ICOM 4015: Advanced Programming

Lecture 6

Chapter Six: Iteration

ICOM 4015 Fall 2008

Chapter Six: Iteration

Chapter Goals

- To be able to program loops with the `while`, `for`, and `do` statements
- To avoid infinite loops and off-by-one errors
- To understand nested loops
- To learn how to process input
- To implement simulations
- To learn about the debugger

`while` Loops

- Executes a block of code repeatedly
- A condition controls how often the loop is executed

```
while (condition)
   statement
```

- Most commonly, the statement is a block statement (set of statements delimited by `{ }`)

Calculating the Growth of an Investment

- Invest $10,000, 5% interest, compounded annually

Year	**Balance**
0	$10,000
1	$10,500
2	$11,025
3	$11,576.25
4	$12,155.06
5	$12,762.82

Calculating the Growth of an Investment

- When has the bank account reached a particular balance?

```
while (balance < targetBalance)
{
   years++;
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

ch06/invest1/Investment.java

```java
/**
   A class to monitor the growth of an investment that
   accumulates interest at a fixed annual rate.
*/
public class Investment
{
   /**
      Constructs an Investment object from a starting balance and
      interest rate.
      @param aBalance the starting balance
      @param aRate the interest rate in percent
   */
   public Investment(double aBalance, double aRate)
   {
      balance = aBalance;
      rate = aRate;
      years = 0;
   }

   /**
      Keeps accumulating interest until a target balance has
      been reached.
      @param targetBalance the desired balance
   */
```

ch06/invest1/Investment.java (cont.)

```
   public void waitForBalance(double targetBalance)
   {
      while (balance < targetBalance)
      {
         years++;
         double interest = balance * rate / 100;
         balance = balance + interest;
      }
   }

   /**
      Gets the current investment balance.
      @return the current balance
   */
   public double getBalance()
   {
      return balance;
   }

   /**
      Gets the number of years this investment has accumulated
      interest.
```

ch06/invest1/Investment.java (cont.)

```java
         @return the number of years since the start of the investment
      */
      public int getYears()
      {
         return years;
      }

      private double balance;
      private double rate;
      private int years;
   }
```

ch06/invest1/InvestmentRunner.java

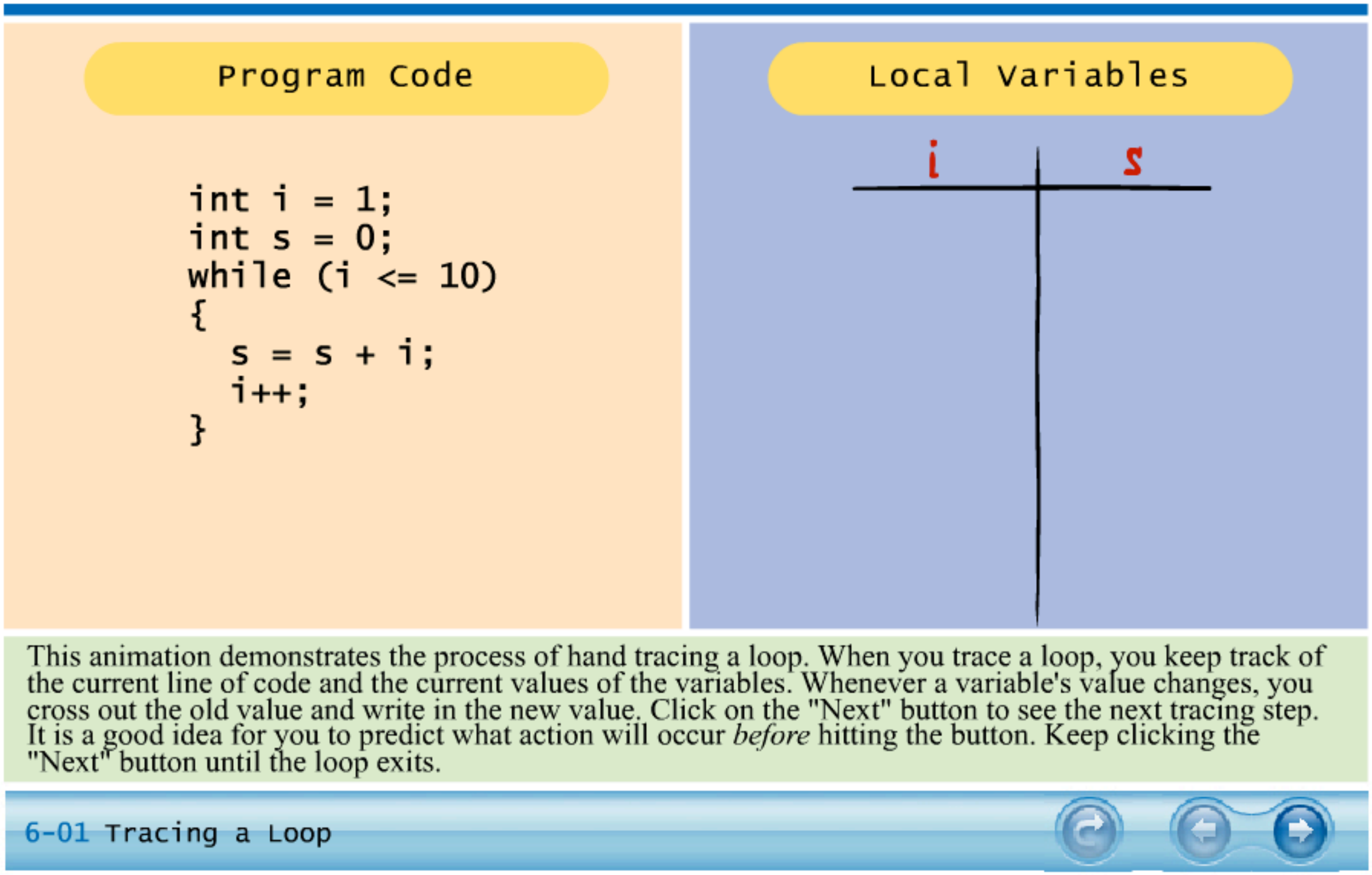
```java
/**
   This program computes how long it takes for an investment
   to double.
*/
public class InvestmentRunner
{
   public static void main(String[] args)
   {
      final double INITIAL_BALANCE = 10000;
      final double RATE = 5;
      Investment invest = new Investment(INITIAL_BALANCE, RATE);
      invest.waitForBalance(2 * INITIAL_BALANCE);
      int years = invest.getYears();
      System.out.println("The investment doubled after "
            + years + " years");
   }
}
```

ch06/invest1/InvestmentRunner.java (cont.)

Output:

```
The investment doubled after 15 years
```

Animation 6.1 –



```
int i = 1;
int s = 0;
while (i <= 10)
{
   s = s + i;
   i++;
}
```

This animation demonstrates the process of hand tracing a loop. When you trace a loop, you keep track of the current line of code and the current values of the variables. Whenever a variable's value changes, you cross out the old value and write in the new value. Click on the "Next" button to see the next tracing step. It is a good idea for you to predict what action will occur *before* hitting the button. Keep clicking the "Next" button until the loop exits.

`while` Loop Flowchart

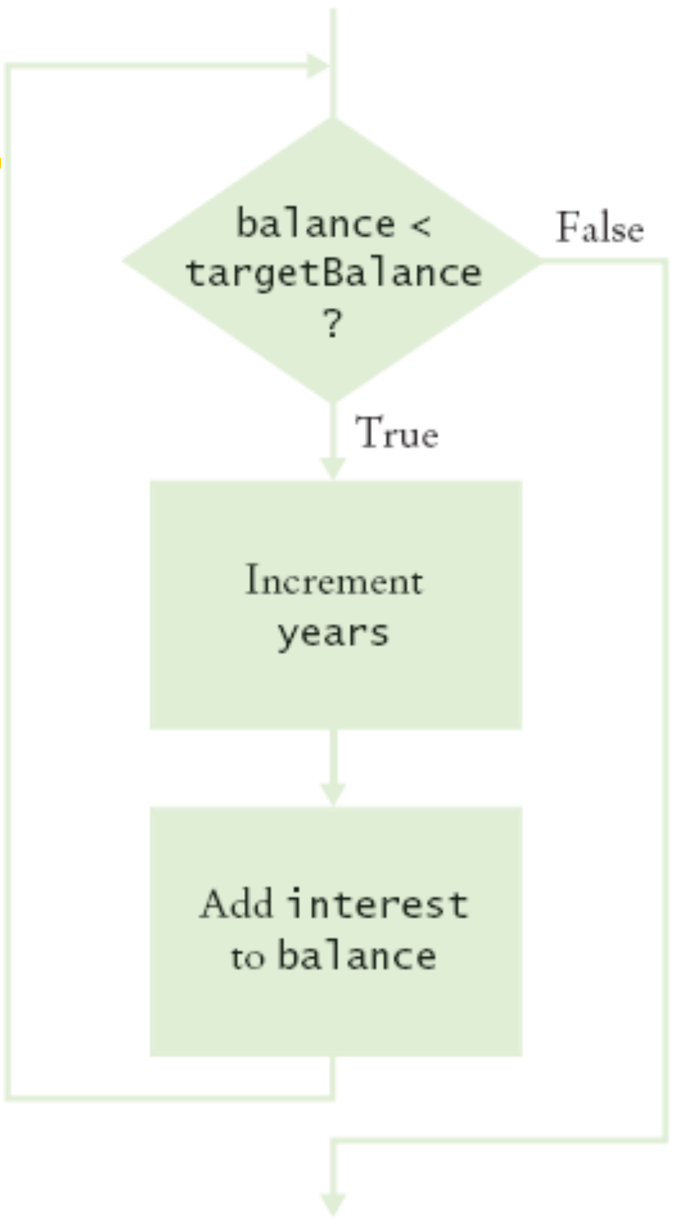


Figure 1 Flowchart of a `while` Loop

Syntax 6.1 The `while` Statement

```
while (condition)
   statement
```

Example:

```
while (balance < targetBalance)
{
  years++;
  double interest = balance * rate / 100;
  balance = balance + interest;
}
```

Purpose:

To repeatedly execute a statement as long as a condition is true.

Self Check 6.1

How often is the statement in the loop

```
while (false) statement;
```

executed?

Answer: Never.

Self Check 6.2

What would happen if `RATE` was set to `0` in the `main` method of the `InvestmentRunner` program?

Answer: The `waitForBalance` method would never return due to an infinite loop.

Common Error: Infinite Loops

-
  ```
  int years = 0;
  while (years < 20)
  {
     double interest = balance * rate / 100;
     balance = balance + interest;
  }
  ```

-
  ```
  int years = 20;
  while (years > 0)
  {
     years++; // Oops, should have been years–
     double interest = balance * rate / 100;
     balance = balance + interest;
  }
  ```

- Loops run forever – must kill program

Common Error: Off-by-One Errors

-
  ```
  int years = 0;
  while (balance < 2 * initialBalance)
  {
     years++;
     double interest = balance * rate / 100;
     balance = balance + interest;
  }
  System.out.println("The investment reached the target
     after " + years + " years.");
  ```

Should `years` start at 0 or 1?

Should the test be < or <=?

Avoiding Off-by-One Error

- Look at a scenario with simple values:
  initial `balance: $100`
  interest `rate: 50%`
  after year 1, the `balance` is $150
  after year 2 it is $225, or over $200
  so the investment doubled after 2 years
  the loop executed two times, incrementing `years` each time
  Therefore: `years` must start at `0`, not at `1`.
- interest rate: 100%
  after one year: `balance` is 2 * `initialBalance`
  loop should stop
  Therefore: must use <
- Think, don't compile and try at random

`do` Loops

* Executes loop body at least once:

```
do
   statement
while (condition);
```

* Example: Validate input

```
double value;
do
{
   System.out.print("Please enter a positive number: ");
   value = in.nextDouble();
}
 while (value <= 0);
```

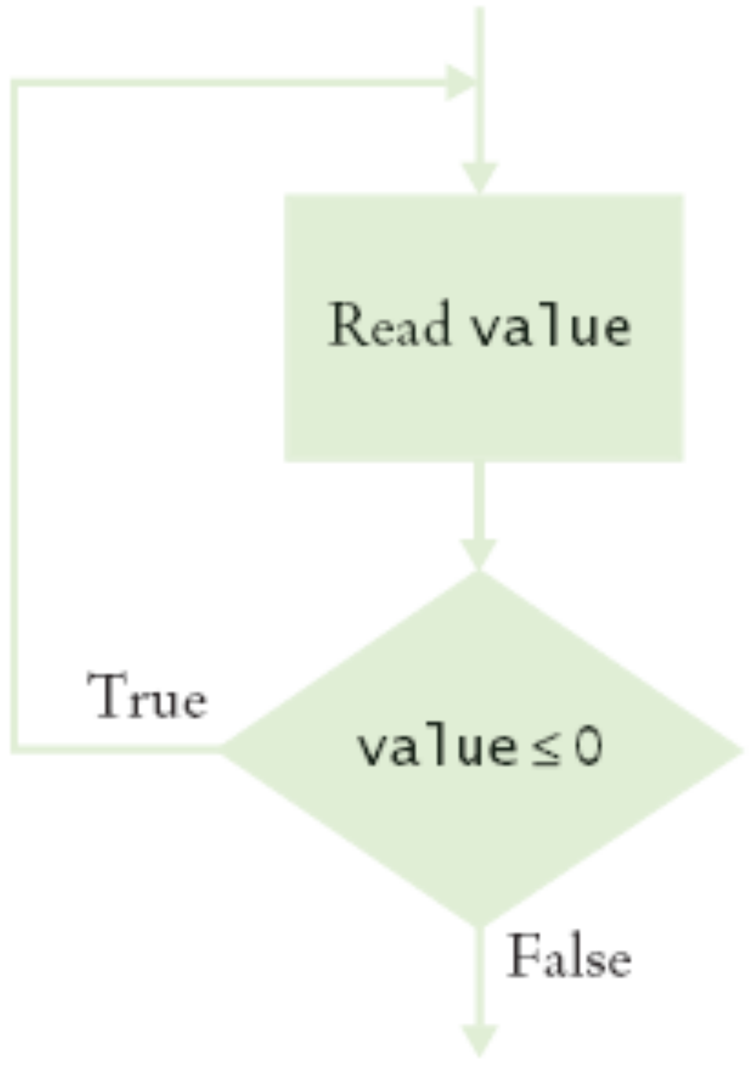
Continued

`do` Loops (cont.)

- Alternative:

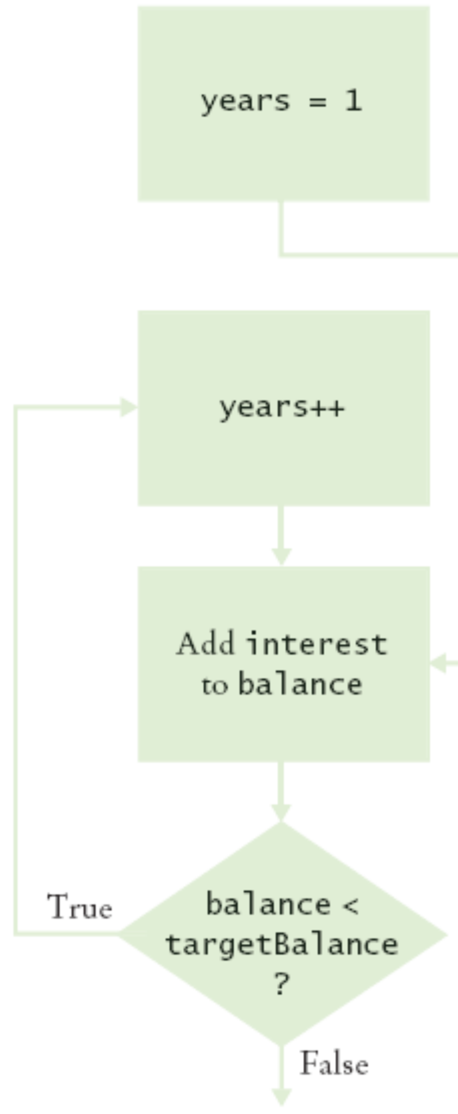
```
boolean done = false;
while (!done)
{
   System.out.print("Please enter a positive number: ");
   value = in.nextDouble();
   if (value > 0) done = true;
}
```

`do` Loop Flowchart



Flowchart of a do Loop

Spaghetti Code



Spaghetti Code

`for` Loops

- `for (`*`initialization`*`; `*`condition`*`; `*`update`*`)`
  ```
      statement
  ```
- Example:
  ```
  for (int i = 1; i <= n; i++)
  {
     double interest = balance * rate / 100;
     balance = balance + interest;
  }
  ```
- Equivalent to
  ```
  initialization;
  while (condition)
  { statement;
    update; }
  ```

Continued

`for` Loops (cont.)

- Other examples:

```
for (years = n; years > 0; years--) . . .
for (x = -10; x <= 10; x = x + 0.5) . . .
```

`for` Loop Flowchart

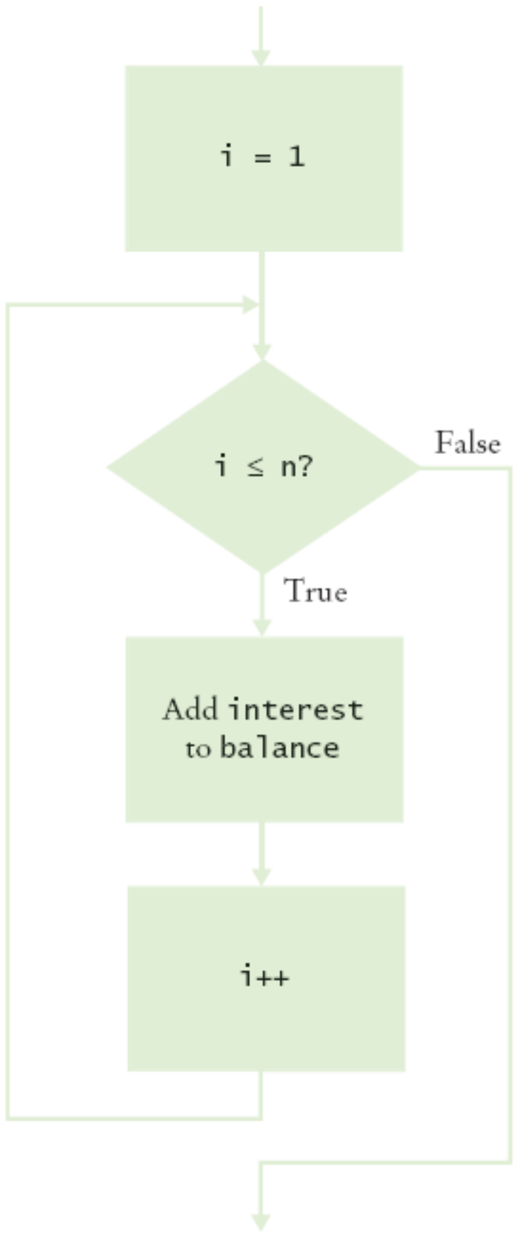
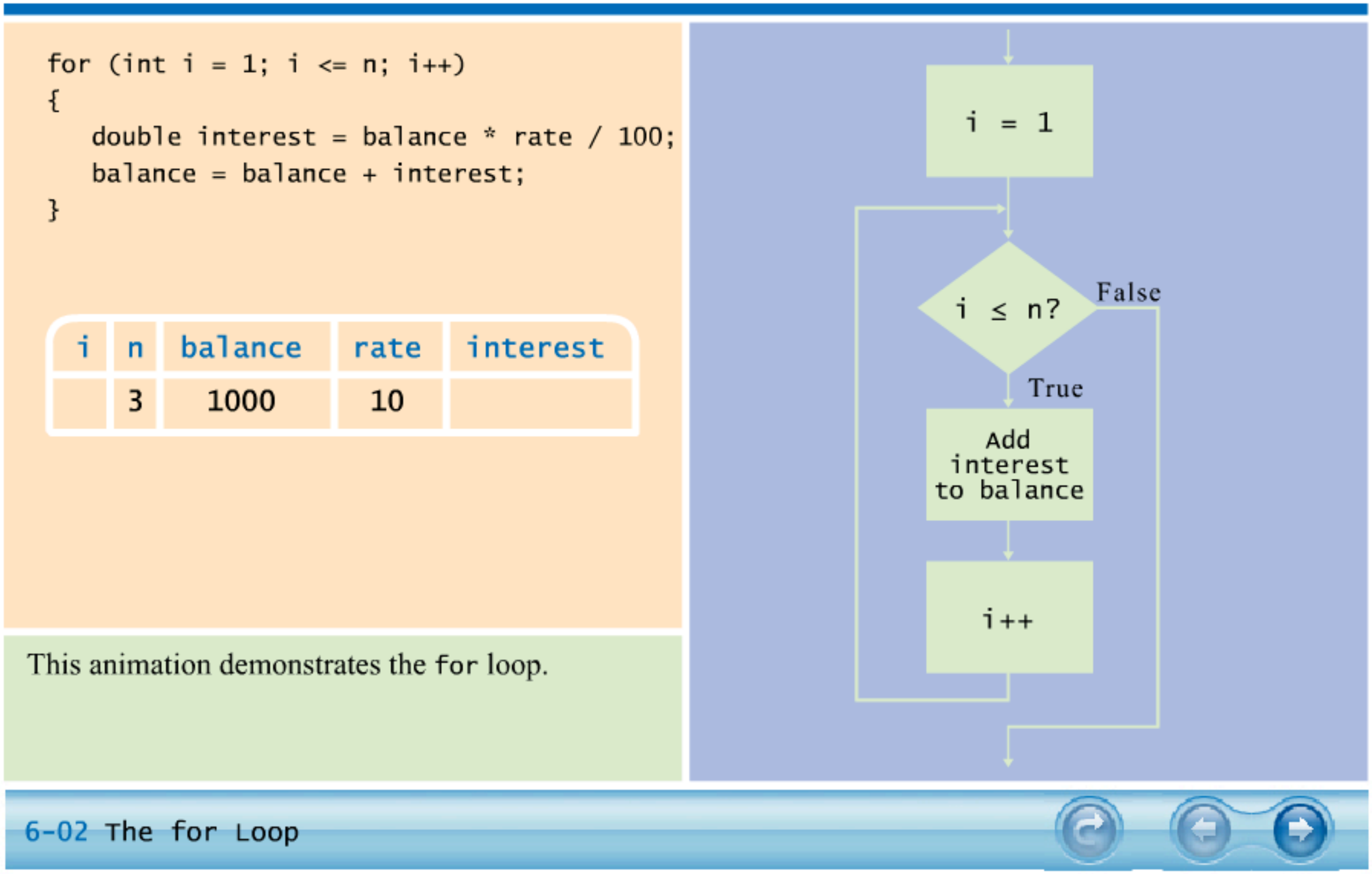


Figure 2 Flowchart of a for Loop

Animation 6.2 –



```
for (int i = 1; i <= n; i++)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

i	n	balance	rate	interest
	3	1000	10	

This animation demonstrates the for loop.

6-02 The for Loop

Syntax 6.2 The `for` Statement

```
for (initialization; condition; update)
   statement
```

Example:

```
for (int i = 1; i <= n; i++)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

Purpose:

To execute an initialization, then keep executing a statement and updating an expression while a condition is true.

ch06/invest2/Investment.java

```java
/**
   A class to monitor the growth of an investment that
   accumulates interest at a fixed annual rate
*/
public class Investment
{
   /**
      Constructs an Investment object from a starting balance and
      interest rate.
      @param aBalance the starting balance
      @param aRate the interest rate in percent
   */
   public Investment(double aBalance, double aRate)
   {
      balance = aBalance;
      rate = aRate;
      years = 0;
   }

   /**
      Keeps accumulating interest until a target balance has
      been reached.
```

ch06/invest2/Investment.java (cont.)

```
         @param targetBalance the desired balance
      */
      {
         while (balance < targetBalance)
         {
            years++;
            double interest = balance * rate / 100;
            balance = balance + interest;
         }
      }

      /**
         Keeps accumulating interest for a given number of years.
         @param n the number of years
      */
      public void waitYears(int n)
      {
         for (int i = 1; i <= n; i++)
         {
            double interest = balance * rate / 100;
            balance = balance + interest;
```

ch06/invest2/Investment.java (cont.)

```
         }
         years = years + n;
      }

      /**
         Gets the current investment balance.
         @return the current balance
      */
      public double getBalance()
      {
         return balance;
      }

      /**
         Gets the number of years this investment has accumulated
         interest.
         @return the number of years since the start of the investment
      */
      public int getYears()
      {
         return years;
      }
```

ch06/invest2/Investment.java (cont.)

```

   private double balance;
   private double rate;
   private int years;
}
```

ch06/invest2/InvestmentRunner.java

```java
/**
   This program computes how much an investment grows in
   a given number of years.
*/
public class InvestmentRunner
{
   public static void main(String[] args)
   {
      final double INITIAL_BALANCE = 10000;
      final double RATE = 5;
      final int YEARS = 20;
      Investment invest = new Investment(INITIAL_BALANCE, RATE);
      invest.waitYears(YEARS);
      double balance = invest.getBalance();
      System.out.printf("The balance after %d years is %.2f\n",
            YEARS, balance);
   }
}
```

Output:

```
The balance after 20 years is 26532.98
```

Self Check 6.3

Rewrite the for loop in the `waitYears` method as a `while` loop.

Answer: `int i = 1; while (i <= n) { double interest = balance * rate / 100; balance = balance + interest; i++; }`

Self Check 6.4

Rewrite the `for` loop in the `waitYears` method as a `while` How many times does the following for loop execute?

```
for (i = 0; i <= 10; i++)
   System.out.println(i * i);
```

Answer: 11 times.

Common Errors: Semicolons

- A missing semicolon

```
for (years = 1;
    (balance = balance + balance * rate / 100) <
       targetBalance;
    years++)
    System.out.println(years);
```

- A semicolon that shouldn't be there

```
sum = 0;
for (i = 1; i <= 10; i++);
    sum = sum + i;
System.out.println(sum);
```

Nested Loops

- Create triangle pattern

```
[]
[][]
[][][]
[][][][]
```

- Loop through rows

```
for (int i = 1; i <= n; i++)
{
   // make triangle row
}
```

- *Make triangle row* is another loop

```
for (int j = 1; j <= i; j++)
   r = r + "[]";
r = r + "\n";
```

- Put loops together → Nested loops

ch06/triangle1/Triangle.java

```java
/**
   This class describes triangle objects that can be displayed
   as shapes like this:
   []
   [][]
   [][][]
*/
public class Triangle
{
   /**
      Constructs a triangle.
      @param aWidth the number of [] in the last row of the triangle.
   */
   public Triangle(int aWidth)
   {
      width = aWidth;
   }

   /**
      Computes a string representing the triangle.
      @return a string consisting of [] and newline characters
   */
```

ch06/triangle1/Triangle.java (cont.)

```
   public String toString()
   {
      String r = "";
      for (int i = 1; i <= width; i++)
      {
         // Make triangle row
         for (int j = 1; j <= i; j++)
            r = r + "[]";
         r = r + "\n";
      }
      return r;
   }

   private int width;
}
```

File TriangleRunner.java

```java
/**
   This program prints two triangles.
*/
public class TriangleRunner
{
   public static void main(String[] args)
   {
      Triangle small = new Triangle(3);
      System.out.println(small.toString());

      Triangle large = new Triangle(15);
      System.out.println(large.toString());
   }
}
```

File TriangleRunner.java (cont.)

Output:

```
[]
[] []
[] [] []

[]
[] []
[] [] []
[] [] [] []
[] [] [] [] []
[] [] [] [] [] []
[] [] [] [] [] [] []
[] [] [] [] [] [] [] []  [] [] [] [] [] [] [] []
[]  [] [] [] [] [] [] [] [] [] []  [] [] [] []
[] [] [] [] [] [] []  [] [] [] [] [] [] [] [] []
[] [] []  [] [] [] [] [] [] [] [] [] [] [] [] []
```

Continued

File TriangleRunner.java (cont.)

Output (continued):

```
[][][][][][][][][][][][][][]  [][]
[][][][][][][][][][][][][]
```

Self Check 6.5

How would you modify the nested loops so that you print a square instead of a triangle?

Answer: Change the inner loop to `for (int j = 1; j <= width; j++)`

Self Check 6.6

How would you modify the nested loops so that you print a square instead of a What is the value of n after the following nested loops?

```
int n = 0;
for (int i = 1; i <= 5; i++)
   for (int j = 0; j < i; j++)
      n = n + j;
```

Answer: 20.

Processing Sentinel Values

- Sentinel value: Can be used for indicating the end of a data set
- `0` or `-1` make poor sentinels; better use `Q`

```
System.out.print("Enter value, Q to quit: ");
String input = in.next();
if (input.equalsIgnoreCase("Q"))
   We are done
else
{
   double x = Double.parseDouble(input);
   . . .
}
```

Loop and a half

- Sometimes termination condition of a loop can only be evaluated in the middle of the loop
- Then, introduce a boolean variable to control the loop:

```
boolean done = false;
while (!done)
{
   Print prompt
   String input = read input;
   if (end of input indicated)
      done = true;
   else
   {
      Process input
   }
}
```

ch06/dataset/DataAnalyzer.java

```java
import java.util.Scanner;

/**
   This program computes the average and maximum of a set
   of input values.
*/
public class DataAnalyzer
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      DataSet data = new DataSet();

      boolean done = false;
      while (!done)
      {
         System.out.print("Enter value, Q to quit: ");
         String input = in.next();
         if (input.equalsIgnoreCase("Q"))
            done = true;
```

Continued

ch06/dataset/DataAnalyzer.java (cont.)

```
            else
            {
               double x = Double.parseDouble(input);
               data.add(x);
            }
         }

         System.out.println("Average = " + data.getAverage());
         System.out.println("Maximum = " + data.getMaximum());
      }
   }
```

ch06/dataset/DataSet.java

```java
/**
   Computes the average of a set of data values.
*/
public class DataSet
{
   /**
      Constructs an empty data set.
   */
   public DataSet()
   {
      sum = 0;
      count = 0;
      maximum = 0;
   }

   /**
      Adds a data value to the data set
      @param x a data value
   */
   public void add(double x)
   {
```

Continued

ch06/dataset/DataSet.java (cont.)

```java
         sum = sum + x;
         if (count == 0 || maximum < x) maximum = x;
         count++;
      }

      /**
         Gets the average of the added data.
         @return the average or 0 if no data has been added
      */
      public double getAverage()
      {
         if (count == 0) return 0;
         else return sum / count;
      }

      /**
         Gets the largest of the added data.
         @return the maximum or 0 if no data has been added
      */
```

Continued

ch06/dataset/DataSet.java (cont.)

```java
    public double getMaximum()
    {
        return maximum;
    }

    private double sum;
    private double maximum;
    private int count;
}
```

Output:

```
Enter value, Q to quit: 10
Enter value, Q to quit: 0
Enter value, Q to quit: -1
Enter value, Q to quit: Q
Average = 3.0
Maximum = 10.0
```

Self Check 6.7

Why does the `DataAnalyzer` class call `in.next` and not `in.nextDouble`?

Answer: Because we don't know whether the next input is a number or the letter `Q`.

Self Check 6.8

Would the `DataSet` class still compute the correct maximum if you simplified the update of the maximum field in the `add` method to the following statement?

```
if (maximum < x) maximum = x;
```

Answer: No. If *all* input values are negative, the maximum is also negative. However, the `maximum` field is initialized with 0. With this simplification, the maximum would be falsely computed as 0.

Random Numbers and Simulations

- In a simulation, you repeatedly generate random numbers and use them to simulate an activity
- Random number generator

```
Random generator = new Random(); int n =
generator.nextInt(a); // 0 < = n < a double x =
generator.nextDouble(); // 0 <= x < 1
```

- Throw die (random number between 1 and 6)

```
int d = 1 + generator.nextInt(6);
```

ch06/random1/Die.java

```java
import java.util.Random;

/**
   This class models a die that, when cast, lands on a random
   face.
*/
public class Die
{
   /**
      Constructs a die with a given number of sides.
      @param s the number of sides, e.g. 6 for a normal die
   */
   public Die(int s)
   {
      sides = s;
      generator = new Random();
   }

   /**
      Simulates a throw of the die
      @return the face of the die
   */
```

Continued

ch06/random1/Die.java (cont.)

```java
   public int cast()
   {
      return 1 + generator.nextInt(sides);
   }

   private Random generator;
   private int sides;
}
```

ch06/random1/DieSimulator.java

```java
/**
   This program simulates casting a die ten times.
*/
public class DieSimulator
{
   public static void main(String[] args)
   {
      Die d = new Die(6);
      final int TRIES = 10;
      for (int i = 1; i <= TRIES; i++)
      {
         int n = d.cast();
         System.out.print(n + " ");
      }
      System.out.println();
   }
}
```

ch06/random1/DieSimulator.java (cont.)

Output:

```
6 5 6 3 2 6 3 4 4 1
```

Second Run:

```
3 2 2 1 6 5 3 4 1 2
```

Buffon Needle Experiment

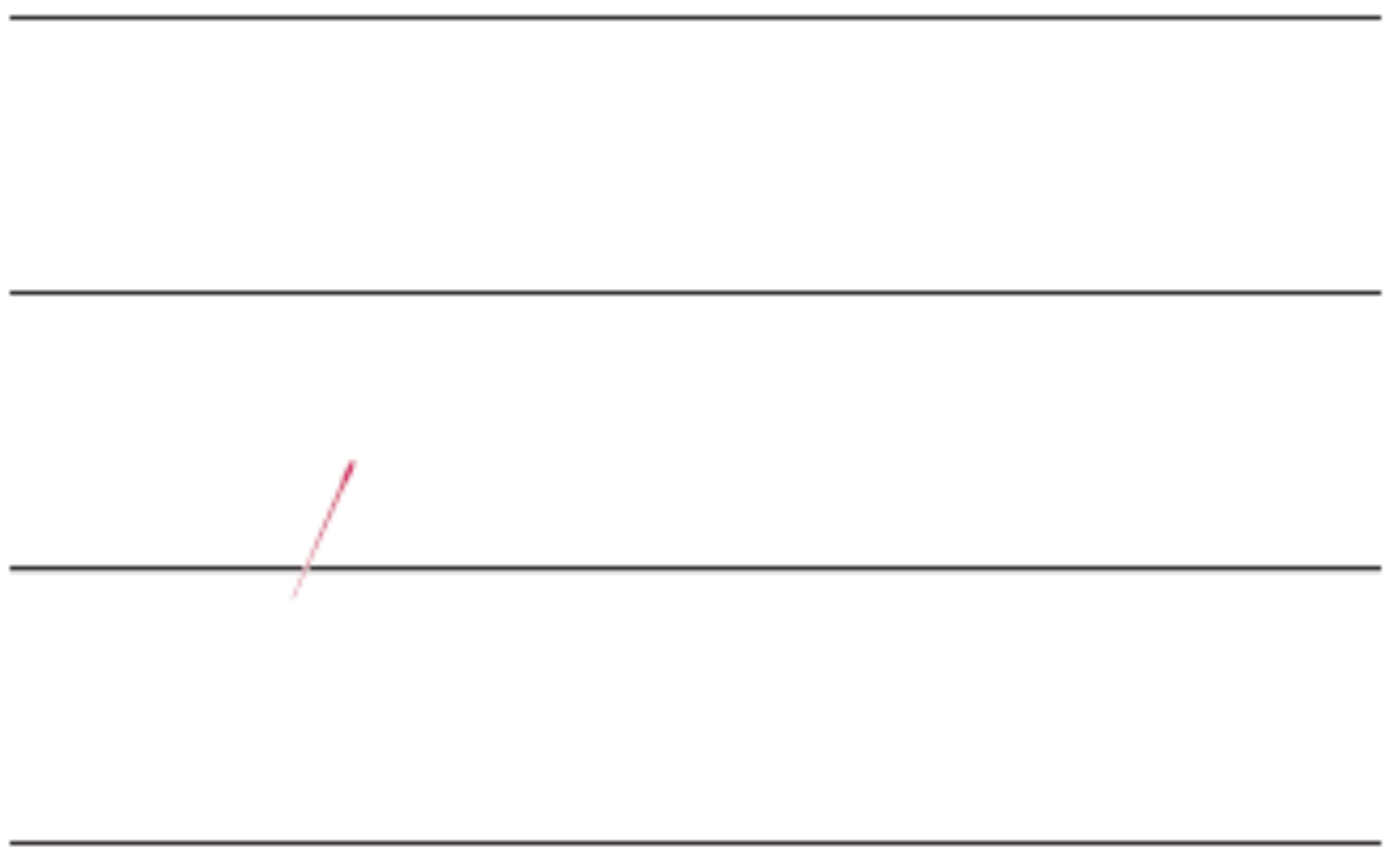


Figure 3 The Buffon Needle Experiment

Needle Position

Buffon Needle Experiment

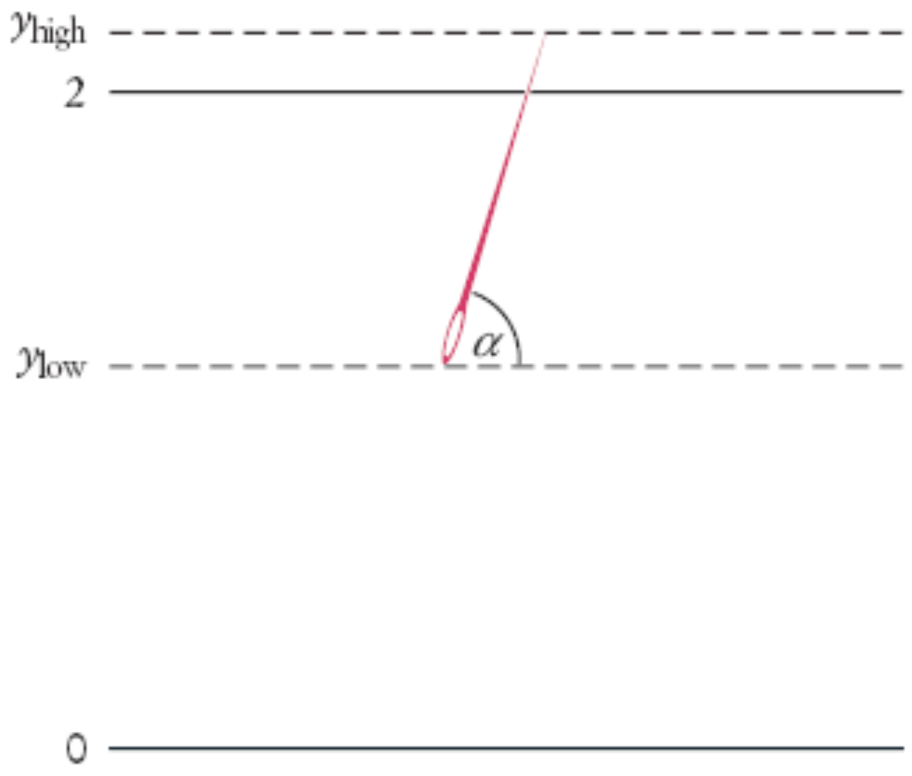


Figure 4
When Does the Needle Fall on a Line?

Needle Position

- Needle length = 1, distance between lines = 2
- Generate random *ylow* between 0 and 2
- Generate random angle α between 0 and 180 degrees
- *yhigh* = *ylow* + sin(α)
- Hit if *yhigh* ≥ 2

ch06/random2/Needle.java

```java
import java.util.Random;

/**
   This class simulates a needle in the Buffon needle experiment.
*/
public class Needle
{
   /**
      Constructs a needle.
   */
   public Needle()
   {
      hits = 0;
      tries = 0;
      generator = new Random();
   }

   /**
      Drops the needle on the grid of lines and
      remembers whether the needle hit a line.
   */
```

Continued

ch06/random2/Needle.java (cont.)

```java
   public void drop()
   {
      double ylow = 2 * generator.nextDouble();
      double angle = 180 * generator.nextDouble();

      // Computes high point of needle

      double yhigh = ylow + Math.sin(Math.toRadians(angle));
      if (yhigh >= 2) hits++;
      tries++;
   }

   /**
      Gets the number of times the needle hit a line.
      @return the hit count
   */
   public int getHits()
   {
      return hits;
   }

```

Continued

ch06/random2/Needle.java (cont.)

```java
   /**
      Gets the total number of times the needle was dropped.
      @return the try count
   */
   public int getTries()
   {
      return tries;
   }

   private Random generator;
   private int hits;
   private int tries;
}
```

Output:

```
Tries = 10000, Tries / Hits = 3.08928
Tries = 1000000, Tries / Hits = 3.14204
```

ch06/random2/NeedleSimulator.java

```java
/**
   This program simulates the Buffon needle experiment
   and prints the resulting approximations of pi.
*/
public class NeedleSimulator
{
   public static void main(String[] args)
   {
      Needle n = new Needle();
      final int TRIES1 = 10000;
      final int TRIES2 = 1000000;

      for (int i = 1; i <= TRIES1; i++)
         n.drop();
      System.out.printf("Tries = %d, Tries / Hits = %8.5f\n",
            TRIES1, (double) n.getTries() / n.getHits());

      for (int i = TRIES1 + 1; i <= TRIES2; i++)
         n.drop();
      System.out.printf("Tries = %d, Tries / Hits = %8.5f\n",
            TRIES2, (double) n.getTries() / n.getHits());
   }
}
```

ch06/random2/NeedleSimulator.java

Output:

```
Tries = 10000, Tries / Hits = 3.08928 Tries = 1000000,
Tries / Hits = 3.14204
```

Self Check 6.9

How do you use a random number generator to simulate the toss of a coin?

Answer: `int n = generator.nextInt(2); // 0 = heads, 1 = tails`

Self Check 6.10

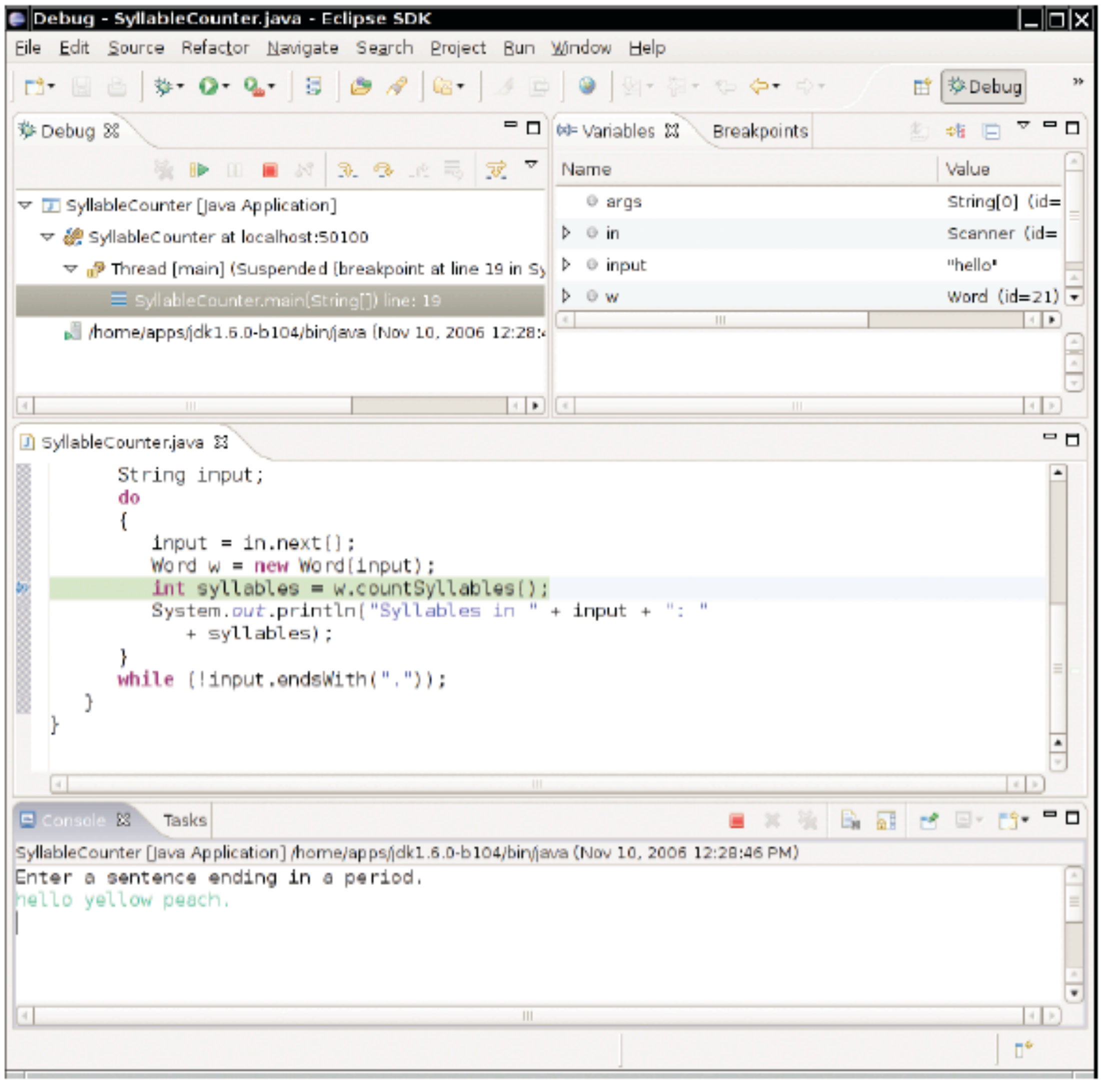
Why is the `NeedleSimulator` program not an efficient method for computing π?

Answer: The program repeatedly calls `Math.toRadians(angle)`. You could simply call `Math.toRadians(180)` to compute π.

Using a Debugger

- Debugger = program to run your program and analyze its run-time behavior
- A debugger lets you stop and restart your program, see contents of variables, and step through it
- The larger your programs, the harder to debug them simply by inserting print commands
- Debuggers can be part of your IDE (e.g. Eclipse, BlueJ) or separate programs (e.g. JSwat)
- Three key concepts:
  - *Breakpoints*
  - *Single-stepping*
  - *Inspecting variables*

The Debugger Stopping at a Breakpoint


Debug - SyllableCounter.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Debug
Variables
Breakpoints
SyllableCounter.java
Console
Tasks

```
SyllableCounter [Java Application]
  SyllableCounter at localhost:50100
    Thread [main] (Suspended (breakpoint at line 19 in Sy
      SyllableCounter.main(String[]) line: 19
  /home/apps/jdk1.6.0-b104/bin/java (Nov 10, 2006 12:28:
```

Name	Value
args	String[0] (id=
in	Scanner (id=
input	"hello"
w	Word (id=21)

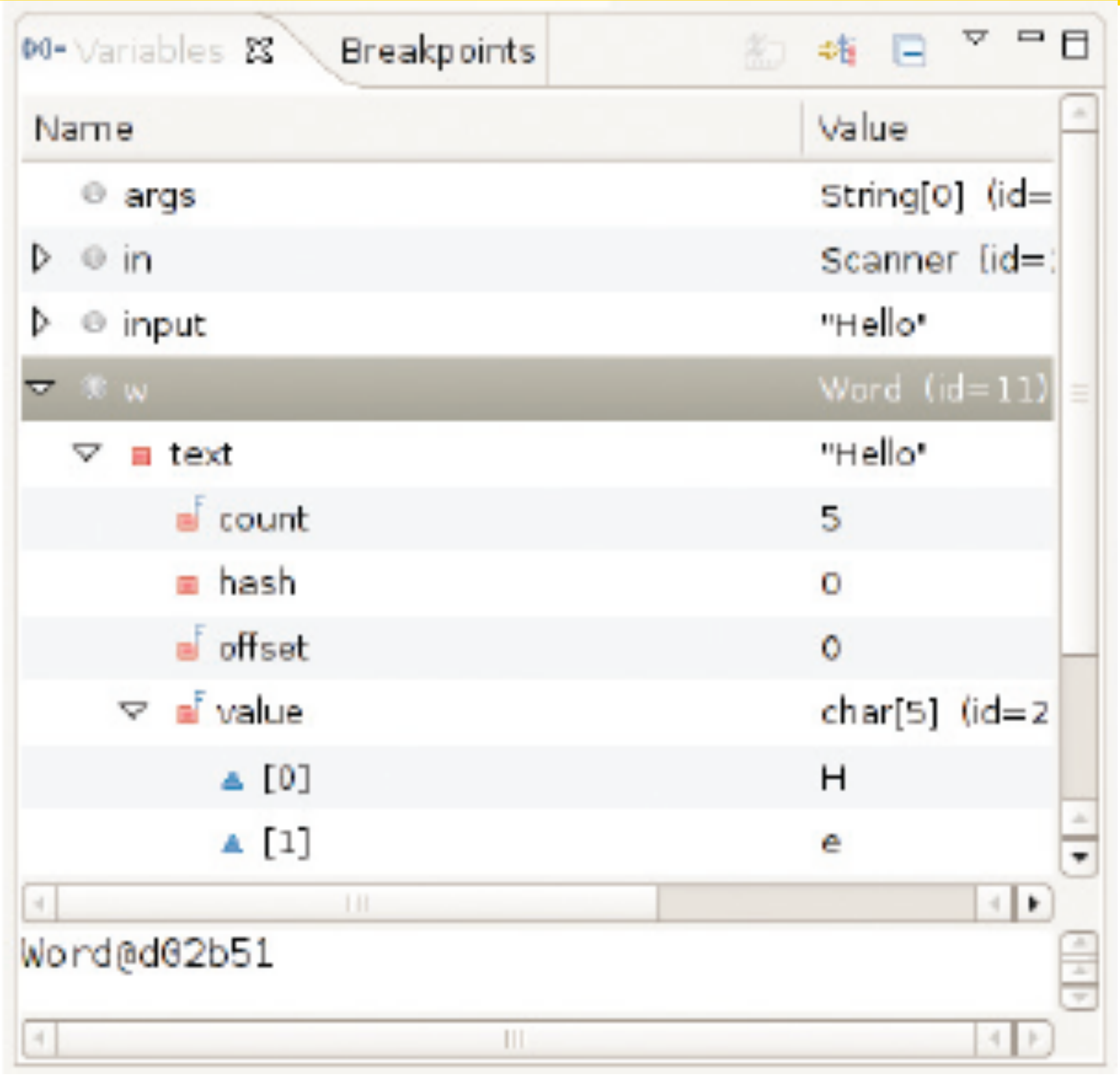
```java
      String input;
      do
      {
         input = in.next();
         Word w = new Word(input);
         int syllables = w.countSyllables();
         System.out.println("Syllables in " + input + ": "
            + syllables);
      }
      while (!input.endsWith("."));
   }
}
```

```
SyllableCounter [Java Application] /home/apps/jdk1.6.0-b104/bin/java (Nov 10, 2006 12:28:46 PM)
Enter a sentence ending in a period.
hello yellow peach.
```

Figure 5 Stopping at a Breakpoint

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

Inspecting Variables



Name	Value
args	String[0] (id=
in	Scanner (id=
input	"Hello"
w	Word (id=11)
text	"Hello"
count	5
hash	0
offset	0
value	char[5] (id=2
[0]	H
[1]	e

```
Word@d02b51
```

Figure 6
Inspecting Variables

Debugging

- Execution is suspended whenever a breakpoint is reached
- In a debugger, a program runs at full speed until it reaches a breakpoint
- When execution stops you can:
  - *Inspect variables*
  - *Step through the program a line at a time*
  - *Or, continue running the program at full speed until it reaches the next breakpoint*
- When program terminates, debugger stops as well
- Breakpoints stay active until you remove them
- Two variations of single-step command:
  - *Step Over: skips method calls*
  - *Step Into: steps inside method calls*

Single-step Example

Current line:

```
String input = in.next();
Word w = new Word(input);
int syllables = w.countSyllables();
System.out.println("Syllables in " + input + ": " +
   syllables);
```

When you step over method calls, you get to the next line:

```
String input = in.next();
Word w = new Word(input);
int syllables = w.countSyllables();
System.out.println("Syllables in " + input + ": " +
   syllables);
```

Continued

Single-step Example (cont.)

However, if you step into method calls, you enter the first line of the `countSyllables` method

```
public int countSyllables()
{
   int count = 0;
   int end = text.length() - 1;
   . . .
}
```

Self Check 6.11

In the debugger, you are reaching a call to `System.out.println`. Should you step into the method or step over it?

Answer: You should step over it because you are not interested in debugging the internals of the `println` method.

Self Check 6.12

In the debugger, you are reaching the beginning of a long method with a couple of loops inside. You want to find out the return value that is computed at the end of the method. Should you set a breakpoint, or should you step through the method?

Answer: You should set a breakpoint. Stepping through loops can be tedious.

Sample Debugging Session

- `Word` class counts syllables in a word
- Each group of adjacent vowels (`a`, `e`, `i`, `o`, `u`, `y`) counts as one syllable
- However, an e at the end of a word doesn't count as a syllable
- If algorithm gives count of `0`, increment to `1`
- Constructor removes non-letters at beginning and end

ch06/debugger/Word.java

```java
/**
   This class describes words in a document.
*/
public class Word
{
   /**
      Constructs a word by removing leading and trailing non-
      letter characters, such as punctuation marks.
      @param s the input string
   */
   public Word(String s)
   {
      int i = 0;
      while (i < s.length() && !Character.isLetter(s.charAt(i)))
         i++;
      int j = s.length() - 1;
      while (j > i && !Character.isLetter(s.charAt(j)))
         j--;
      text = s.substring(i, j);
   }

```

Continued

ch06/debugger/Word.java (cont.)

```
   /**
      Returns the text of the word, after removal of the
      leading and trailing non-letter characters.
      @return the text of the word
   */
   public String getText()
   {
      return text;
   }

   /**
      Counts the syllables in the word.
      @return the syllable count
   */
   public int countSyllables()
   {
      int count = 0;
      int end = text.length() - 1;
      if (end < 0) return 0; // The empty string has no syllables

```

Continued

ch06/debugger/Word.java (cont.)

```
      // An e at the end of the word doesn't count as a vowel
      char ch = Character.toLowerCase(text.charAt(end));
      if (ch == 'e') end--;
      boolean insideVowelGroup = false;
      for (int i = 0; i <= end; i++)
      {
         ch = Character.toLowerCase(text.charAt(i));
         String vowels = "aeiouy";
         if (vowels.indexOf(ch) >= 0)
         {
            // ch is a vowel
            if (!insideVowelGroup)
            {
               // Start of new vowel group
               count++;
               insideVowelGroup = true;
            }
         }
      }

```

Continued

ch06/debugger/Word.java (cont.)

```
      // Every word has at least one syllable
      if (count == 0)
         count = 1;

      return count;
   }

   private String text;
}
```

ch06/debugger/SyllableCounter.java

```java
import java.util.Scanner;

/**
   This program counts the syllables of all words in a sentence.
*/
public class SyllableCounter
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.println("Enter a sentence ending in a period.");

      String input;
      do
      {
         input = in.next();
         Word w = new Word(input);
         int syllables = w.countSyllables();
         System.out.println("Syllables in " + input + ": "
            + syllables);
      }
```

Continued

ch06/debugger/SyllableCounter.java (cont.)

```
      while (!input.endsWith("."));
   }
}
```

Debug the Program

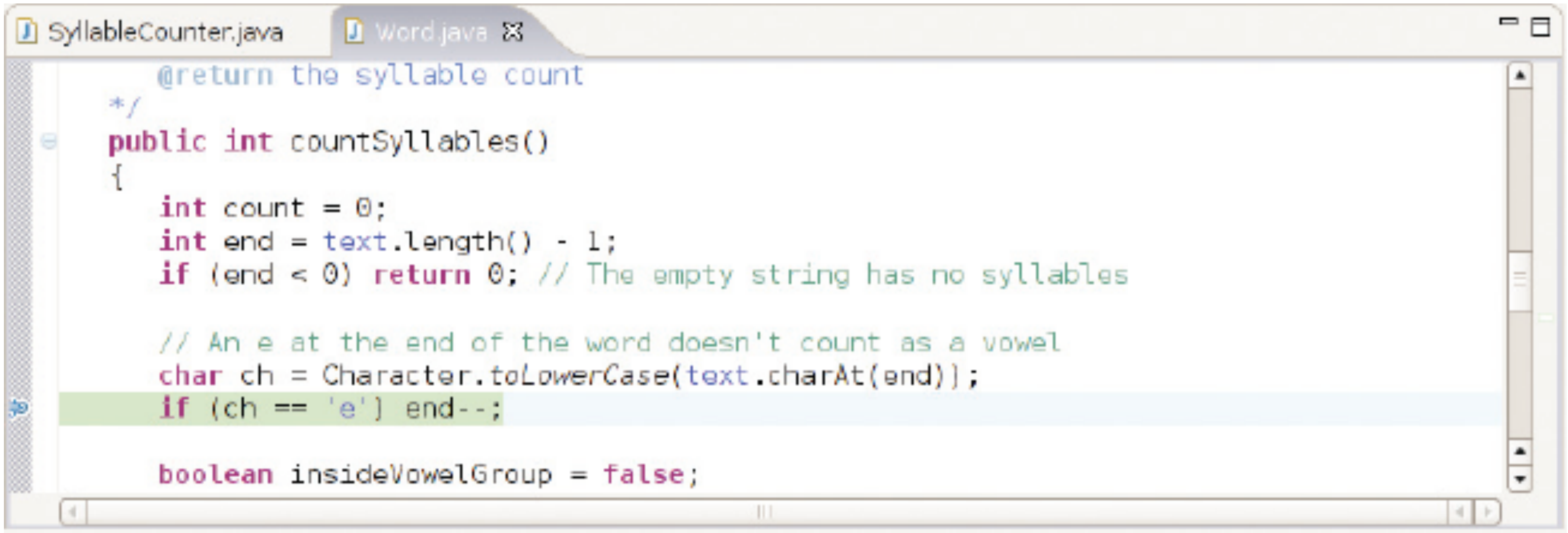
- Buggy output (for input "`hello yellow peach.`"):

```
Syllables in hello: 1
Syllables in yellow: 1
Syllables in peach.: 1
```

- Set breakpoint in first line of `countSyllables` of `Word` class
- Start program, supply input. Program stops at breakpoint
- Method checks if final letter is '`e`'

Continued

Debug the Program (cont.)

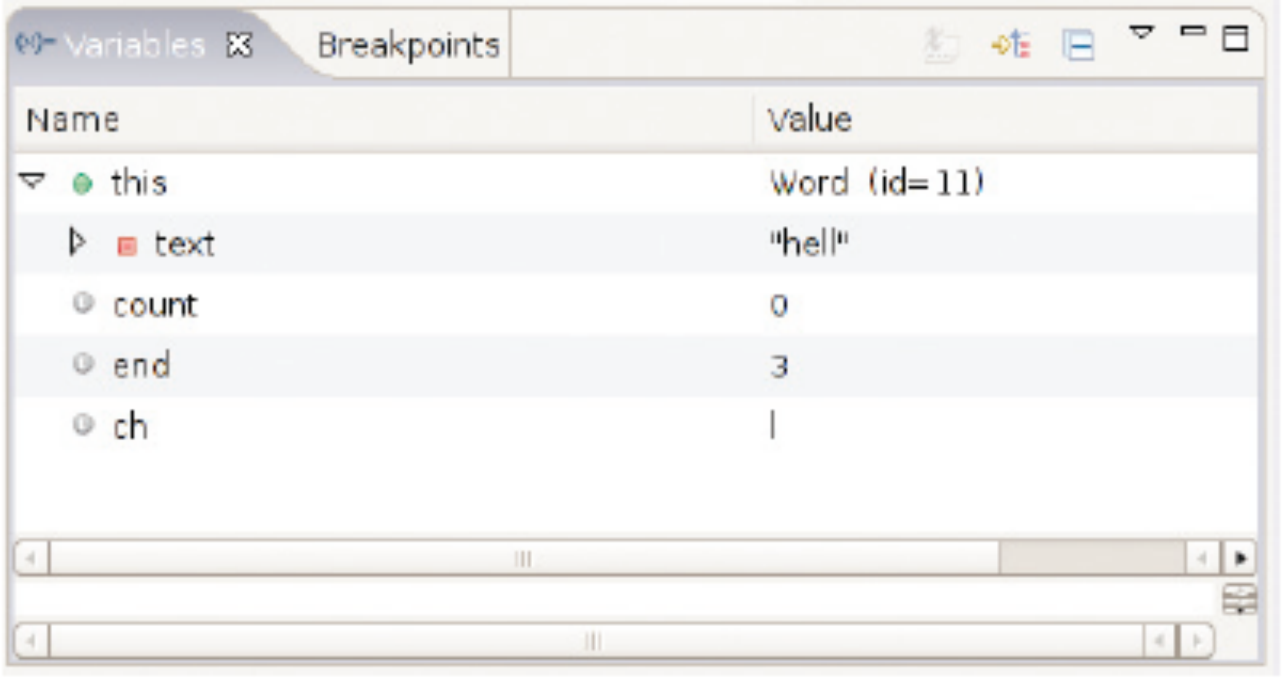


```
      @return the syllable count
    */
    public int countSyllables()
    {
       int count = 0;
       int end = text.length() - 1;
       if (end < 0) return 0; // The empty string has no syllables

       // An e at the end of the word doesn't count as a vowel
       char ch = Character.toLowerCase(text.charAt(end));
       if (ch == 'e') end--;

       boolean insideVowelGroup = false;
```

Figure 7 Debugging the `countSyllables` Method

- Check if this works: step to line where check is made and inspect variable `ch`
- Should contain final letter but contains '`l`'

More Problems Found



Name	Value
this	Word (id=11)
text	"hell"
count	0
end	3
ch	l

Figure 8 The Current Values of the Local and Instance Variables

- `end` is set to 3, not 4
- `text` contains "`hell`", not "`hello`"
- No wonder `countSyllables` returns 1

Continued

More Problems Found (cont.)

- Culprit is elsewhere
- Can't go back in time
- Restart and set breakpoint in `Word` constructor

Debugging the Word Constructor

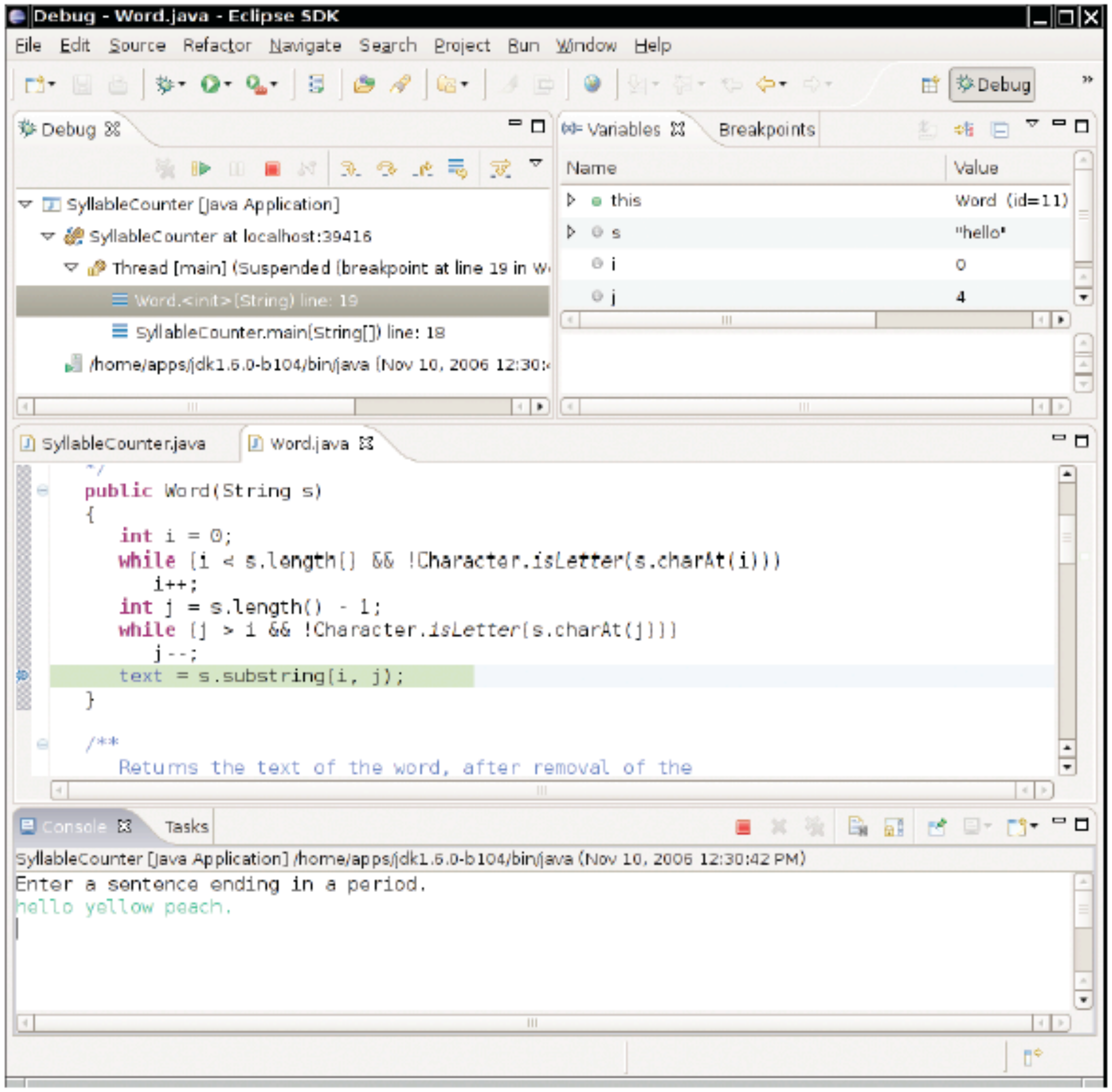
- Supply "`hello`" input again
- Break past the end of second loop in constructor
- Inspect `i` and `j`
- They are 0 and 4 – makes sense since the input consists of letters
- Why is `text` set to "`hell`"?
- Off-by-one error: Second parameter of `substring` is the first position *not* to include

```
text = substring(i, j);
```

  should be

```
text = substring(i, j + 1);
```

Debugging the Word Constructor


Debug - Word.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Variables
Breakpoints
SyllableCounter.java
Word.java
Console
Tasks

```
SyllableCounter [Java Application]
  SyllableCounter at localhost:39416
    Thread [main] (Suspended (breakpoint at line 19 in W
      Word.<init>(String) line: 19
      SyllableCounter.main(String[]) line: 18
  /home/apps/jdk1.6.0-b104/bin/java (Nov 10, 2006 12:30:
```

Name	Value
this	Word (id=11)
s	"hello"
i	0
j	4

```
    */
   public Word(String s)
   {
      int i = 0;
      while (i < s.length() && !Character.isLetter(s.charAt(i)))
         i++;
      int j = s.length() - 1;
      while (j > i && !Character.isLetter(s.charAt(j)))
         j--;
      text = s.substring(i, j);
   }

   /**
      Returns the text of the word, after removal of the
```

```
SyllableCounter [Java Application] /home/apps/jdk1.6.0-b104/bin/java (Nov 10, 2006 12:30:42 PM)
Enter a sentence ending in a period.
hello yellow peach.
```

Figure 9 Debugging the Word Constructor

Another Error

- Fix the error
- Recompile
- Test again:

```
 Syllables in hello: 1
Syllables in yellow: 1
Syllables in peach.: 1
```

- Oh no, it's still not right
- Start debugger
- Erase all old breakpoints and set a breakpoint in `countSyllables` method
- Supply input "`hello.`"

Debugging `countSyllables` (again)

Break in the beginning of `countSyllables`. Then, single-step through loop

```
boolean insideVowelGroup = false;
for (int i = 0; i <= end; i++)
{
   ch = Character.toLowerCase(text.charAt(i));
   if ("aeiouy".indexOf(ch) >= 0)
{
     // ch is a vowel
     if (!insideVowelGroup)
     {
         // Start of new vowel group
         count++;
         insideVowelGroup = true;
     }
   }
 }
```

Continued

Debugging `countSyllables` (again)

- First iteration (`'h')`: skips test for vowel
- Second iteration ('`e`'): passes test, increments `count`
- Third iteration ('`l`'): skips test
- Fifth iteration ('`o`'): passes test, but second `if` is skipped, and `count` is not incremented

Fixing the Bug

- `insideVowelGroup` was never reset to `false`
- Fix

```
if ("aeiouy".indexOf(ch) >= 0)
 {
     . . .
 }
 else insideVowelGroup = false;
```

- Retest: All test cases pass

```
Syllables in hello: 2
Syllables in yellow: 2
Syllables in peach.: 1
```

- Is the program now bug-free? The debugger can't answer that.

Self Check 6.13

What caused the first error that was found in this debugging session?

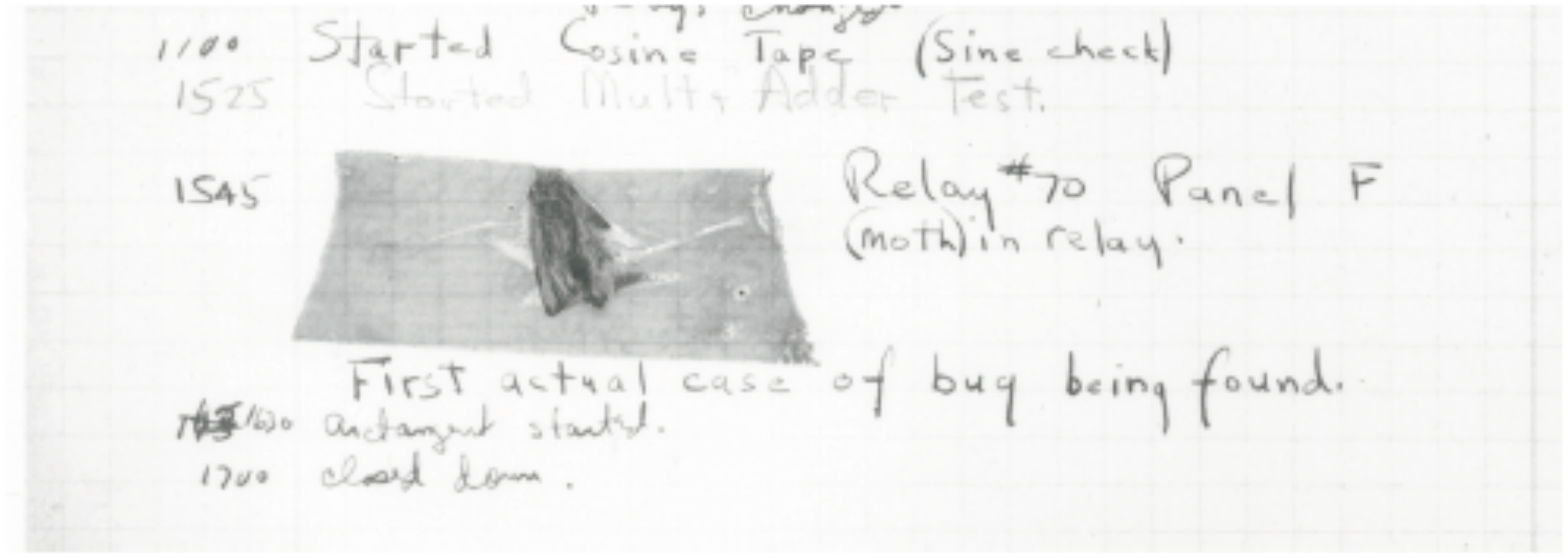
Answer: The programmer misunderstood the second parameter of the substring method–it is the index of the first character not to be included in the substring.

Self Check 6.14

What caused the second error? How was it detected?

Answer: The second error was caused by failing to reset `insideVowelGroup` to false at the end of a vowel group. It was detected by tracing through the loop and noticing that the loop didn't enter the conditional statement that increments the vowel count.

The First Bug



1100 Started Cosine Tape (Sine check)

1525 Started Multy Adder Test.

1545 Relay #70 Panel F (moth) in relay.

First actual case of bug being found.

~~1600~~ 1630 Antangent started.

1700 closed down.

The First Bug