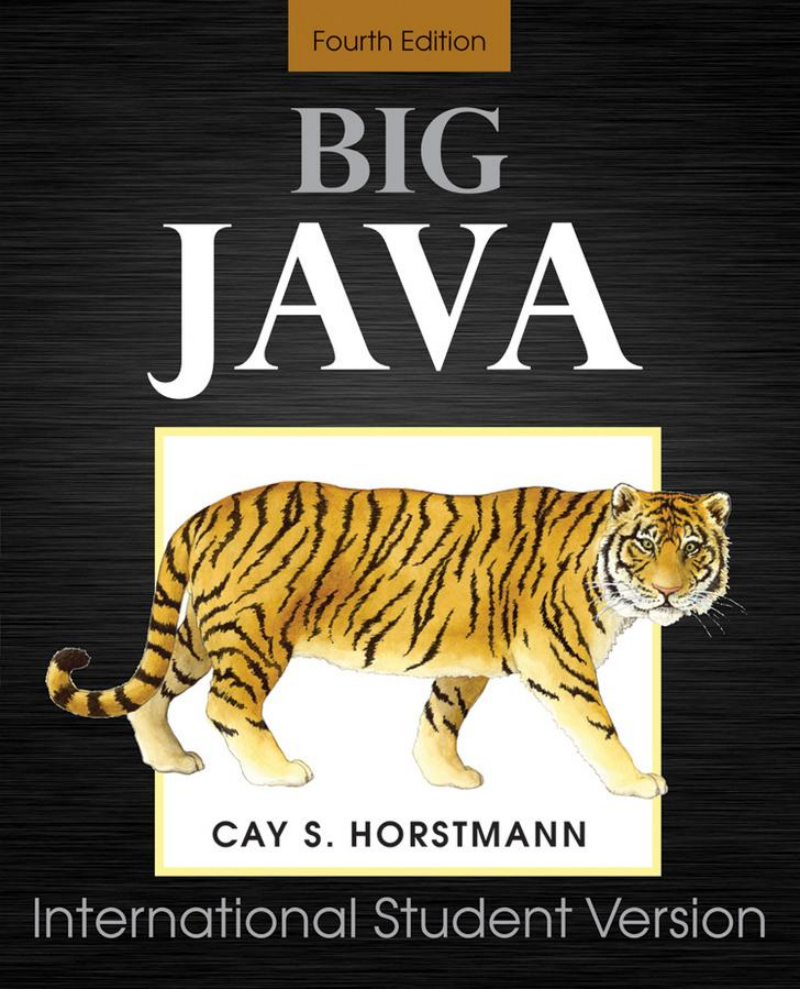


Chapter 2 – Einführung in Objekte und Klassen

BFH, M. Hudritsch based on Big Java by Cay Horstmann

Einführung in Objekte

- Lernen was Datentypen sind
- Lernen was Variablen sind
- Konzept verstehen von Klassen und Objekten
- Methoden aufrufen können
- Parameter und Rückgabewerte verstehen
- Die API Dokumentation verwenden können
- T Ein Testprogram implementieren können
- Den Unterschied zw. Objekt und einer Referenz auf ein Objekt verstehen.

Typen

- Ein **Typ** definiert ein Set an Werten und Operationen die auf mit den Werten ausgeführt werden können.
- **Beispiel**:
  - `13` *hat den Typ* `int`
  - `"Hello, World"` *hat den Typ* `String`
  - `System.out` *hat den Typ* `PrintStream`

Ausgabe Konsole

- Java hat separate Typen für **integer (Ganzahl)** und **floating-point (Fliesskomma) Zahlen**
  - *Der* `double` *Typ ist für Fliesskommazahlen*
- Ein Wert wie `13` oder `1.3` in einem Java Programm wir **Zahlenliteral (number literal)** genannt.

Zahlenliterale (Number Literals)

Table 1 Number Literals in Java

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

Zahlentypen (number types)

- Zahlentype sind **primitive Typen**
- Zahlen sind keine Objekte
- Zahlen können mit arithmetischen Operatoren wie +,-,* und / verknüpft werden.

Selbst-Check 2.1

Was ist der Typ der Werte `0` und `"0"`?

Antwort: `int` und `String`.

Selbst-Check 2.2

Was für einen Zahlentyp würden Sie verwenden, um die Fläche eines Kreises zu speichern?

Antwort: `double`.

Selbst-Check 2.3

Warum ist der Ausdruck `13.println()` ein Fehler?

Antwort: Ein `int` ist kein Objekt und man kann keine Methode darauf aufrufen.

Selbst-Check 2.4

Schreiben Sie einen Ausdruck, der den Mittelwert der Werte `x` und `y` berechnet.

Antwort: `(x + y) * 0.5`

Besser Multiplikation statt Division

Variablen

- **Variablen** werden verwendet, um Werte zu speichern, die Sie später noch einmal verwenden wollen.
- Variablen haben einen Typ, einen Namen und ein Wert:

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

- Variablen können anstelle der Werte verwendet werden:

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

Variablen

- Es ist ein Fehler, Werte zu speichern, die nicht zum Typ der Variable passt.

```
String greeting = 20; // ERROR: Types don't match
```

Variablen Deklaration

Table 2 Variable Declarations in Java

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

Bezeichner (Identifiers)

- **Bezeichner:** Name einer Variable, Methode oder Klasse
- **Regeln** für Bezeichner in Java:
  - *Kann aus Buchstaben, Zahlen, Unterstrich (underscore) und Dollar-Zeichen (`$`) bestehen*
  - *Kann nicht mit einer Zahl beginnen*
  - *Lehrzeichen sind nicht erlaubt*
  - *Reservierte Worte aus der Java Sprache, wie z.B. public können nicht verwendet werden*
  - *Gross- und Kleinschreibung wird berücksichtigt (case sensitive)*

Bezeichner (Identifiers)

- In Java beginnen **Variablen** konventionell mit **Kleinbuchstaben**
  - *"Camel case": Bei zusammengehängten Bezeichnern wird der erste Buchstaben des nächsten Wortes gross geschrieben:* `farewellMessage`
- Bezeichner für **Klassen** beginnen konventionell mit **Grossbuchstaben**
- Beginnen Sie keine Variablen mit dem $-Zeichen. Die werden konventionell für generierte Variablen verwendet.

Syntax 2.1 Variablen Deklaration

Syntax

typeName variableName = value;
or
typeName variableName;

Example

```
String greeting = "Hello, Dave!";
```

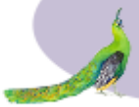
See the rules for and table of examples of valid names.

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.

Use a descriptive variable name.



Variablen Namen

Table 3 Variable Names in Java

Variable Name	Comment
`farewellMessage`	Use "camel case" for variable names consisting of multiple words.
`x` besser übersichtlich als zu lange Namen. Besonders bei Sachen, die sehr oft verwendet werden, Durchlaufvariablen	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.
⚠ `Greeting`	**Caution:** Variable names are case-sensitive. This variable name is different from `greeting`.
⊘ `6pack`	**Error:** Variable names cannot start with a number.
⊘ `farewell message`	**Error:** Variable names cannot contain spaces.
⊘ `public`	**Error:** You cannot use a reserved word as a variable name.

Selbst-Check 2.5

Welche der nachfolgenden Bezeichner sind korrekt?

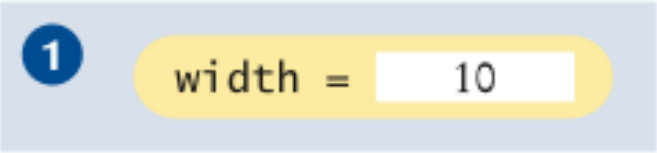
```
Greeting1
g
void
101dalmatians
Hello, World
<greeting>
```

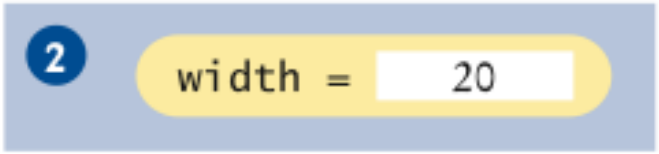
Antwort: Nur die ersten beiden.

Zuweisungsoperator

- Zuweisungsoperator: =
- Wird gebraucht, um den Wert einer Variable zu ändern:

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





Uninitialisierte Variablen

- Es ist ein Fehler eine Variable zu verwenden, die noch nie einen Wert zugewiesen bekam:

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

Figure 2
An Uninitialized Variable

- Abhilfe: Weisen Sie einen Wert zu vor der Verwendung:

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

- Noch besser, initialisieren Sie direkt hinter der Deklaration:

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

Syntax 2.2 Zuweisung

Syntax *variableName* = *value*;

Example

```
double width = 30;
.
.
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.

Zuweisung

- Die rechte Seite vom Symbol `=` kann eine mathematischer Ausdruck sein:

```
width = width + 10;
```

- Sprich:

  1. Berechne den Wert von `width + 10`

  2. Weise das Resultat der Variable `width` *zu.*

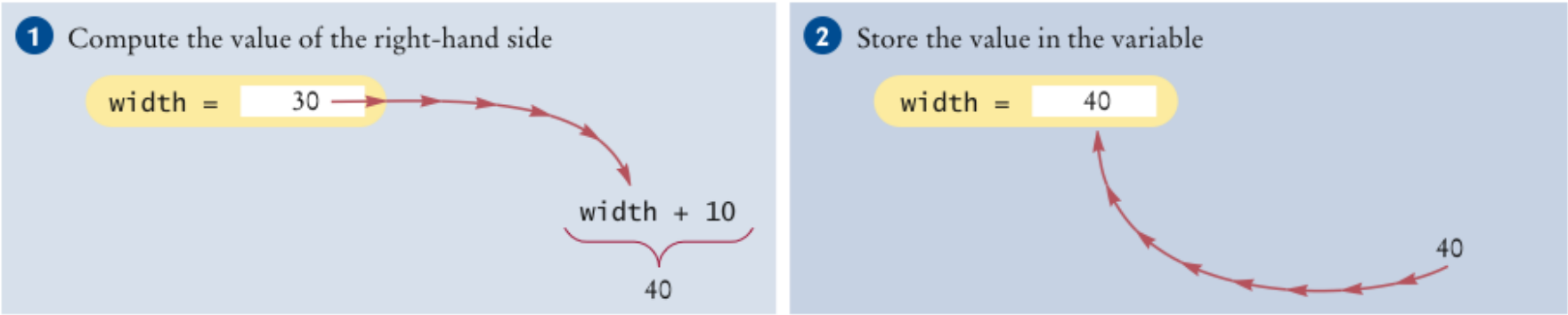


Figure 3 Executing the Statement `width = width + 10`

Selbst-Check 2.7

Ist `12 = 12` ein gültiger Ausdruck in der Java Sprache?

Antwort: Nein, links vom Zuweisungsoperator = muss eine Variable sein.

Selbst-Check 2.8

Wie ändern Sie den Wert der `greeting` Variable zu `"Hello, Nina!"`?

Antwort:

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

Beachte, dass

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

nicht korrekt ist. Sie versuchen damit eine zweite Variable zu erstellen. Diese ist aber nicht erlaubt.

Objekte und Klassen

- **Objekte:** Eine Einheit, die Sie in Ihrem Programm über Methoden manipulieren können.
- Jedes Objekt ist von einer bestimmten **Klasse**
- Beispiel: `System.out` gehört zur Klasse `PrintStream`

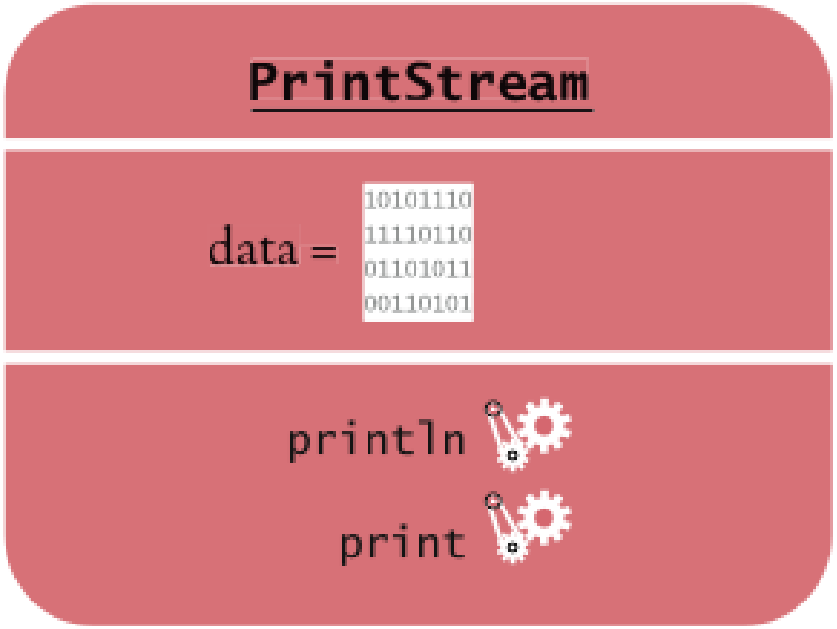


Figure 4 Representation of the `System.out` Object

Methoden

- **Methode:** Eine Sequenz von Befehlen die auf die Daten des Objekts zugreifen können
- Sie manipulieren Objekte, indem Sie dessen Methoden aufrufen
- **Klasse:** Deklariert die Methoden die auf einem Objekt angewendet werden können.
- **Public Interface:** Definiert welche Methoden von Aussen aufgerufen werden können.

Überladene Methoden

- **Überladene Methode:** Wenn eine Klasse 2 Methoden mit demselben Namen hat aber mit unterschiedlichen Parametern
- Beispiel: Die `PrintStream` Klasse deklariert eine 2. Methode für eine Integerzahl:

```
public void println(int output)
```

String Methoden

- `length`: Zählt die Anzahl Zeichen im String:

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

- `toUpperCase`: Erzeugt ein String-Objekt mit allen Buchstaben als Grossbuchstaben:

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

Selbst-Check 2.9

Wie bestimmen Sie die Länge des Strings `"Mississippi"`?

Antwort: `river.length()` oder `"Mississippi".length()`

Ein String in Anführungszeichen wird automatisch direkt zum Objekt, sodass es auch Methoden aufrufen kann. Anders als bei Zahlen, die niemals Objekte sind und keine Methoden aufrufen können!

Selbst-Check 2.10

Wie geben Sie den folgenden String in Grossbuchstaben aus?
`"Hello, World!"`?

Antwort:

```
System.out.println(greeting.toUpperCase());
```

von innen nach aussen wie in der Mathematik

Selbst-Check 2.11

Ist folgender Aufruf gültig: `river.println()`?
Warum oder warum nicht?

Antwort: Nicht gültig. `river` ist wahrscheinlich ein `String`. `println` ist keine Methode der `String` Klasse.

Parameter

- **Parameter:** Eingangswert in eine Methode
- **Implizite Parameter:** Das Objekt das zur Methode gehört:

```
System.out.println(greeting)
```

- **Explizite Parameter:** Alle übergebenen Parameter:

```
System.out.println(greeting)
```

- Es braucht nicht immer einen expliziten Parameter:

```
greeting.length() // has no explicit
 parameter
```

Rückgabewerte (Return values)

- **Rückgabewert:** Der von einer Funktion zurückgegebene Wert:

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

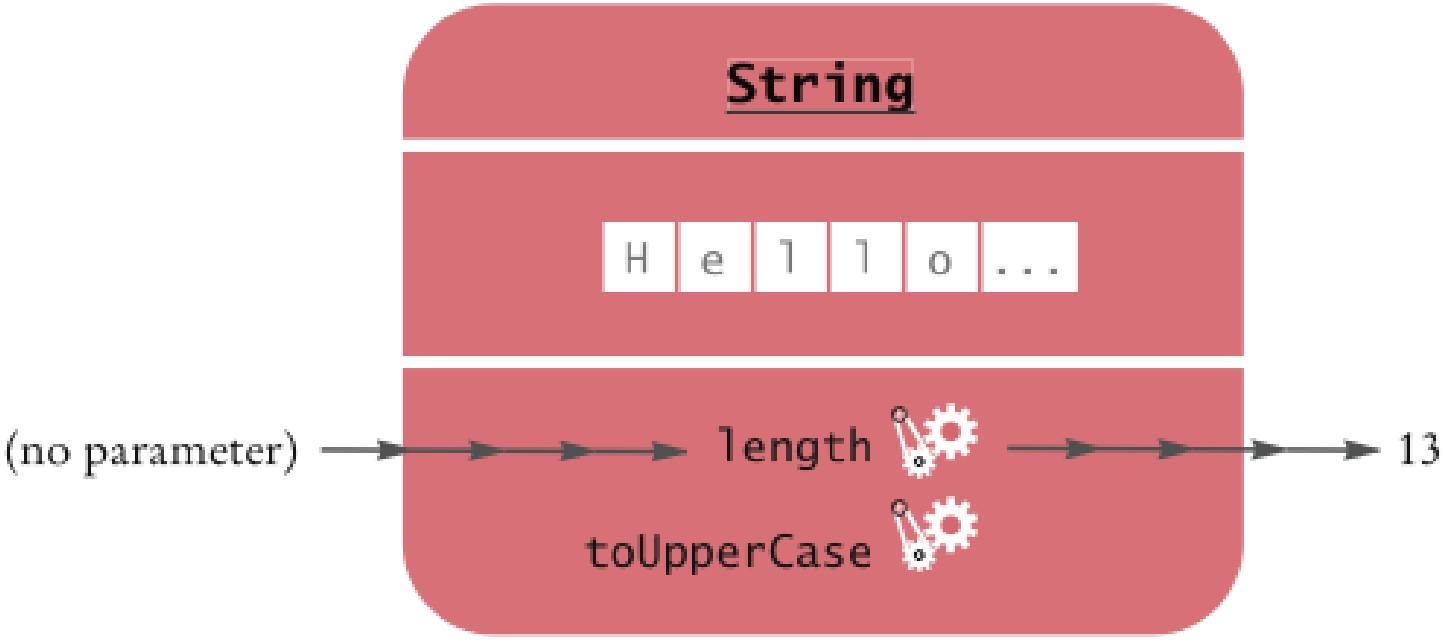


Figure 7 Invoking the length Method on a String Object

Rückgabewerte weitergeben

- Rückgabewerte können direkt an eine andere Methode weitergegeben werden:

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

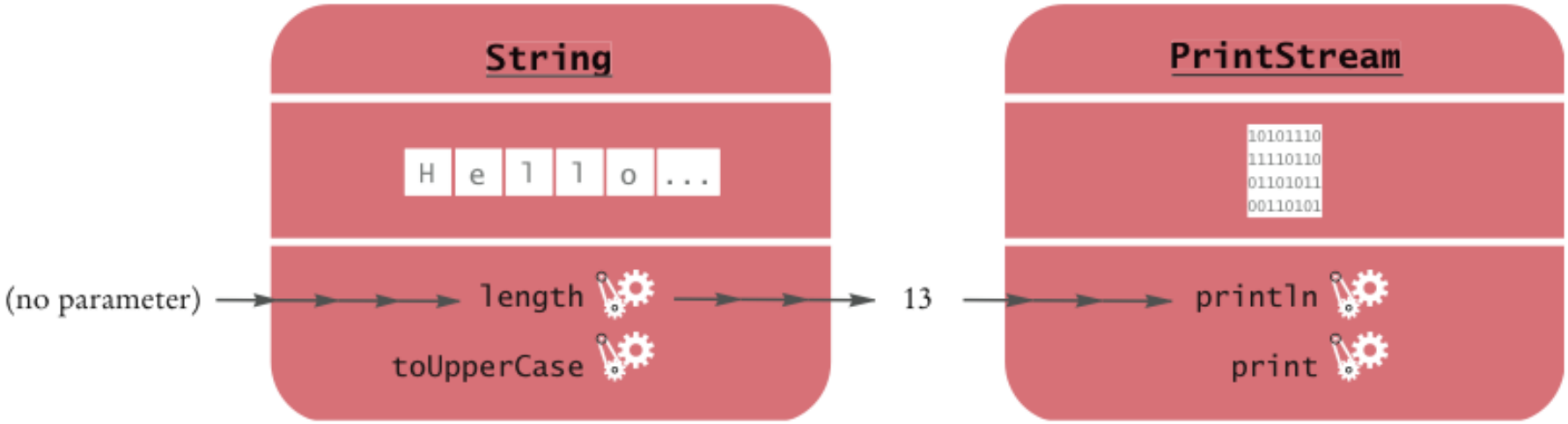


Figure 8 Passing the Result of a Method Call to Another Method

- Nicht alle Methoden haben Rückgabewerte: `println`

Ein etwas komplexerer Aufruf

- Die `String` Methode `replace` sucht eine String und ersetzt ihn durch einen anderen:

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

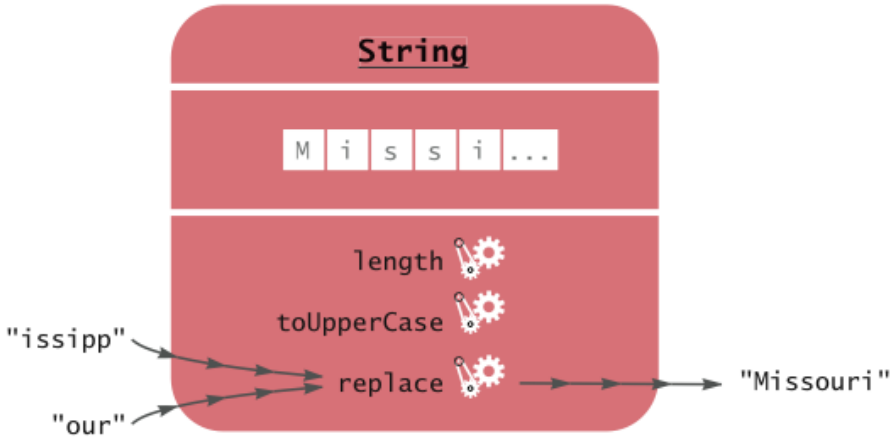


Figure 9 Calling the replace Method

- Diese Methode hat:
  - *1 impliziten Parameter: Den String* `"Mississippi"`
  - *2 explizite Parameter: Die Strings* `"issipp"` *und* `"our"`
  - *1 Rückgabewert: Den String* `"Missouri"`

Selbst-Check 2.12

Was sind die expliziten, die impliziten Parameter und der Rückgabewert von `river.length()`?

Antwort: Der implizite Parameter ist `river`.
Es hat kein expliziten Parameter.
Der Rückgabewert ist 11.

Selbst-Check 2.15

Wie ist die Methode `toUpperCase` in der `String` Klasse deklariert?

Antwort: Als `public String toUpperCase()`, ohne explizite Parameter und mit Rückgabetyp `String`.

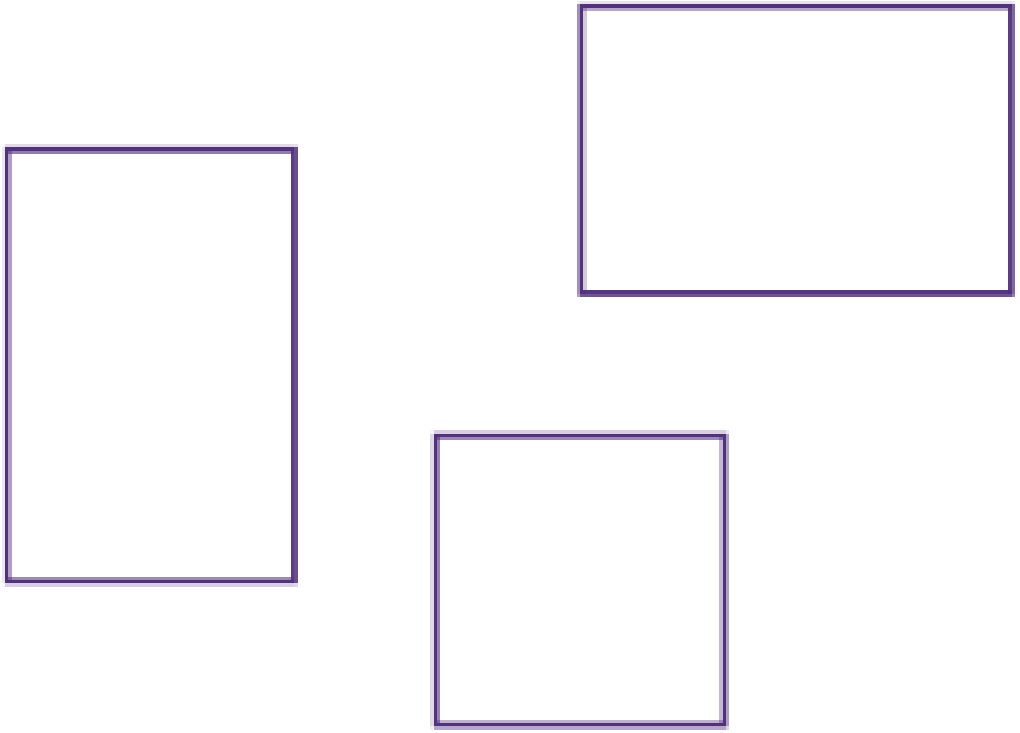
Chapter 2.6: Object Construction

BFH, M. Hudritsch based on Big Java by Cay Horstmann

Rectangular Shapes and Rectangle Objects

- Objects of type `Rectangle` *describe* rectangular shapes:

Figure 10
Rectangular Shapes



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:

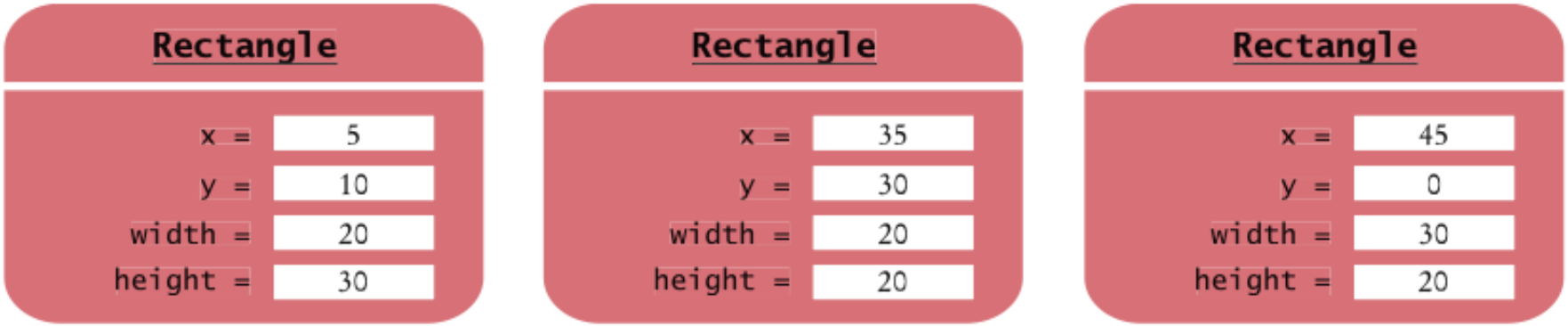


Figure 11 Rectangle Objects

Constructing Objects

```
new Rectangle(5, 10, 20, 30)
```

Für Objekte, also nicht-primitive Datentypen

- 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* Generiert durch Konstruktor der Klasse
  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*

überladene Klassen (Konstruktoren)

- Some classes let you construct objects in multiple ways:

```
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.

Construction parameters

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

Usually, you save the constructed object in a variable.

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

toString -Methode wird hier aufgeführt, gibt ansonsten Java Objektname und Speicheradresse aus

Supply the parentheses even when there are no parameters.

You can also pass the constructed object to a method.

Self Check 2.7

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

Answer:

```
new Rectangle(90, 90, 20, 20)
```

Self Check 2.8

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

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

Answer:

0

Defaultwert

Accessor and Mutator Methods

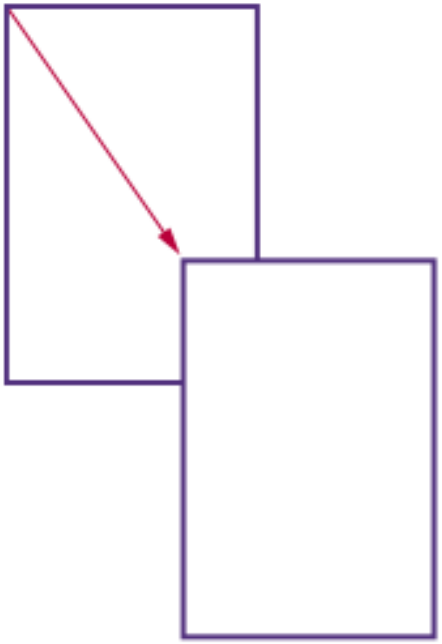
- **Accessor method**: does not change the state of its implicit parameter:

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

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

```
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.

The API Documentation

- **API:** Application Programming Interface
- **API documentation:** lists classes and methods in the Java library
- http://docs.oracle.com/javase/7/docs/api/

The API Documentation of the Standard Java Library

Overview (Java Platfo ×

docs.oracle.com/javase/7/docs/api/

Other bookmarks

Java™ Platform Standard Ed. 7

All Classes

Packages

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractExecutorService
AbstractInterruptibleChannel

docs.oracle.com/javase/7/docs/.../AbstractCellEditor.html

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 7

Prev Next Frames No Frames

Java™ Platform, Standard Edition 7 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring

The API Documentation for the Rectangle Class

Rectangle (Java Platf

docs.oracle.com/javase/7/docs/api/

Other bookmarks

Java™ Platform
Standard Ed. 7

All Classes

Packages

Receiver
Rectangle
Rectangle2D
Rectangle2D.Double
Rectangle2D.Float
RectangularShape
RecursiveAction
RecursiveTask
ReentrantLock
ReentrantReadWriteLock
ReentrantReadWriteLock.ReadLock
ReentrantReadWriteLock.WriteLock
Ref
RefAddr
Reference
Reference
Reference
Referenceable
ReferenceQueue

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 7

Prev Class **Next Class** Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.awt

Class Rectangle

java.lang.Object
  java.awt.geom.RectangularShape
    java.awt.geom.Rectangle2D
      java.awt.Rectangle

All Implemented Interfaces:

Shape, Serializable, Cloneable

Direct Known Subclasses:

DefaultCaret

```
public class Rectangle
extends Rectangle2D
implements Shape, Serializable
```

Method Summary

Rectangle (Java Platf

docs.oracle.com/javase/7/docs/api/

Java™ Platform
Standard Ed. 7

All Classes

Packages

Receiver
Rectangle
Rectangle2D
Rectangle2D.Double
Rectangle2D.Float
RectangularShape
RecursiveAction
RecursiveTask
ReentrantLock
ReentrantReadWriteLock
ReentrantReadWriteLock.ReadLock
ReentrantReadWriteLock.WriteLock
Ref
RefAddr
Reference
Reference
Reference
Referenceable
ReferenceQueue

Method Summary

Methods

Modifier and Type	Method and Description
void	**add**(int newx, int newy) Adds a point, specified by the integer arguments newx,newy to the bounds of this Rectangle.
void	**add**(Point pt) Adds the specified Point to the bounds of this Rectangle.
void	**add**(Rectangle r) Adds a Rectangle to this Rectangle.
boolean	**contains**(int x, int y) Checks whether or not this Rectangle contains the point at the specified location (x,y).
boolean	**contains**(int X, int Y, int W, int H) Checks whether this Rectangle entirely contains the Rectangle at the specified location (X,Y) with the specified dimensions (W,H).
boolean	**contains**(Point p) Checks whether or not this Rectangle contains the specified Point.
boolean	**contains**(Rectangle r)

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)

translate

```
public void translate(int dx,
              int dy)
```

Translates this `Rectangle` the indicated distance, to the right along the X coordinate axis, and downward along the Y coordinate axis.

Parameters:

`dx` - the distance to move this `Rectangle` along the X axis

`dy` - the distance to move this `Rectangle` along the Y axis

See Also:

`setLocation(int, int), setLocation(java.awt.Point)`

Packages

- **Package**: a collection of classes with a related purpose
- Import library classes by specifying the package and class name:

```
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 `import` *packageName.ClassName*;

Example

Package name

Class name

```
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 2.9

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 2.10

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.

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.

ch02/rectangle/MoveTester.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");
   }
}
```

ch02/rectangle/MoveTester.java (cont.)

Program Run:

```
x: 20
Expected: 20
y: 35
Expected: 35
```

Self Check 2.13

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.

Introduction to classes (Chapter 3 in the PDF version

- 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

Figure 1 A Tally Counter

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

```
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:

```
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

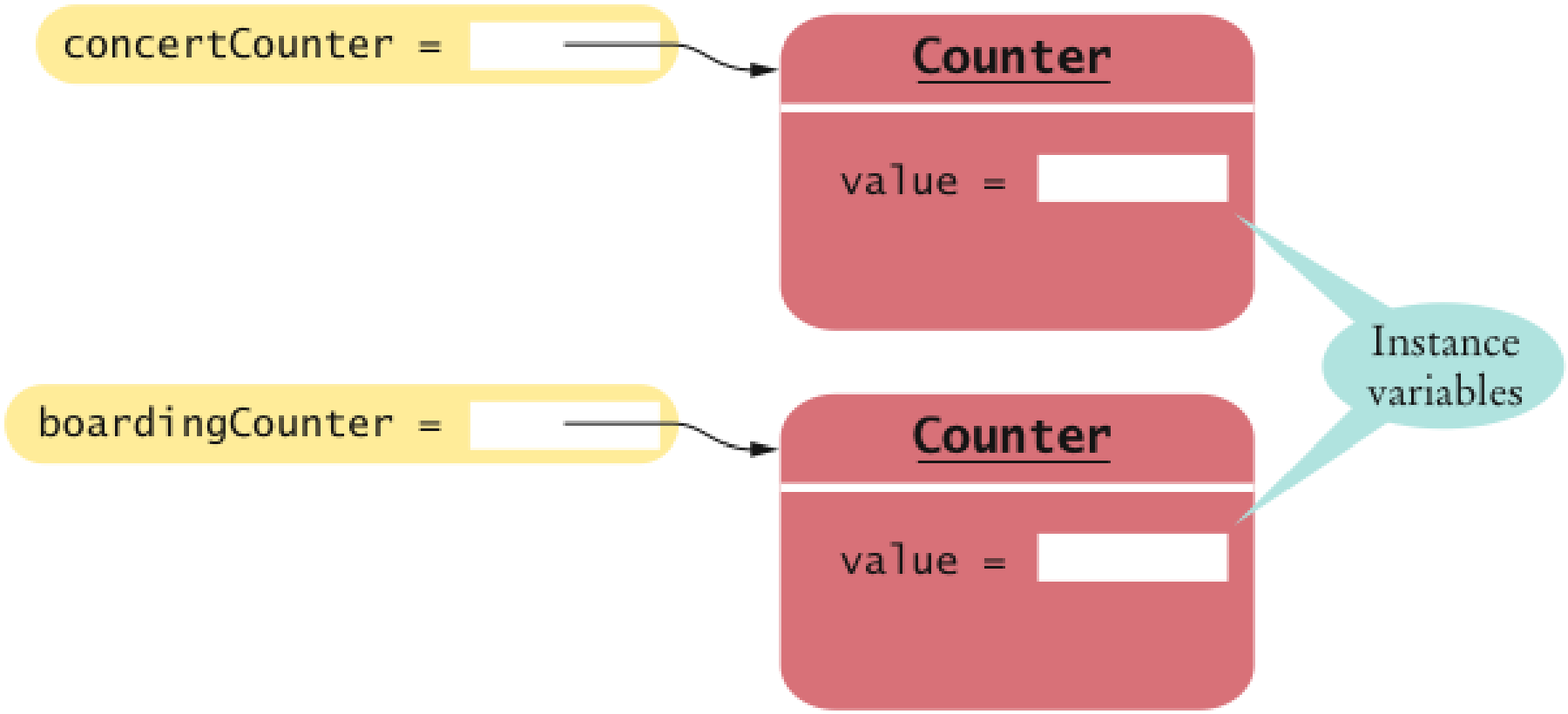


Figure 2 Instance Variables

Syntax 2.5 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

2.7 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());
```

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()`

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 2.6 Class Declaration

Syntax

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

Example

```
public class Counter
{
    private int value;

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

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

Public interface

Private implementation

Self Check 2.16

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

Answer:

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

Self Check 2.17

What is wrong with this sequence of statements?

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

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

Self Check 2.18

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

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

Commenting the Public Interface

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

Class Comment

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

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

Javadoc Method Summary

You can generate the Javadoc in Eclipse with:
Project > Generate Javadoc …

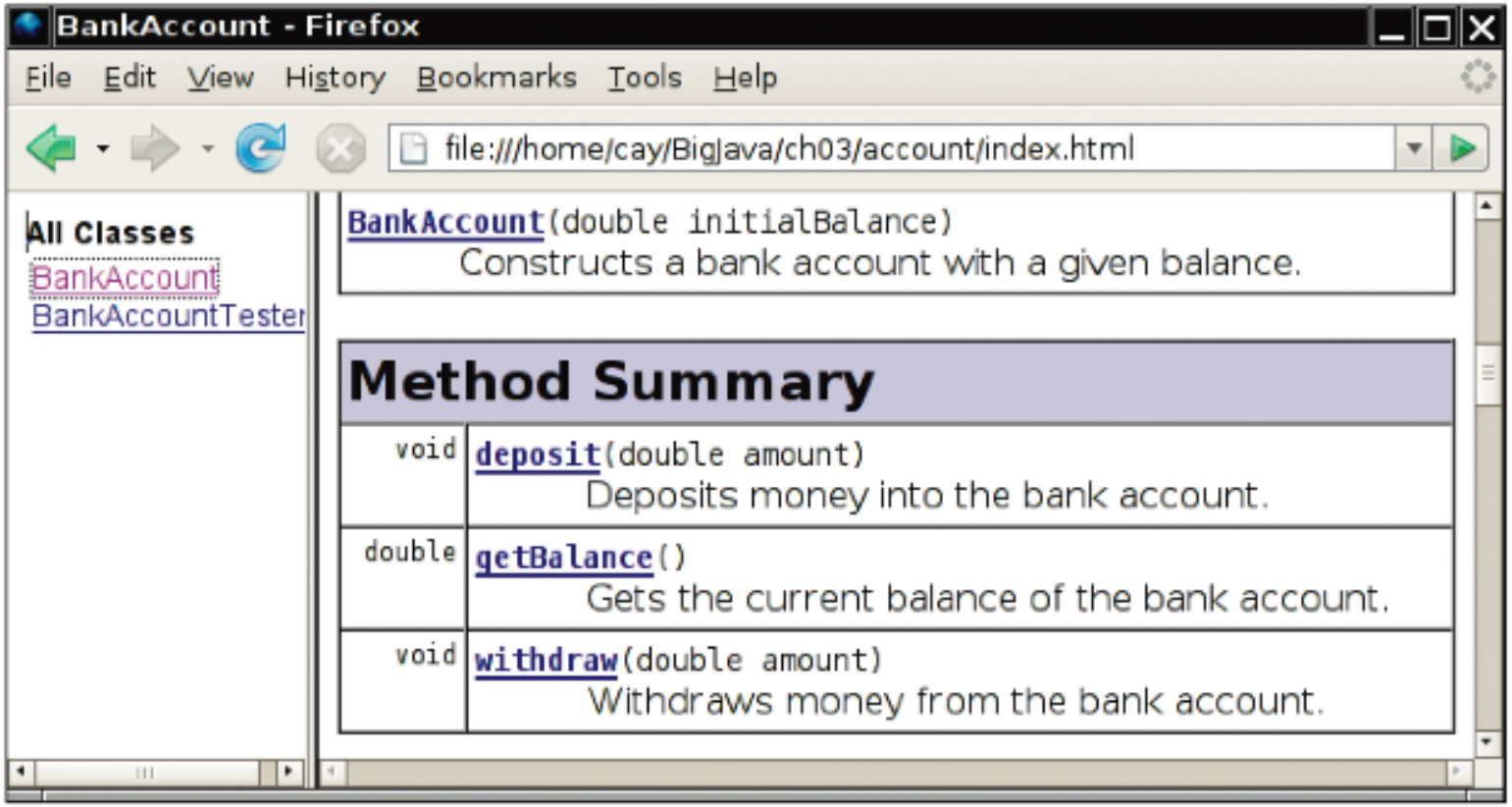


Figure 3 A Method Summary Generated by javadoc

Javadoc Method Detail

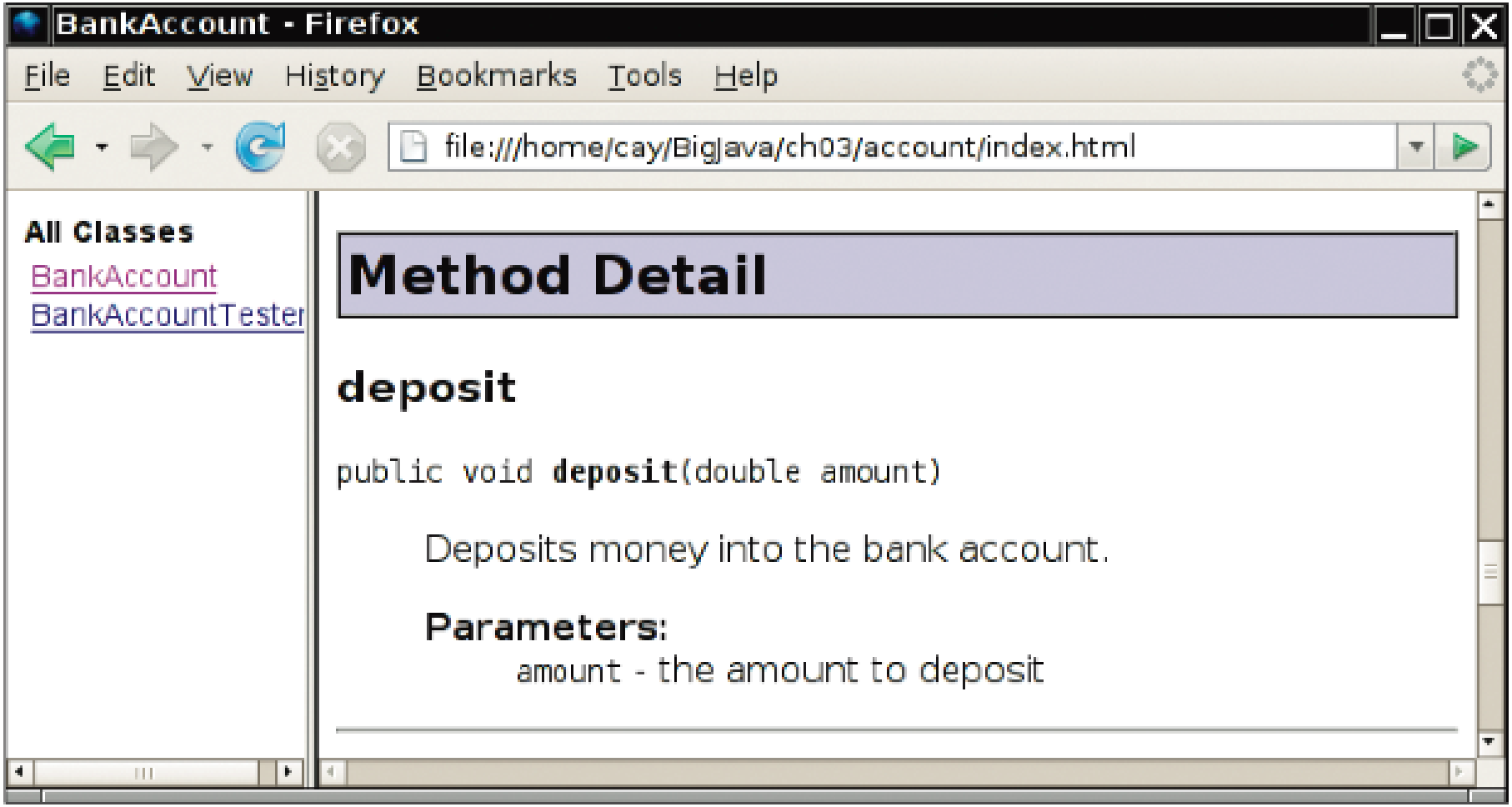


Figure 4 Method Detail Generated by javadoc

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 2.7 Method Declaration

Syntax

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

Example

This method does not return a value.

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

A mutator method modifies an instance variable.

These methods are part of the public interface.

This method has no parameters.

```
public double getBalance()
{
   return balance;
}
```

An accessor method returns a value.

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;
  }
  ```

ch02/account/BankAccount.java

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

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

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

Continued

ch02/account/BankAccount.java (cont.)

```

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

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

```

Continued

ch02/account/BankAccount.java (cont.)

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

Self Check 2.19

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

Answer:

An instance variable

```
private int accountNumber;
```

needs to be added to the class.

Self Check 2.20

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

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

      momsSavings.balance = 0;
   }
}
```

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

2.9 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*

ch02/account/BankAccountTester.java

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

Program Run:

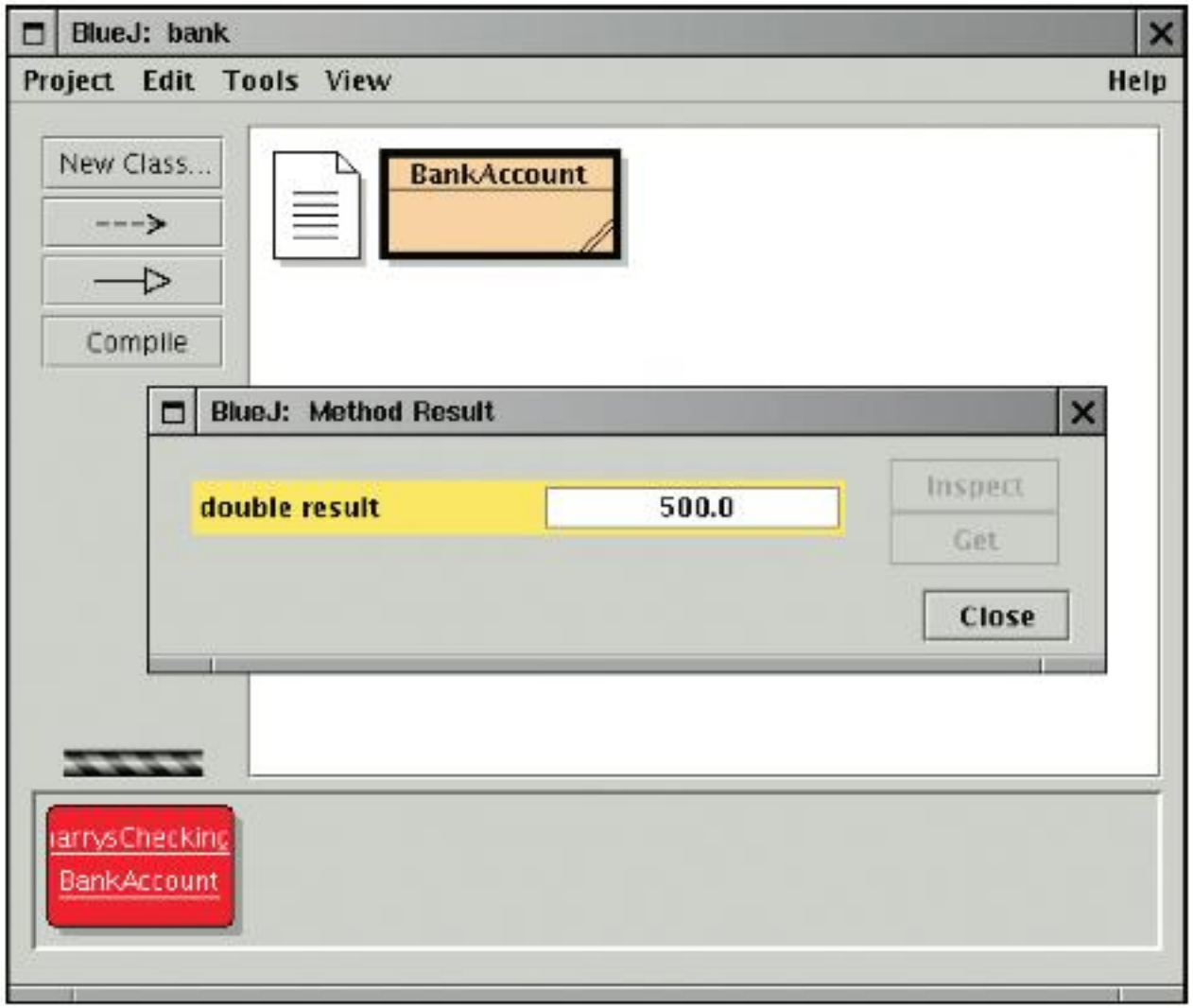
```
1500
Expected: 1500
```

Unit Testing (cont.)

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

Testing With BlueJ

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



Self Check 2.23

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 2.24

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.

2.10 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

Self Check 2.25

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

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

Self Check 2.26

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.

2.11 Object References

- **Object reference:** describes the location of an object
- The `new` operator returns a reference to a new object:

```
Rectangle box = new Rectangle();
```

- Multiple object variables can refer to the same object:

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

- Primitive type variables ≠ object variables

Object Variables and Number Variables

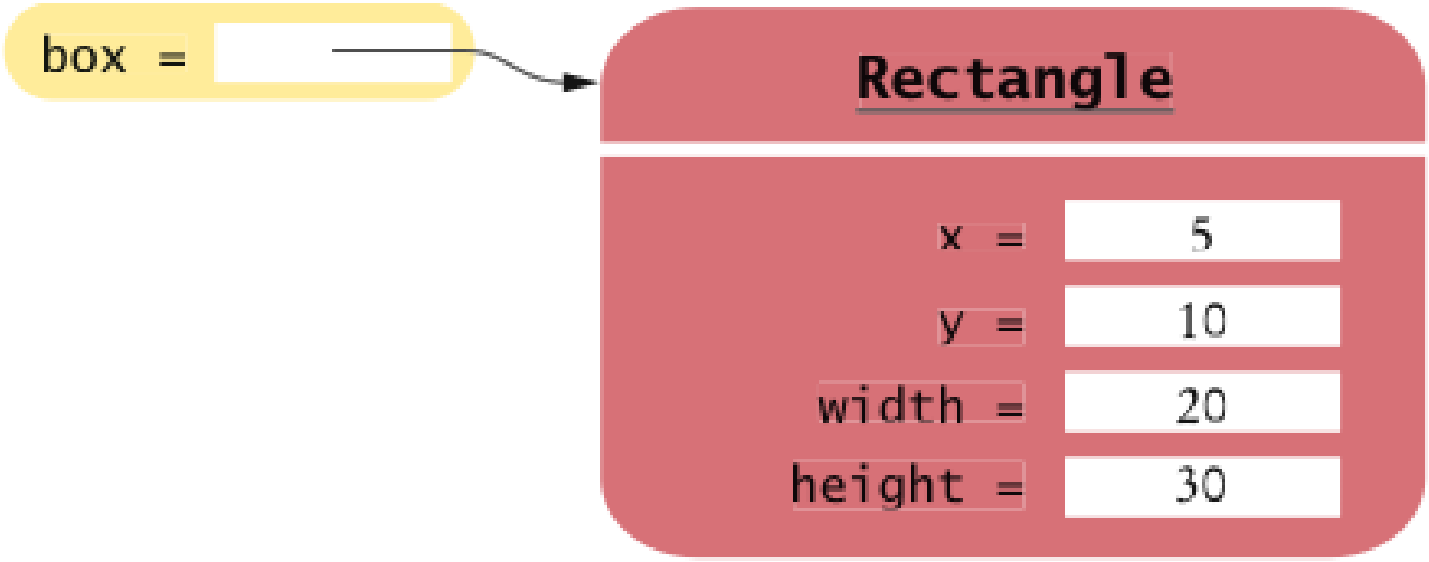


Figure 17 An Object Variable Containing an Object Reference

Object Variables and Number Variables

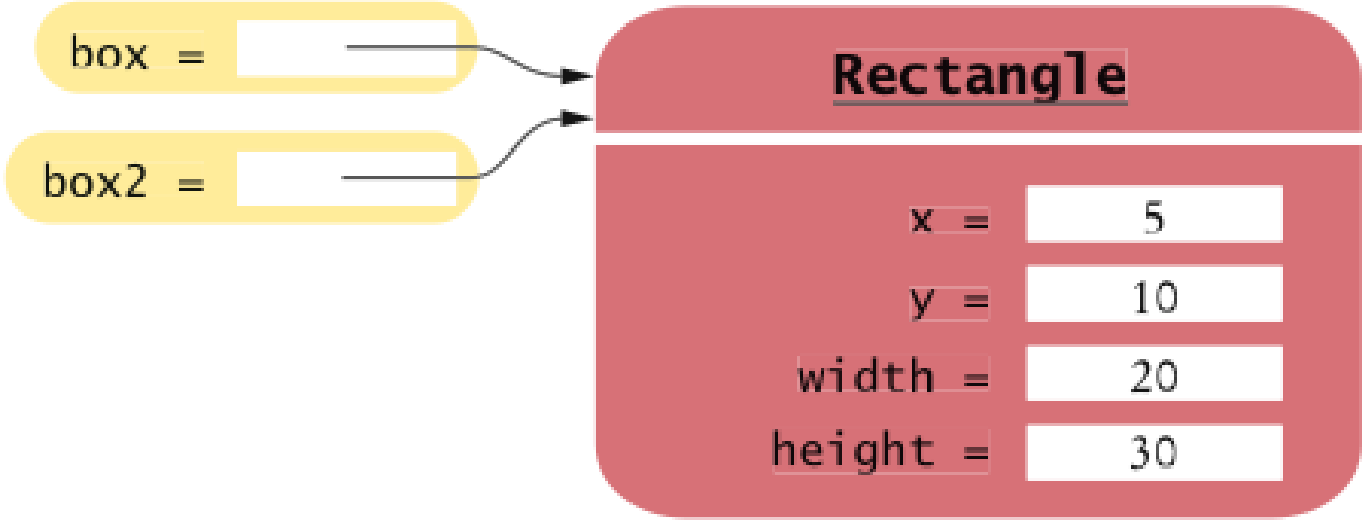


Figure 18 Two Object Variables Referring to the Same Object

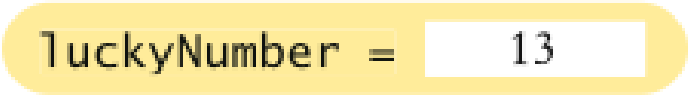
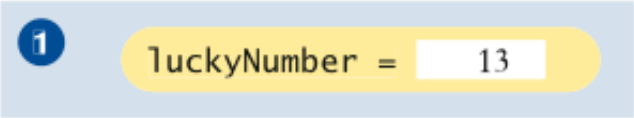


Figure 19 A Number Variable Stores a Number

Copying Numbers

```
int luckyNumber = 13;
```

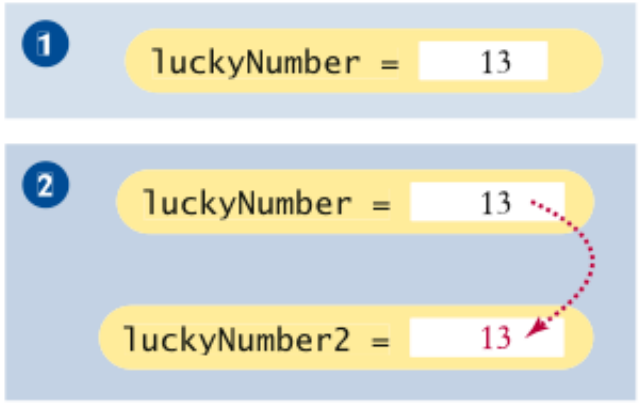
Figure 20
Copying Numbers



Copying Numbers (cont.)

```
int luckyNumber = 13;
int luckyNumber2 = luckyNumber;
```

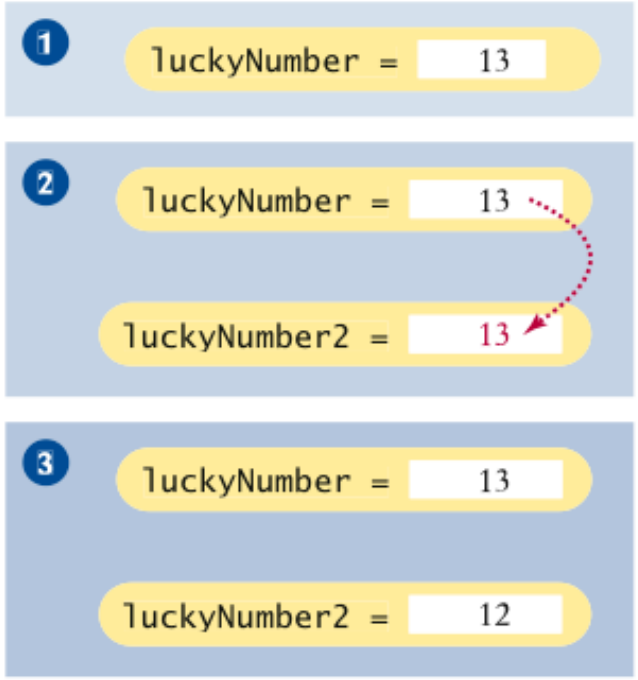
Figure 20
Copying Numbers



Copying Numbers (cont.)

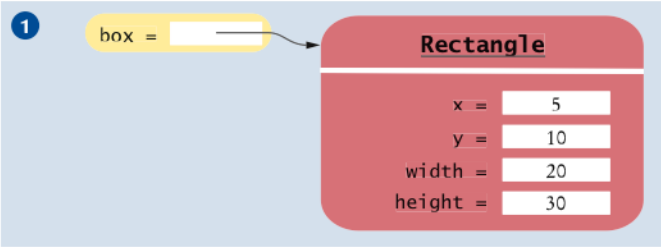
```
int luckyNumber = 13;  ❶
int luckyNumber2 = luckyNumber;  ❷
luckyNumber2 = 12;  ❸
```

Figure 20
Copying Numbers



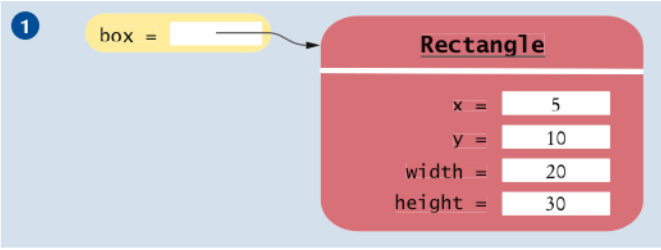
Copying Object References

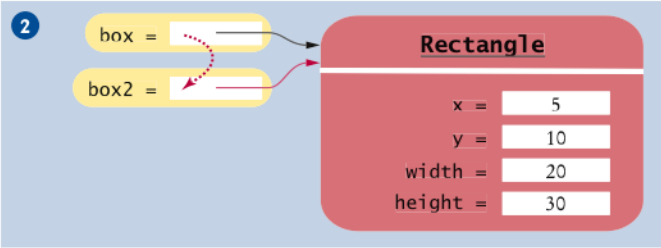
```
Rectangle box = new Rectangle(5, 10, 20, 30);
```



Copying Object References (cont.)

```
Rectangle box = new Rectangle(5, 10, 20, 30);  ❶
Rectangle box2 = box;  ❷
```





Copying Object References (cont.)

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

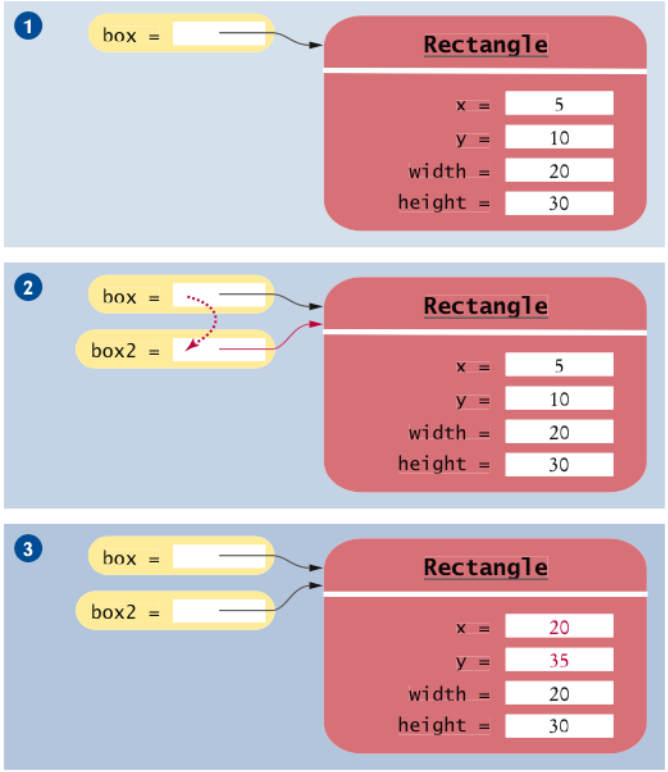


Figure 21 Copying Object References

Self Check 2.27

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

Answer: Now `greeting` and `greeting2` both refer to the same `String` object.

Self Check 2.28

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.

2.12 Implicit Parameter

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

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

- In the call

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

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

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

Implicit Parameters and `this`

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

  actually means

  `this.balance = this.balance + amount;`

- When you refer to an instance variable in a method, the compiler automatically applies it to the `this` reference

Implicit Parameters and `this`

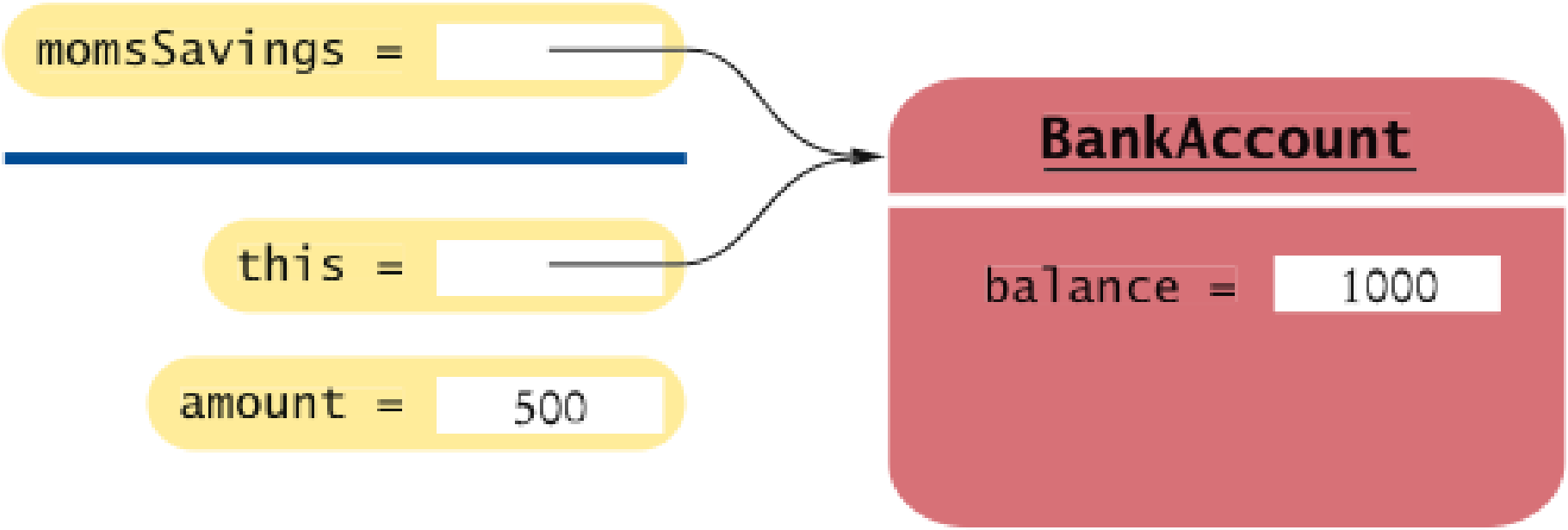


Figure 6 The Implicit Parameter of a Method Call

Implicit Parameters and `this`

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

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

Implicit Parameters and `this`

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

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

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

Implicit Parameters and `this`

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

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

Self Check 2.29

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 2.30

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

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

Self Check 2.31

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

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

2.13 Graphical Applications and Frame Windows

To show a frame:

1. Construct an object of the `JFrame` class:

```
JFrame frame = new JFrame();
```

2. Set the size of the frame:

```
frame.setSize(300, 400);
```

3. If you'd like, set the title of the frame:

```
frame.setTitle("An Empty Frame");
```

4. Set the "default close operation":

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

5. Make the frame visible:

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

A Frame Window

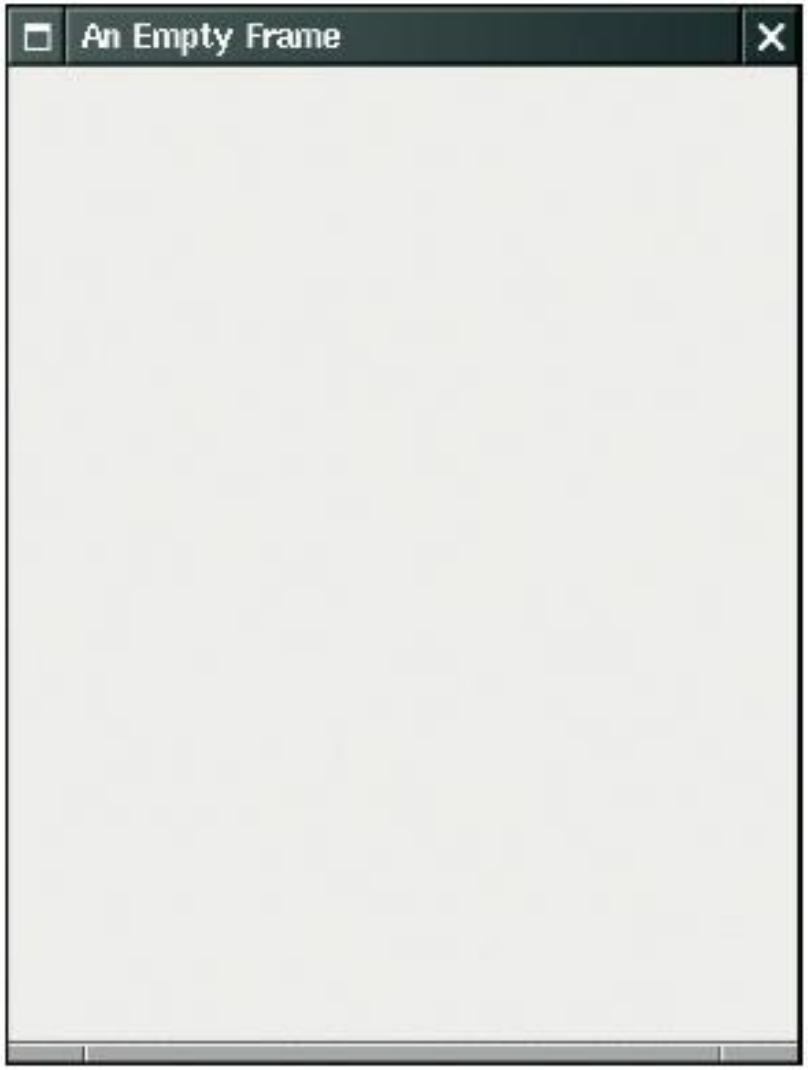


Figure 22
A Frame Window

ch02/emptyframe/EmptyFrameViewer.java

```
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);
   }
}
```

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:

```
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:

```
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:

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

Drawing Rectangles

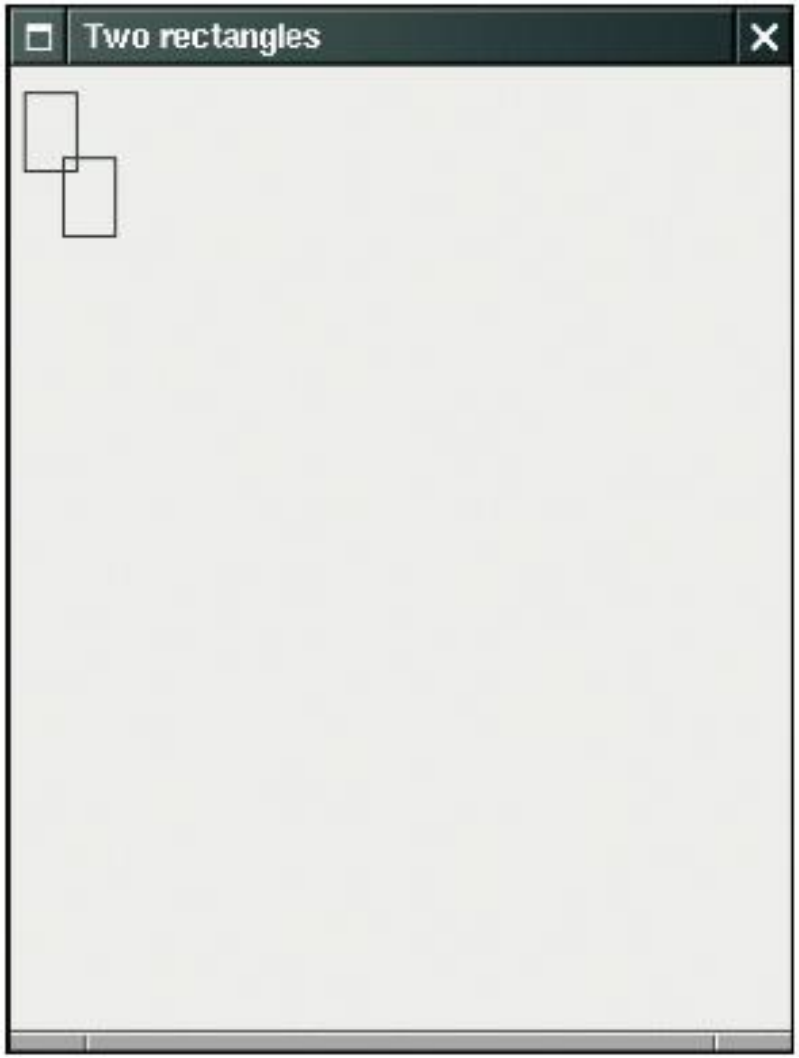


Figure 23
Drawing Rectangles

ch02/rectangles/RectangleComponent.java

```
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

ch02/rectangles/RectangleComponent.java (cont.)

```
        // Move rectangle 15 units to the right and 25 units down
        box.translate(15, 25);

        // Draw moved rectangle
        g2.draw(box);
    }
}
```

Using a Component

1. Construct a frame.
2. Construct an object of your component class:
```
RectangleComponent component = new RectangleComponent();
```
3. Add the component to the frame:
```
frame.add(component);
```
4. Make the frame visible.

ch02/rectangles/RectangleViewer.java

```
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 2.36

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:

```
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
```

ch02/applet/RectangleApplet.java

```
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);

```

Continued

ch02/applet/RectangleApplet.java (cont.)

```
        // Move rectangle 15 units to the right and 25 units down
        box.translate(15, 25);

        // Draw moved rectangle
        g2.draw(box);
    }
}
```

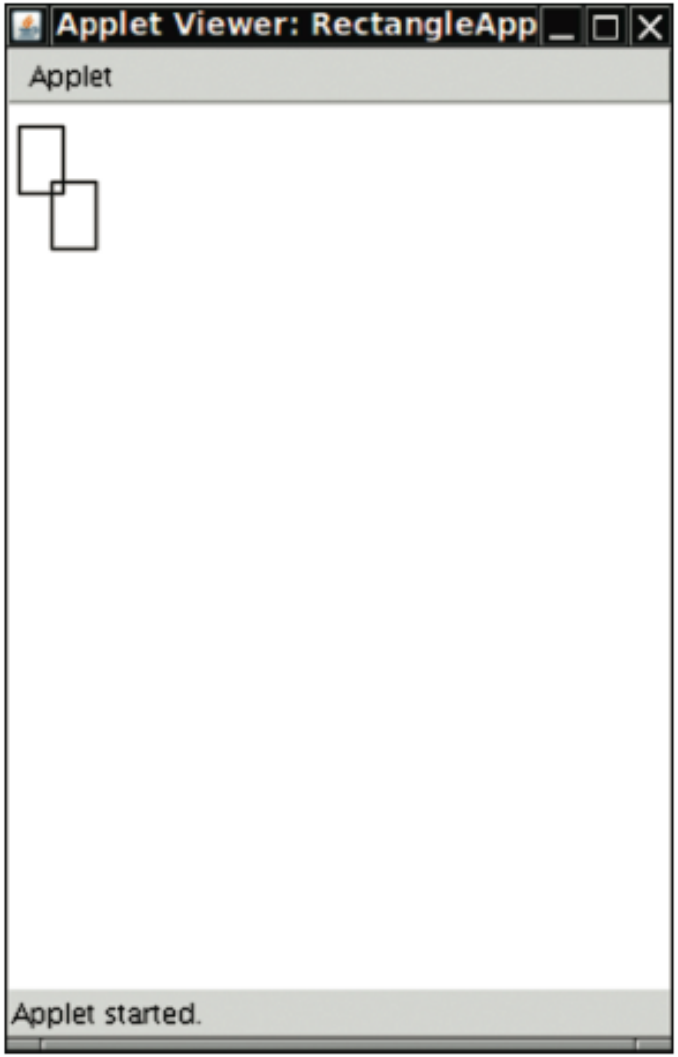
ch02/applet/RectangleApplet.html

```
<applet code="RectangleApplet.class" width="300" height="400">
</applet>
```

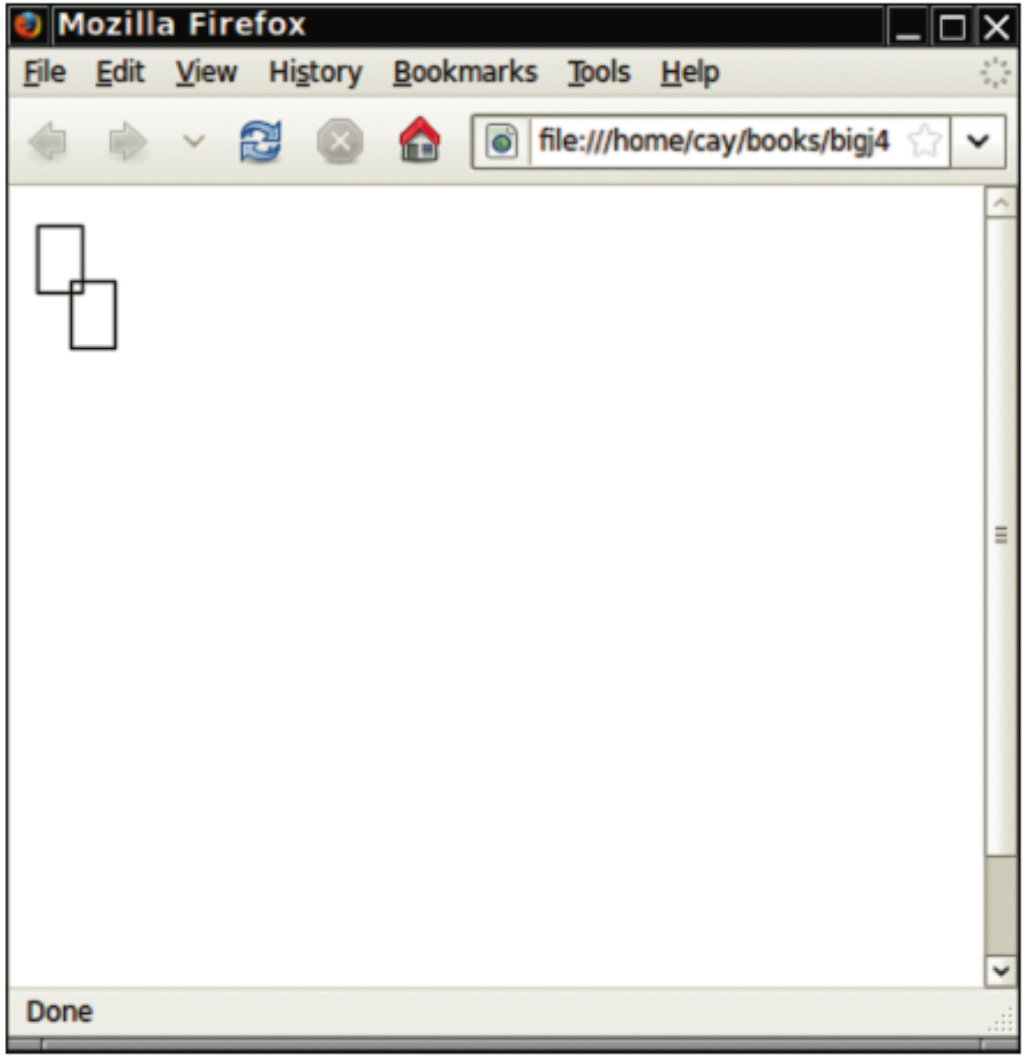
ch02/applet/RectangleAppletExplained.html

```
<html>
   <head>
      <title>Two rectangles</title>
   </head>
   <body>
      <p>Here is my <i>first applet</i>:</p>
      <applet code="RectangleApplet.class" width="300" height="400">
      </applet>
   </body>
</html>
```

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:

```
import java.awt.geom.Ellipse2D; // no .Double
```

- Must construct *and draw* the shape:

```
Ellipse2D.Double ellipse =
   new Ellipse2D.Double(x, y, width, height);
g2.draw(ellipse);
```

An Ellipse

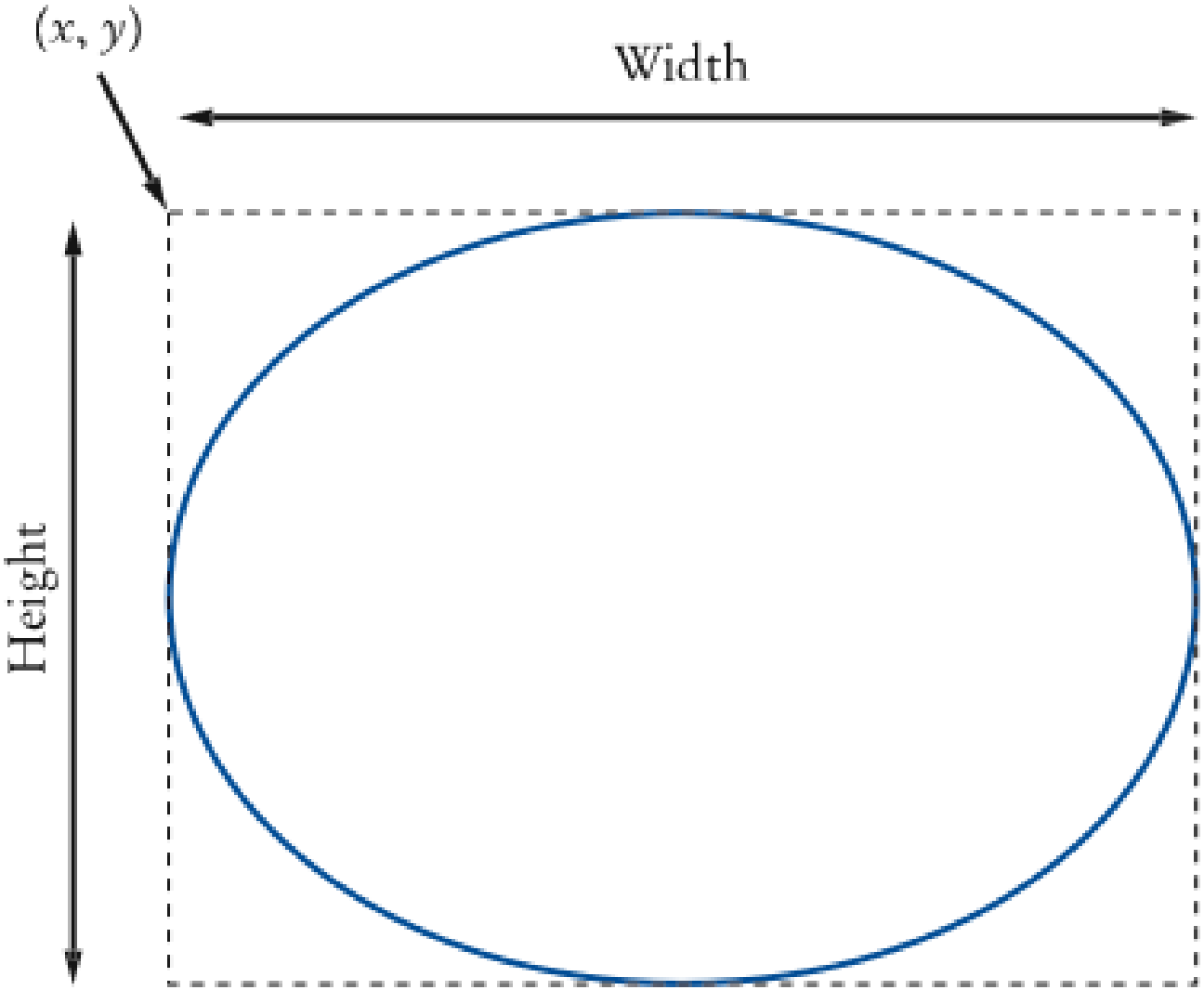


Figure 24 An Ellipse and Its Bounding Box

Drawing Lines

- To draw a line:

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

or,

```
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`:

```
Color magenta = new Color(255, 0, 255);
```

- Set color in graphics context:

```
g2.setColor(magenta);
```

- Color is used when drawing and filling shapes:

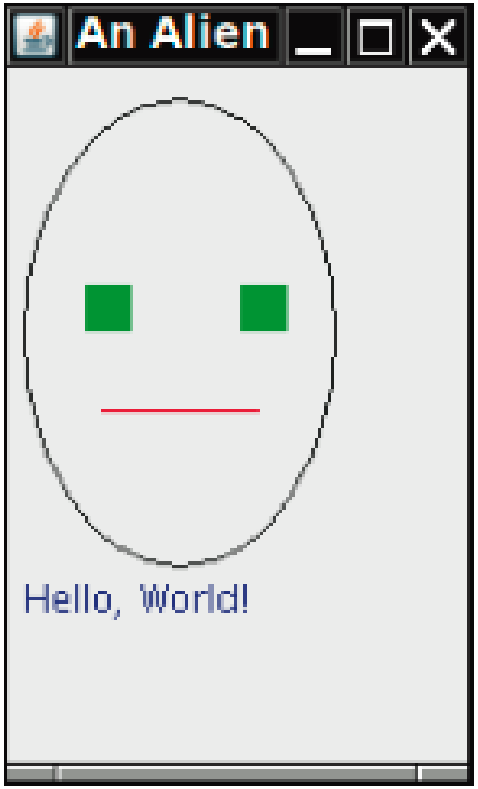
```
g2.fill(rectangle); // filled with current color
```

Predefined Colors and Their RGB Values

Color	RGB Value	
Color.BLACK	0, 0, 0	
Color.BLUE	0, 0, 255	
Color.CYAN	0, 255, 255	
Color.GRAY	128, 128, 128	
Color.DARKGRAY	64, 64, 64	
Color.LIGHTGRAY	192, 192, 192	
Color.GREEN	0, 255, 0	
Color.MAGENTA	255, 0, 255	
Color.ORANGE	255, 200, 0	
Color.PINK	255, 175, 175	
Color.RED	255, 0, 0	
Color.WHITE	255, 255, 255	
Color.YELLOW	255, 255, 0	

Alien Face

Figure 26
An Alien Face



ch02/face/FaceComponent.java

```
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

ch02/face/FaceComponent.java (cont.)

```
19          // Draw the head
20          Ellipse2D.Double head = new Ellipse2D.Double(5, 10, 100, 150);
21          g2.draw(head);
22
23          // Draw the eyes
24          g2.setColor(Color.GREEN);
25          Rectangle eye = new Rectangle(25, 70, 15, 15);
26          g2.fill(eye);
27          eye.translate(50, 0);
28          g2.fill(eye);
29
30          // Draw the mouth
31          Line2D.Double mouth = new Line2D.Double(30, 110, 80, 110);
32          g2.setColor(Color.RED);
33          g2.draw(mouth);
34
35          // Draw the greeting
36          g2.setColor(Color.BLUE);
37          g2.drawString("Hello, World!", 5, 175);
38      }
39  }
```

ch02/face/FaceViewer.java

```
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 2.37

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

Answer:

```
g2.draw(new Ellipse2D.Double(75, 75, 50, 50));
```

Self Check 2.39

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

Answer:

```
g2.drawString("V", 0, 30);
```

Self Check 2.40

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

Answer: `0, 0,` and `255`

Self Check 2.41

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:

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

2.16 Shape Classes

- Good practice: Make a class for each graphical shape

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

Drawing Cars

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

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

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

Continued

Drawing Cars (cont.)

Figure 7
The Car Component Draws Two Car Shapes

Plan Complex Shapes on Graph Paper

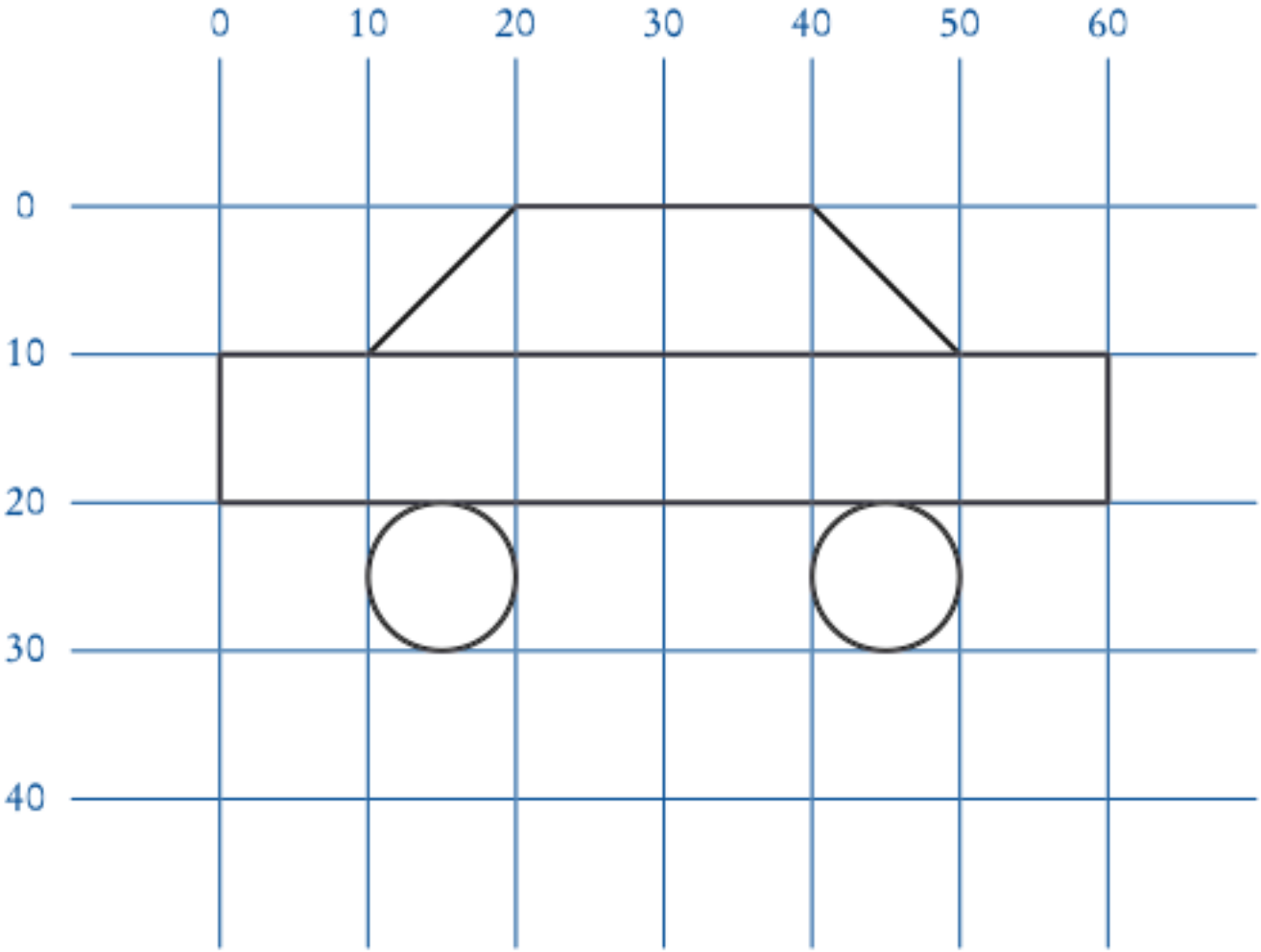


Figure 8 Using Graph Paper to Find Shape Coordinates

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`

ch02/car/Car.java

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

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

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

Continued

ch02/car/Car.java (cont.)

```
25
26      /**
27         Draws the car.
28         @param g2 the graphics context
29      */
30      public void draw(Graphics2D g2)
31      {  Rectangle body
33            = new Rectangle(xLeft, yTop + 10, 60, 10);
34         Ellipse2D.Double frontTire
35            = new Ellipse2D.Double(xLeft + 10, yTop + 20, 10, 10);
36         Ellipse2D.Double rearTire
37            = new Ellipse2D.Double(xLeft + 40, yTop + 20, 10, 10);
38
39         // The bottom of the front windshield
40         Point2D.Double r1
41            = new Point2D.Double(xLeft + 10, yTop + 10);
42         // The front of the roof
43         Point2D.Double r2
44            = new Point2D.Double(xLeft + 20, yTop);
45         // The rear of the roof
46         Point2D.Double r3
47            = new Point2D.Double(xLeft + 40, yTop);
```

Continued

ch02/car/Car.java (cont.)

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

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

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

ch02/car/CarComponent.java

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

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

      Car car1 = new Car(0, 0);

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

      Car car2 = new Car(x, y);

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

ch02/car/CarViewer.java

```
import javax.swing.JFrame;

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

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

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

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

Self Check 2.42

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

Answer: `CarComponent`

Self Check 2.43

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

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

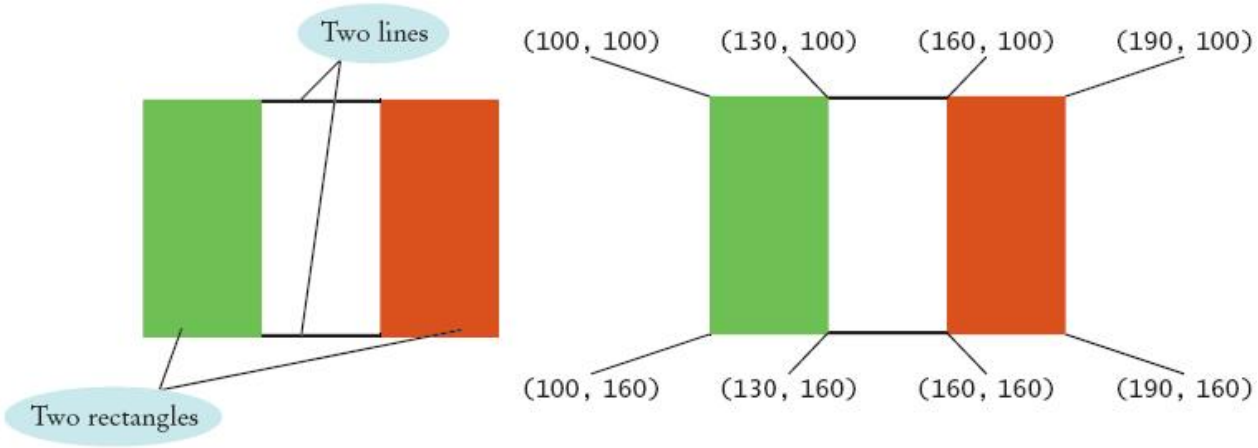
```
g2.fill(frontTire);
g2.fill(rearTire);
```

Self Check 2.44

How do you make the cars twice as big?

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

Drawing Graphical Shapes




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

Exercises for Chapter 2 (2 & 3 in PDF)

- Read chapter 2 (2 & 3 in PDF)
- Answer the review questions

Exercises for Chapter 2 (2 & 3 in PDF)

- Programming
  - P2.25 (P3.16 in PDF)

Write a program that draws a picture of a house. It could be as simple as the accompanying figure, or if you like, make it more elaborate (3-D, skyscraper, marble columns in the entryway, whatever).

Implement a class `House` and supply a method `draw(Graphics2D g2)` that draws the house.

Exercises for Chapter 2

- **CodingBat** Programming Exercises:
  - Create an account on http://codingbat.com with your BFH email address.
  - Go to ***prefs*** and fill in your full name in ***Name (first,last)***
  - Add my email address in the field ***Share To***: marcus.hudritsch@bfh.ch and leave it for the rest of the course.
  - Do the following simple string exercises (no if statement or loops are needed, only string methods):
    - **String1**: helloName, makeAbba, makeTags, makeOutWord, extraEnd, firstHalf, withoutEnd, nonStart, left2, right2, middleTwo, nTwice, middleThree,