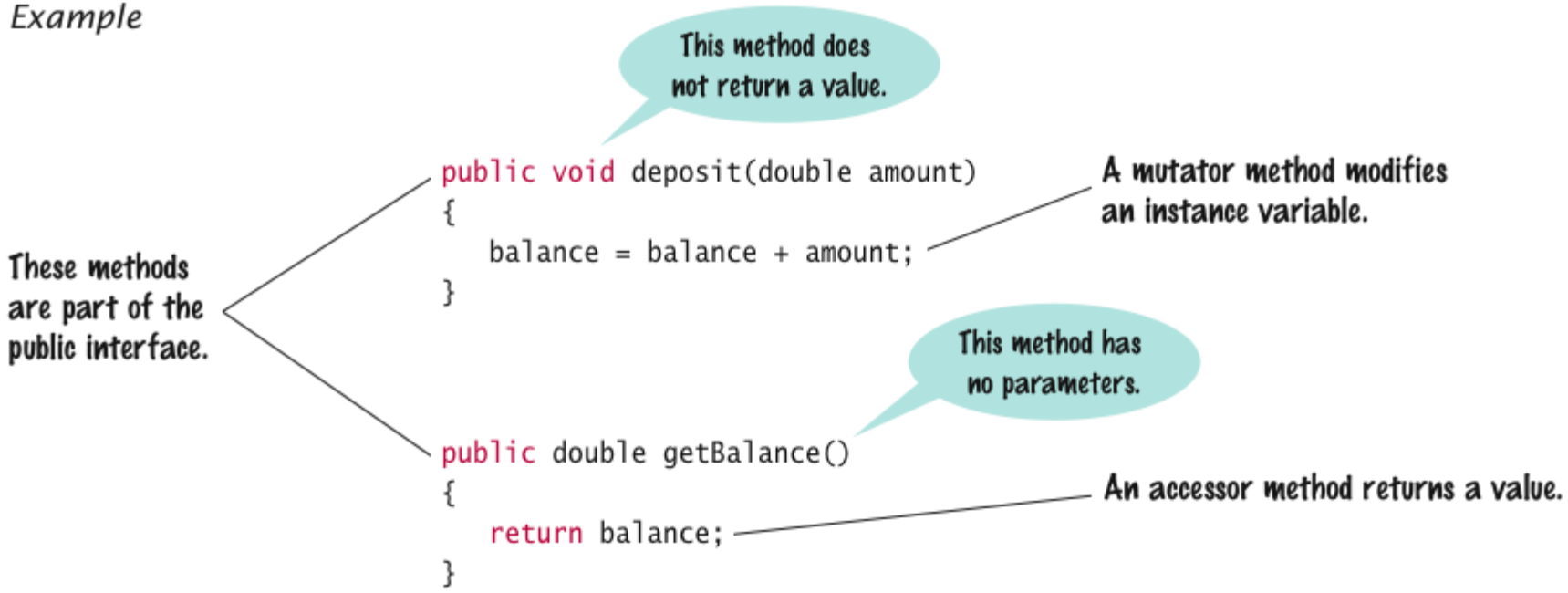
  - *Create a new object of type* `BankAccount`
  - *Call the second constructor (because a construction parameter is supplied in the constructor call)*
  - *Set the parameter variable* `initialBalance` *to* `1000`
  - *Set the* `balance` *instance variable of the newly created object to* `initialBalance`
  - *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;
  }
  ```

ch03/account/BankAccount.java

```
1  /**
2     A bank account has a balance that can be changed by
3     deposits and withdrawals.
4  */
5  public class BankAccount
6  {
7     private double balance;
8
9     /**
10       Constructs a bank account with a zero balance.
11    */
12    public BankAccount()
13    {
14       balance = 0;
15    }
16
17    /**
18       Constructs a bank account with a given balance.
19       @param initialBalance the initial balance
20    */
21    public BankAccount(double initialBalance)
22    {
23       balance = initialBalance;
24    }
```

Continued

ch03/account/BankAccount.java (cont.)

```
25
26      /**
27         Deposits money into the bank account.
28         @param amount the amount to deposit
29      */
30      public void deposit(double amount)
31      {
32         balance = balance + amount;
33      }
34
35      /**
36         Withdraws money from the bank account.
37         @param amount the amount to withdraw
38      */
39      public void withdraw(double amount)
40      {
41         balance = balance - amount;
42      }
43
```

Continued

ch03/account/BankAccount.java (cont.)

```
44      /**
45         Gets the current balance of the bank account.
46         @return the current balance
47      */
48      public double getBalance()
49      {
50         return balance;
51      }
52   }
```

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

```
1  /**
2     A class to test the BankAccount class.
3  */
4  public class BankAccountTester
5  {
6     /**
7        Tests the methods of the BankAccount class.
8        @param args not used
9     */
10    public static void main(String[] args)
11    {
12       BankAccount harrysChecking = new BankAccount();
13       harrysChecking.deposit(2000);
14       harrysChecking.withdraw(500);
15       System.out.println(harrysChecking.getBalance());
16       System.out.println("Expected: 1500");
17    }
18 }
```

Program Run:

```
1500
Expected: 1500
```

Local 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*

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

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`

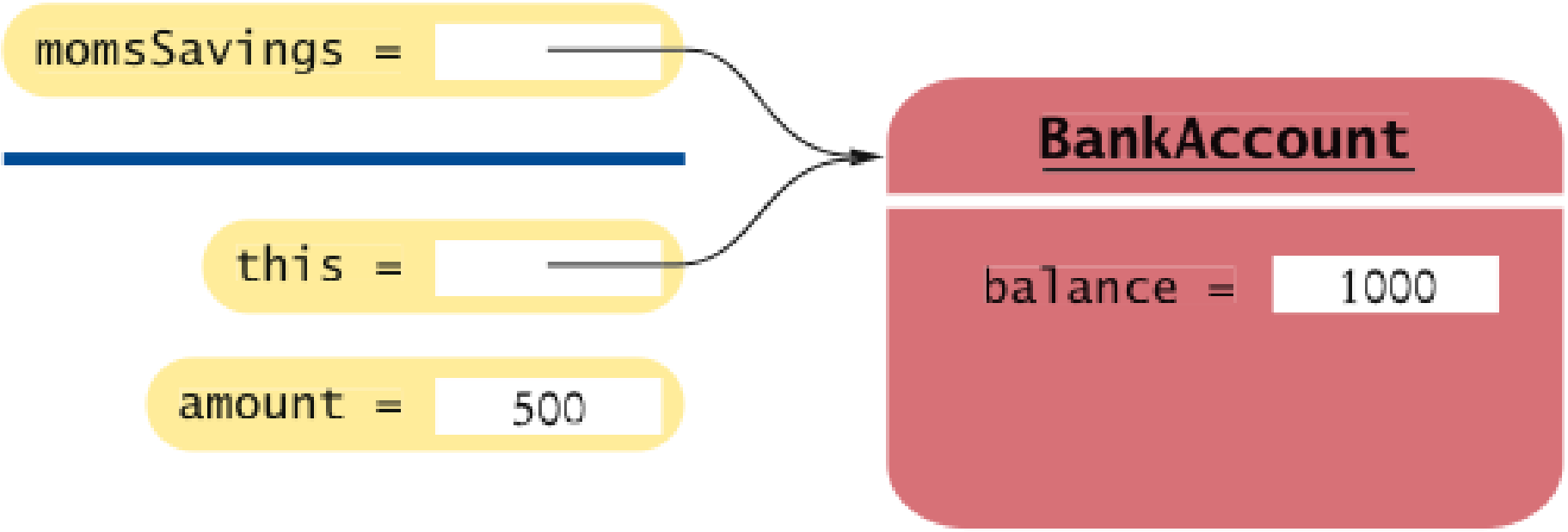


Figure 6 The Implicit Parameter of a Method Call

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
   }
}
```