Chapter 2 – An Introduction to Objects and Classes
Chapter Goals

• To learn about variables
• To understand the concepts of classes and objects
• To be able to call methods
• To learn about parameters and return values
• To be able to browse the API documentation
• To implement test programs
• To understand the difference between objects and object references
• To write programs that display simple shapes
Types

- A **type** defines a set of values and the operations that can be carried out on the values

Examples:

- 13 has type `int`
- "Hello, World" has type `String`
- `System.out` has type `PrintStream`

Java has separate types for **integers** and **floating-point numbers**

- The **double** type denotes floating-point numbers

A value such as 13 or 1.3 that occurs in a Java program is called a **number literal**
## Number Literals

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>int</td>
<td>An integer has no fractional part.</td>
</tr>
<tr>
<td>-6</td>
<td>int</td>
<td>Integers can be negative.</td>
</tr>
<tr>
<td>0</td>
<td>int</td>
<td>Zero is an integer.</td>
</tr>
<tr>
<td>0.5</td>
<td>double</td>
<td>A number with a fractional part has type double.</td>
</tr>
<tr>
<td>1.0</td>
<td>double</td>
<td>An integer with a fractional part .0 has type double.</td>
</tr>
<tr>
<td>1E6</td>
<td>double</td>
<td>A number in exponential notation: $1 \times 10^6$ or 1000000. Numbers in exponential notation always have type double.</td>
</tr>
<tr>
<td>2.96E-2</td>
<td>double</td>
<td>Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$</td>
</tr>
<tr>
<td>100,000</td>
<td></td>
<td><strong>Error:</strong> Do not use a comma as a decimal separator.</td>
</tr>
<tr>
<td>3 1/2</td>
<td></td>
<td><strong>Error:</strong> Do not use fractions; use decimal notation: 3.5.</td>
</tr>
</tbody>
</table>
Number Types

• A **type** defines a set of values and the operations that can be carried out on the values

• Number types are **primitive types**
  
  • *Numbers are not objects*

• Numbers can be combined by arithmetic operators such as +, -, and *
Self Check 2.1

What is the type of the values 0 and "0"?

Answer: int and String.
Self Check 2.2

Which number type would you use for storing the area of a circle?

Answer: double.
Why is the expression `13.println()` an error?

**Answer:** An `int` is not an object, and you cannot call a method on it.
Write an expression to compute the average of the values $x$ and $y$.

**Answer:** $(x + y) \times 0.5$
Variables

• Use a **variable** to store a value that you want to use at a later time

• A variable has a type, a name, and a value:

  ```java
  String greeting = "Hello, World!"
  PrintStream printer = System.out;
  int width = 13;
  ```

• Variables can be used in place of the values that they store:

  ```java
  printer.println(greeting);
  // Same as System.out.println("Hello, World!")
  printer.println(width);
  // Same as System.out.println(20)
  ```
Variables

• It is an error to store a value whose type does not match the type of the variable:

```java
String greeting = 20;  // ERROR: Types don't match
```
### Variable Declarations

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>int width = 10;</td>
<td>Declares an integer variable and initializes it with 10.</td>
</tr>
<tr>
<td>int area = width * height;</td>
<td>The initial value can depend on other variables. (Of course, width and</td>
</tr>
<tr>
<td></td>
<td>height must have been previously declared.)</td>
</tr>
<tr>
<td>![Error] height = 5;</td>
<td><strong>Error:</strong> The type is missing. This statement is not a declaration but</td>
</tr>
<tr>
<td></td>
<td>an assignment of a new value to an existing variable—see Section 2.3.</td>
</tr>
<tr>
<td>![Error] int height = &quot;5&quot;;</td>
<td><strong>Error:</strong> You cannot initialize a number with a string.</td>
</tr>
<tr>
<td>int width, height;</td>
<td>Declares two integer variables in a single statement. In this book, we</td>
</tr>
<tr>
<td></td>
<td>will declare each variable in a separate statement.</td>
</tr>
</tbody>
</table>
Identifiers

• **Identifier**: name of a variable, method, or class

• Rules for identifiers in Java:
  
  • *Can be made up of letters, digits, and the underscore (\_) and dollar sign ($) characters*
  
  • *Cannot start with a digit*
  
  • *Cannot use other symbols such as ? or %*
  
  • *Spaces are not permitted inside identifiers*
  
  • *You cannot use reserved words such as public*
  
  • *They are case sensitive*
Identifiers

• By convention, variable names start with a lowercase letter
  • “Camel case”: *Capitalize the first letter of a word in a compound word such as farewellMessage*

• By convention, class names start with an uppercase letter

• Do not use the $ symbol in names — it is intended for names that are automatically generated by tools
Syntax 2.1 Variable Declaration

Syntax

typeName variableName = value;

or

typeName variableName;

Example

String greeting = "Hello, Dave!";

See the rules for and table of examples of valid names.

Use a descriptive variable name.

The type specifies what can be done with values stored in this variable.

A variable declaration ends with a semicolon.

Supplying an initial value is optional, but it is usually a good idea.
# Variable Names

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>farewellMessage</td>
<td>Use “camel case” for variable names consisting of multiple words.</td>
</tr>
<tr>
<td>x</td>
<td>In mathematics, you use short variable names such as x or y. This is legal in Java, but not very common, because it can make programs harder to understand.</td>
</tr>
<tr>
<td>! Greeting</td>
<td><strong>Caution:</strong> Variable names are case-sensitive. This variable name is different from greeting.</td>
</tr>
<tr>
<td>❌ 6pack</td>
<td><strong>Error:</strong> Variable names cannot start with a number.</td>
</tr>
<tr>
<td>❌ farewell message</td>
<td><strong>Error:</strong> Variable names cannot contain spaces.</td>
</tr>
<tr>
<td>❌ public</td>
<td><strong>Error:</strong> You cannot use a reserved word as a variable name.</td>
</tr>
</tbody>
</table>
Which of the following are legal identifiers?

Greetings

101dalmatians
Hello, World

Answer: Only the first two are legal identifiers.
Define a variable to hold your name. Use camel case in the variable name.

Answer:

```java
String myName = "John Q. Public";
```
The Assignment Operator

• Assignment operator: =

• Used to change the value of a variable:

```java
int width = 10; 1
width = 20; 2
```

1 width = 10

2 width = 20
Uninitialized Variables

• It is an error to use a variable that has never had a value assigned to it:

```java
int height;
width = height; // ERROR—uninitialized variable height
```

• Remedy: assign a value to the variable before you use it:

```java
int height = 30;
width = height; // OK
```

• Even better, initialize the variable when you declare it:

```java
int height = 30;
int width = height; // OK
```
Syntax 2.2 Assignment

Syntax

```
variableName = value;
```

Example

```
double width = 30;
width = width + 10;
```

This is a variable declaration.
This is an assignment statement.
The value of this variable is changed.
The new value of the variable
The same name can occur on both sides.
See Figure 3.
Assignment

• The right-hand side of the $=$ symbol can be a mathematical expression:

\[
\text{width} = \text{height} + 10;
\]

• Means:

1. compute the value of $\text{width} + 10$
2. store that value in the variable $\text{width}$

Figure 3  Executing the Statement $\text{width} = \text{width} + 10$
Self Check 2.7

Is $12 = 12$ a valid expression in the Java language?

**Answer:** No, the left-hand side of the $=$ operator must be a variable.
Self Check 2.8

How do you change the value of the `greeting` variable to "Hello, Nina!"?

Answer:

```java
    greeting = "Hello, Nina!";
```

Note that

```java
    String greeting = "Hello, Nina!";
```

is not the right answer – that statement defines a new variable.
Objects and Classes

- **Object**: entity that you can manipulate in your programs (by calling methods)
- Each object belongs to a **class**
- Example: `System.out` belongs to the class `PrintStream`

![Diagram of PrintStream class]

**Figure 4** Representation of the `System.out` Object
Methods

- **Method**: sequence of instructions that accesses the data of an object
- You manipulate objects by calling its methods
- **Class**: declares the methods that you can apply to its objects
- Class determines legal methods:
  
  ```java
  String greeting = "Hello";
  greeting.println() // Error
  greeting.length() // OK
  ```
- **Public Interface**: specifies what you can do with the objects of a class
Overloaded Method

• **Overloaded method:** when a class declares two methods with the same name, but different parameters

• Example: the `PrintStream` class declares a second method, also called `println`, as

  ```java
  public void println(int output)
  ```
A Representation of Two String Objects

Figure 5  A Representation of Two String Objects
String Methods

- **length**: counts the number of characters in a string:

  ```java
  String greeting = "Hello, World!";
  int n = greeting.length(); // sets n to 13
  ```

- **toUpperCase**: creates another String object that contains the characters of the original string, with lowercase letters converted to uppercase:

  ```java
  String river = "Mississippi";
  String bigRiver = river.toUpperCase(); // sets bigRiver to "MISSISSIPPI"
  ```

- When applying a method to an object, make sure method is defined in the appropriate class:

  ```java
  System.out.length(); // This method call is an error
  ```
Self Check 2.9

How can you compute the length of the string "Mississippi"?

Answer: `river.length()` or "Mississippi".length()
Self Check 2.10

How can you print out the uppercase version of "Hello, World!"?

Answer:

```java
System.out.println(greeting.toUpperCase());
```
Self Check 2.11

Is it legal to call `river.println()`? Why or why not?

**Answer:** It is not legal. The variable `river` has type `String`. The `println` method is not a method of the `String` class.
Parameters

- **Parameter**: an input to a method
- **Implicit parameter**: the object on which a method is invoked:
  ```java
  System.out.println(greeting)
  ```
- **Explicit parameters**: all parameters except the implicit parameter:
  ```java
  System.out.println(greeting)
  ```
- Not all methods have explicit parameters:
  ```java
greeting.length() // has no explicit parameter
  ```
Passing a Parameter

Figure 6  Passing a Parameter to the println Method
Return Values

- **Return value**: a result that the method has computed for use by the code that called it:

  ```java
  int n = greeting.length(); // return value stored in n
  ```

![Diagram](image)

**Figure 7** Invoking the `length` Method on a String Object
Passing Return Values

• You can also use the return value as a parameter of another method:

```java
System.out.println(greeting.length());
```

![Diagram showing method call flow]

**Figure 8** Passing the Result of a Method Call to Another Method

• Not all methods return values. Example: `println`
A More Complex Call

- `String` method `replace` carries out a search-and-replace operation:

  ```java
river.replace("issipp", "our")
  // constructs a new string ("Missouri")
```

- This method call has
  - one implicit parameter: the string "Mississippi"
  - two explicit parameters: the strings "issipp" and "our"
  - a return value: the string "Missouri"
Self Check 2.12

What are the implicit parameters, explicit parameters, and return values in the method call `river.length()`?

**Answer:** The implicit parameter is `river`. There is no explicit parameter. The return value is 11.
Self Check 2.13

What is the result of the call `river.replace("p", "s")`?

Answer: "Missississi".
Self Check 2.14

What is the result of the call

`greeting.replace("World", "Dave").length()`?

**Answer:** 12.
Self Check 2.15

How is the `toUpperCase` method defined in the `String` class?

**Answer:** As `public String toUpperCase()`, with no explicit parameter and return type `String`. 
Rectangular Shapes and Rectangle Objects

- Objects of type `Rectangle` describe rectangular shapes:

  ![Figure 10 Rectangular Shapes](image)
Rectangular Shapes and Rectangle Objects

- A `Rectangle` object isn’t a rectangular shape – it is an object that contains a set of numbers that describe the rectangle:

```
Figure 11  Rectangle Objects
```

```
<table>
<thead>
<tr>
<th>Rectangle</th>
<th>Rectangle</th>
<th>Rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 5</td>
<td>x = 35</td>
<td>x = 45</td>
</tr>
<tr>
<td>y = 10</td>
<td>y = 30</td>
<td>y = 0</td>
</tr>
<tr>
<td>width = 20</td>
<td>width = 20</td>
<td>width = 30</td>
</tr>
<tr>
<td>height = 30</td>
<td>height = 20</td>
<td>height = 20</td>
</tr>
</tbody>
</table>
```
Constructing Objects

new Rectangle(5, 10, 20, 30)

• Detail:

  1. The new operator makes a Rectangle object

  2. It uses the parameters (in this case, 5, 10, 20, and 30) to initialize the data of the object

  3. It returns the object

• Usually the output of the new operator is stored in a variable:

  Rectangle box = new Rectangle(5, 10, 20, 30);
Constructing Objects

- **Construction:** the process of creating a new object

- The four values 5, 10, 20, and 30 are called the *construction parameters*

- Some classes let you construct objects in multiple ways:

  ```java
  new Rectangle()
  // constructs a rectangle with its top-left corner
  // at the origin (0, 0), width 0, and height 0
  ```
Syntax 2.3 Object Construction

Syntax

new ClassName(parameters)

Example

The new expression yields an object.

Rectangle box = new Rectangle(5, 10, 20, 30);

System.out.println(new Rectangle());

Usually, you save the constructed object in a variable.

You can also pass the constructed object to a method.

Construction parameters

Supply the parentheses even when there are no parameters.
Self Check 2.16

How do you construct a square with center (100, 100) and side length 20?

Answer:

```java
new Rectangle(90, 90, 20, 20)
```
Self Check 2.17

The `getWidth` method returns the width of a `Rectangle` object. What does the following statement print?

```java
System.out.println(new Rectangle().getWidth());
```

Answer:

0
Accessor and Mutator Methods

• **Accessor method**: does not change the state of its implicit parameter:

```java
    double width = box.getWidth();
```

• **Mutator method**: changes the state of its implicit parameter:

```java
    box.translate(15, 25);
```

**Figure 12**
Using the `translate` Method to Move a Rectangle
Self Check 2.18

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

**Answer:** An accessor – it doesn’t modify the original string but returns a new string with uppercase letters.
Self Check 2.19

Which call to `translate` is needed to move the `box` rectangle so that its top-left corner is the origin (0, 0)?

**Answer:** `box.translate(-5, -10)`, provided the method is called immediately after storing the new rectangle into `box`. 
The API Documentation

• **API**: Application Programming Interface

• **API documentation**: lists classes and methods in the Java library

• [http://java.sun.com/javase/7/docs/api/index.html](http://java.sun.com/javase/7/docs/api/index.html)
The API Documentation of the Standard Java Library

Figure 13  The API Documentation of the Standard Java Library
The API Documentation for the Rectangle Class

Figure 14 The API Documentation for the Rectangle Class
Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void add(int newx, int newy)</td>
<td>Adds a point, specified by the integer arguments newx, newy to the bounds of this Rectangle.</td>
</tr>
<tr>
<td>void add(Point pt)</td>
<td>Adds the specified Point to the bounds of this Rectangle.</td>
</tr>
<tr>
<td>void add( Rectangle r)</td>
<td>Adds a Rectangle to this Rectangle.</td>
</tr>
<tr>
<td>boolean contains(int x, int y)</td>
<td>Checks whether or not this Rectangle contains the point at the specified location (x, y).</td>
</tr>
<tr>
<td>boolean contains(int x, int y, int w, int h)</td>
<td>Checks whether this Rectangle entirely contains the Rectangle at the specified location (x, y) with the specified dimensions (w, h).</td>
</tr>
<tr>
<td>boolean contains(Point p)</td>
<td>Checks whether or not this Rectangle contains the specified Point.</td>
</tr>
<tr>
<td>boolean contains( Rectangle r)</td>
<td>Checks whether or not this Rectangle entirely contains the specified Rectangle.</td>
</tr>
</tbody>
</table>

Figure 15  The Method Summary for the Rectangle Class
Detailed Method Description

The detailed description of a method shows:

- The action that the method carries out
- The parameters that the method receives
- The value that it returns (or the reserved word void if the method doesn’t return any value)

Figure 16 The API Documentation of the translate Method
Packages

• **Package**: a collection of classes with a related purpose

• Import library classes by specifying the package and class name:

  ```java
  import java.awt.Rectangle;
  ```

• You don’t need to import classes in the `java.lang` package such as `String` and `System`
Syntax 2.4 Importing a Class from a Package

Syntax

```java
import packageName.ClassName;
```

Example

```java
import java.awt.Rectangle;
```

- Import statements must be at the top of the source file.
- You can look up the package name in the API documentation.
Self Check 2.20

Look at the API documentation of the String class. Which method would you use to obtain the string "hello, world!" from the string "Hello, World!"?

Answer: toLowerCase
Self Check 2.21

In the API documentation of the String class, look at the description of the trim method. What is the result of applying trim to the string " Hello, Space ! "? (Note the spaces in the string.)

**Answer:** "Hello, Space !" – only the leading and trailing spaces are trimmed.
The `Random` class is defined in the `java.util` package. What do you need to do in order to use that class in your program?

**Answer:** Add the statement

```java
import java.util.Random;
```

at the top of your program.
Implementing a Test Program

1. Provide a tester class.
2. Supply a `main` method.
3. Inside the `main` method, construct one or more objects.
4. Apply methods to the objects.
5. Display the results of the method calls.
6. Display the values that you expect to get.
```java
import java.awt.Rectangle;

public class MoveTester {
    public static void main(String[] args) {
        Rectangle box = new Rectangle(5, 10, 20, 30);

        // Move the rectangle
        box.translate(15, 25);

        // Print information about the moved rectangle
        System.out.print("x: ");
        System.out.println(box.getX());
        System.out.println("Expected: 20");

        System.out.print("y: ");
        System.out.println(box.getY());
        System.out.println("Expected: 35");
    }
}
```
Program Run:

x: 20
Expected: 20
y: 35
Expected: 35
Suppose we had called `box.translate(25, 15)` instead of `box.translate(15, 25)`. What are the expected outputs?

**Answer:**

```
x: 30, y: 25
```
Self Check 2.24

Why doesn’t the MoveTester program print the width and height of the rectangle?

**Answer:** Because the `translate` method doesn’t modify the shape of the rectangle.
Testing Classes in an Interactive Environment

Testing a Method Call in BlueJ
Object References

• Object reference: describes the location of an object

• The **new** operator returns a reference to a new object:
  
  ```java
  Rectangle box = new Rectangle();
  ```

• Multiple object variables can refer to the same object:
  
  ```java
  Rectangle box = new Rectangle(5, 10, 20, 30);
  Rectangle box2 = box;
  box2.translate(15, 25);
  ```

• Primitive type variables ≠ object variables
Figure 17 An Object Variable Containing an Object Reference
Object Variables and Number Variables

Figure 18  Two Object Variables Referring to the Same Object

Figure 19  A Number Variable Stores a Number
int luckyNumber = 13;
int luckyNumber = 13;  ①
int luckyNumber2 = luckyNumber;  ②
int luckyNumber = 13;  
int luckyNumber2 = luckyNumber;  
luckyNumber2 = 12;

Figure 20
Copying Numbers
Copying Object References

```java
Rectangle box = new Rectangle(5, 10, 20, 30);  // 1
```
Copying Object References (cont.)

```java
Rectangle box = new Rectangle(5, 10, 20, 30);  
Rectangle box2 = box;
```
Copying Object References (cont.)

Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
Box2.translate(15, 25);

Figure 21  Copying Object References
Self Check 2.25

What is the effect of the assignment `greeting2 = greeting`?

**Answer:** Now `greeting` and `greeting2` both refer to the same `String` object.
Self Check 2.26

After calling `greeting2.toUpperCase()`, what are the contents of `greeting` and `greeting2`?

**Answer:** Both variables still refer to the same string, and the string has not been modified. Recall that the `toUpperCase` method constructs a new string that contains uppercase characters, leaving the original string unchanged.
Instance Variables

• **Example:** tally counter

• Simulator statements:

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

• 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:

```java
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

\[
\text{accessSpecifer\ class\ Class\ Name} \\
\{ \\
\text{accessSpecifer\ type\ Name\ variable\ Name}; \\
\ldots \\
\}\n\]

Example

```java
public\ class\ Counter \\
\{ \\
\text{private\ int\ value}; \\
\ldots \\
\}
```

Instance variables should always be private.
Accessing Instance Variables

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

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

• The `getValue` method returns the current value:

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

• Private instance variables can only be accessed by methods of the same class
Self Check 2.27

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

Answer:

```java
public void reset()
{
    value = 0;
}
```
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.
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.
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 2.30

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.
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: 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:

```java
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; }
}
```
Self Check 2.32

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

Answer:

harrysChecking.withdraw(harrysChecking.getBalance())
What is wrong with this sequence of statements?

```java
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.
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.
public void withdraw(double amount)
{
    //implementation filled in later
}

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
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void deposit(double amount)</code></td>
<td>Deposits money into the bank account.</td>
</tr>
<tr>
<td><code>double getBalance()</code></td>
<td>Gets the current balance of the bank account.</td>
</tr>
<tr>
<td><code>void withdraw(double amount)</code></td>
<td>Withdraws money from the bank account.</td>
</tr>
</tbody>
</table>

**Figure 3**  A Method Summary Generated by javadoc
Figure 4  Method Detail Generated by javadoc
Provide documentation comments for the `Counter` class of Section 3.1.

**Answer:**
```java
/**
 * 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 2.35 (cont.)

    /**
     * Advances the value of this counter by 1.
     */
    public void count()
    {
        value = value + 1;
    
    }
Self Check 2.36

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

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

**Answer:**

```java
/**
   * 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 2.37

Why is the following documentation comment questionable?

```java
/**
   * 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.
Constructors contain instructions to initialize the instance variables of an object:

```java
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: \texttt{accessSpecifer returnType methodName(parameterType parameterName, \ldots)}
\begin{verbatim}
{
    method body
}
\end{verbatim}

Example:

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

This method does not return a value.

public double getBalance()
{
    return balance;
}

An accessor method returns a value.

These methods are part of the public interface.

A mutator method modifies an instance variable.

This method has no parameters.
Implementing Methods

• deposit method:

```java
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;
  }
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;
    }
}
/**
   * 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;
}
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     }
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

```java
private int accountNumber;
```

needs to be added to the class.
Self Check 2.39

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

```java
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 2.40

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

Answer:

```java
public int getWidth()
{
    return width;
}
```
Self Check 2.41

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:

```java
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
A class to test the BankAccount class.

```java
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 2.41

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 2.42

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

- 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 belong 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
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.
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.
Implicit Parameter

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

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

- In the call:
  ```java
  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

- \[ \text{balance} = \text{balance} + \text{amount}; \]
  
  actually means

  \[ \text{this.balance} = \text{this.balance} + \text{amount}; \]

- When you refer to an instance variable in a method, the compiler automatically applies it to the **this** reference
Implicit Parameters and this

```java
momsSavings = ___

this = ___

amount = 500

BankAccount

balance = 1000
```

**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:

```java
class BankAccount {
    private double balance;

    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:

```java
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:

```java
public class BankAccount {
    
    . . .
    public void monthlyFee() {
        this.withdraw(10); // Withdraw $10 from this account
    }
}
```
Self Check 2.45

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 2.45
Self Check 2.46

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.
Self Check 2.47

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 invoked on any object – and one explicit parameter, called `args`.
Graphical Applications and Frame Windows

To show a frame:

1. Construct an object of the JFrame class:
   ```java
   JFrame frame = new JFrame();
   ```
2. Set the size of the frame:
   ```java
   frame.setSize(300, 400);
   ```
1. If you’d like, set the title of the frame:
   ```java
   frame.setTitle("An Empty Frame");
   ```
4. Set the “default close operation”:
   ```java
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   ```
5. Make the frame visible:
   ```java
   frame.setVisible(true);
   ```
A Frame Window

Figure 22
A Frame Window
import javax.swing.JFrame;

public class EmptyFrameViewer {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setSize(300, 400);
        frame.setTitle("An Empty Frame");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
Self Check 2.48

How do you display a square frame with a title bar that reads "Hello, World!"?

Answer: Modify the EmptyFrameViewer program as follows:

```java
frame.setSize(300, 300);
frame.setTitle("Hello, World!");
```
Self Check 2.49

How can a program display two frames at once?

**Answer:** Construct two `JFrame` objects, set each of their sizes, and call `setVisible(true)` on each of them.
Drawing on a Component

• In order to display a drawing in a frame, define a class that extends the `JComponent` class

• Place drawing instructions inside the `paintComponent` method. That method is called whenever the component needs to be repainted:

```java
public class RectangleComponent extends JComponent {
    public void paintComponent(Graphics g) {
        Drawing instructions go here
    }
}
```
Classes **Graphics** and **Graphics2D**

- **Graphics** class lets you manipulate the graphics state (such as current color)
- **Graphics2D** class has methods to draw shape objects
- Use a cast to recover the **Graphics2D** object from the **Graphics** parameter:

```java
public class RectangleComponent extends JComponent {
    public void paintComponent(Graphics g) {
        // Recover Graphics2D
        Graphics2D g2 = (Graphics2D) g;
        // ... 
    }
}
```
Classes **Graphics and Graphics2D**

- Call method `draw` of the `Graphics2D` class to draw shapes, such as rectangles, ellipses, line segments, polygons, and arcs:

  ```java
  public class RectangleComponent extends JComponent {
      public void paintComponent(Graphics g) {
          ... 
          Rectangle box = new Rectangle(5, 10, 20, 30);
          g2.draw(box);
          ... 
      }
  }
  ```
Drawing Rectangles

Figure 23
Drawing Rectangles
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import javax.swing.JComponent;

/**
   * A component that draws two rectangles.
   */
public class RectangleComponent extends JComponent {
    public void paintComponent(Graphics g) {
        // Recover Graphics2D
        Graphics2D g2 = (Graphics2D) g;

        // Construct a rectangle and draw it
        Rectangle box = new Rectangle(5, 10, 20, 30);
        g2.draw(box);
    }
}

Continued
20    // Move rectangle 15 units to the right and 25 units down
21    box.translate(15, 25);
22
23    // Draw moved rectangle
24    g2.draw(box);
25    }
26    }
Using a Component

1. Construct a frame.

2. Construct an object of your component class:
   ```java
   RectangleComponent component = new RectangleComponent();
   ```

3. Add the component to the frame:
   ```java
   frame.add(component);
   ```

4. Make the frame visible.
import javax.swing.JFrame;

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

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

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

        frame.setVisible(true);
    }
}
Self Check 2.50

How do you modify the program to draw two squares?

Answer:

```java
Rectangle box = new Rectangle(5, 10, 20, 20);
```
Self Check 2.51

How do you modify the program to draw one rectangle and one square?

**Answer:** Replace the call to `box.translate(15, 25)` with

```java
box = new Rectangle(20, 35, 20, 20);
```
Self Check 2.52

What happens if you call `g.draw(box)` instead of `g2.draw(box)`?

**Answer:** The compiler complains that `g` doesn’t have a `draw` method.
Applets

- **Applet**: program that runs inside a web browser
- To implement an applet, use this code outline:

```java
public class MyApplet extends JApplet {
    public void paint(Graphics g) {
        // Recover Graphics2D
        Graphics2D g2 = (Graphics2D) g;
        // Drawing instructions go here
        . . .
    }
}
```
Applets

• This is almost the same outline as for a component, with two minor differences:
  1. You extend JApplet, not JComponent
  2. You place the drawing code inside the paint method, not inside paintComponent

• To run an applet, you need an HTML file with the applet tag

• An HTML file can have multiple applets; add a separate applet tag for each applet

• You view applets with the applet viewer or a Java enabled browser:

  appletviewer RectangleApplet.html
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import javax.swing.JApplet;

/**
 * An applet that draws two rectangles.
 */
public class RectangleApplet extends JApplet
{
    public void paint(Graphics g)
    {
        // Prepare for extended graphics
        Graphics2D g2 = (Graphics2D) g;

        // Construct a rectangle and draw it
        Rectangle box = new Rectangle(5, 10, 20, 30);
        g2.draw(box);
    }
}
20     // Move rectangle 15 units to the right and 25 units down
21     box.translate(15, 25);
22
23     // Draw moved rectangle
24     g2.draw(box);
25 }
26 }
27
1 <applet code="RectangleApplet.class" width="300" height="400">
2 </applet>
Here is my *first applet*:
Applets

An Applet in the Applet Viewer

An Applet in a Web Browser
Ellipses

- `Ellipse2D.Double` describes an ellipse
- This class is an inner class – doesn’t matter to us except for the import statement:
  
  ```java
  import java.awt.geom.Ellipse2D; // no .Double
  ```
- Must construct *and draw* the shape:
  
  ```java
  Ellipse2D.Double ellipse =
      new Ellipse2D.Double(x, y, width, height);
  g2.draw(ellipse);
  ```
An Ellipse

Figure 24  An Ellipse and Its Bounding Box
• To draw a line:

```java
Line2D.Double segment =
    new Line2D.Double(x1, y1, x2, y2);
g2.draw(segment);
```

or,

```java
Point2D.Double from = new Point2D.Double(x1, y1);
Point2D.Double to = new Point2D.Double(x2, y2);
Line2D.Double segment = new Line2D.Double(from, to);
g2.draw(segment);
```
Drawing Text

g2.drawString("Message", 50, 100);

Figure 25  Basepoint and Baseline
Colors

- Standard colors `Color.BLUE`, `Color.RED`, `Color.PINK`, etc.
- Specify red, green, blue between 0 and 255:
  ```java
  Color magenta = new Color(255, 0, 255);
  ```
- Set color in graphics context:
  ```java
  g2.setColor(magenta);
  ```
- Color is used when drawing and filling shapes:
  ```java
  g2.fill(rectangle); // filled with current color
  ```
## Predefined Colors and Their RGB Values

<table>
<thead>
<tr>
<th>Color</th>
<th>RGB Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color.BLACK</td>
<td>0, 0, 0</td>
</tr>
<tr>
<td>Color.BLUE</td>
<td>0, 0, 255</td>
</tr>
<tr>
<td>Color.CYAN</td>
<td>0, 255, 255</td>
</tr>
<tr>
<td>Color.GRAY</td>
<td>128, 128, 128</td>
</tr>
<tr>
<td>Color.DARKGRAY</td>
<td>64, 64, 64</td>
</tr>
<tr>
<td>Color.LIGHTGRAY</td>
<td>192, 192, 192</td>
</tr>
<tr>
<td>Color.GREEN</td>
<td>0, 255, 0</td>
</tr>
<tr>
<td>Color.MAGENTA</td>
<td>255, 0, 255</td>
</tr>
<tr>
<td>Color.ORANGE</td>
<td>255, 200, 0</td>
</tr>
<tr>
<td>Color.PINK</td>
<td>255, 175, 175</td>
</tr>
<tr>
<td>Color.RED</td>
<td>255, 0, 0</td>
</tr>
<tr>
<td>Color.WHITE</td>
<td>255, 255, 255</td>
</tr>
<tr>
<td>Color.YELLOW</td>
<td>255, 255, 0</td>
</tr>
</tbody>
</table>
Alien Face

Figure 26
An Alien Face

Hello, World!
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import javax.swing.JComponent;

/**
 * A component that draws an alien face
 */
public class FaceComponent extends JComponent {
  public void paintComponent(Graphics g) {
    // Recover Graphics2D
    Graphics2D g2 = (Graphics2D) g;
  }

  // Continued
// Draw the head
Ellipse2D.Double head = new Ellipse2D.Double(5, 10, 100, 150);
g2.draw(head);

// Draw the eyes
g2.setColor(Color.GREEN);
Rectangle eye = new Rectangle(25, 70, 15, 15);
g2.fill(eye);
eye.translate(50, 0);
g2.fill(eye);

// Draw the mouth
Line2D.Double mouth = new Line2D.Double(30, 110, 80, 110);
g2.setColor(Color.RED);
g2.draw(mouth);

// Draw the greeting
g2.setColor(Color.BLUE);
g2.drawString("Hello, World!", 5, 175);
import javax.swing.JFrame;

public class FaceViewer {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setSize(150, 250);
        frame.setTitle("An Alien Face");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

        frame.setVisible(true);
    }
}
Self Check 2.53

Give instructions to draw a circle with center (100, 100) and radius 25.

Answer:

```java
g2.draw(new Ellipse2D.Double(75, 75, 50, 50));
```
Self Check 2.54

Give instructions to draw a letter "V" by drawing two line segments.

Answer:

```java
Line2D.Double segment1 = new Line2D.Double(0, 0, 10, 30);
g2.draw(segment1);
Line2D.Double segment2 = new Line2D.Double(10, 30, 20, 0);
g2.draw(segment2);
```
Self Check 2.55

Give instructions to draw a string consisting of the letter "V".

Answer:

g2.drawString("V", 0, 30);
Self Check 2.56

What are the RGB color values of `Color.BLUE`?

Answer: 0, 0, and 255
How do you draw a yellow square on a red background?

**Answer:** First fill a big red square, then fill a small yellow square inside:

```java
    g2.setColor(Color.RED);
    g2.fill(new Rectangle(0, 0, 200, 200));
    g2.setColor(Color.YELLOW);
    g2.fill(new Rectangle(50, 50, 100, 100));
```
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:

  ```java
  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
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 a CarComponent
```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;
    }
}
```
/**
* 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);
48     // The bottom of the rear windshield
49     Point2D.Double r4
50       = new Point2D.Double(xLeft + 50, yTop + 10);
51
52     Line2D.Double frontWindshield
53       = new Line2D.Double(r1, r2);
54     Line2D.Double roofTop
55       = new Line2D.Double(r2, r3);
56     Line2D.Double rearWindshield
57       = new Line2D.Double(r3, r4);
58
59     g2.draw(body);
60     g2.draw(frontTire);
61     g2.draw(rearTire);
62     g2.draw(frontWindshield);
63     g2.draw(roofTop);
64     g2.draw(rearWindshield);
65   }
66 }

ch03/car/Car.java (cont.)
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);
    }
}
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 2.58

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

**Answer:** CarComponent
Self Check 2.59

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

```java
g2.fill(frontTire);
g2.fill(rearTire);
```
Self Check 2.60

How do you make the cars twice as big?

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