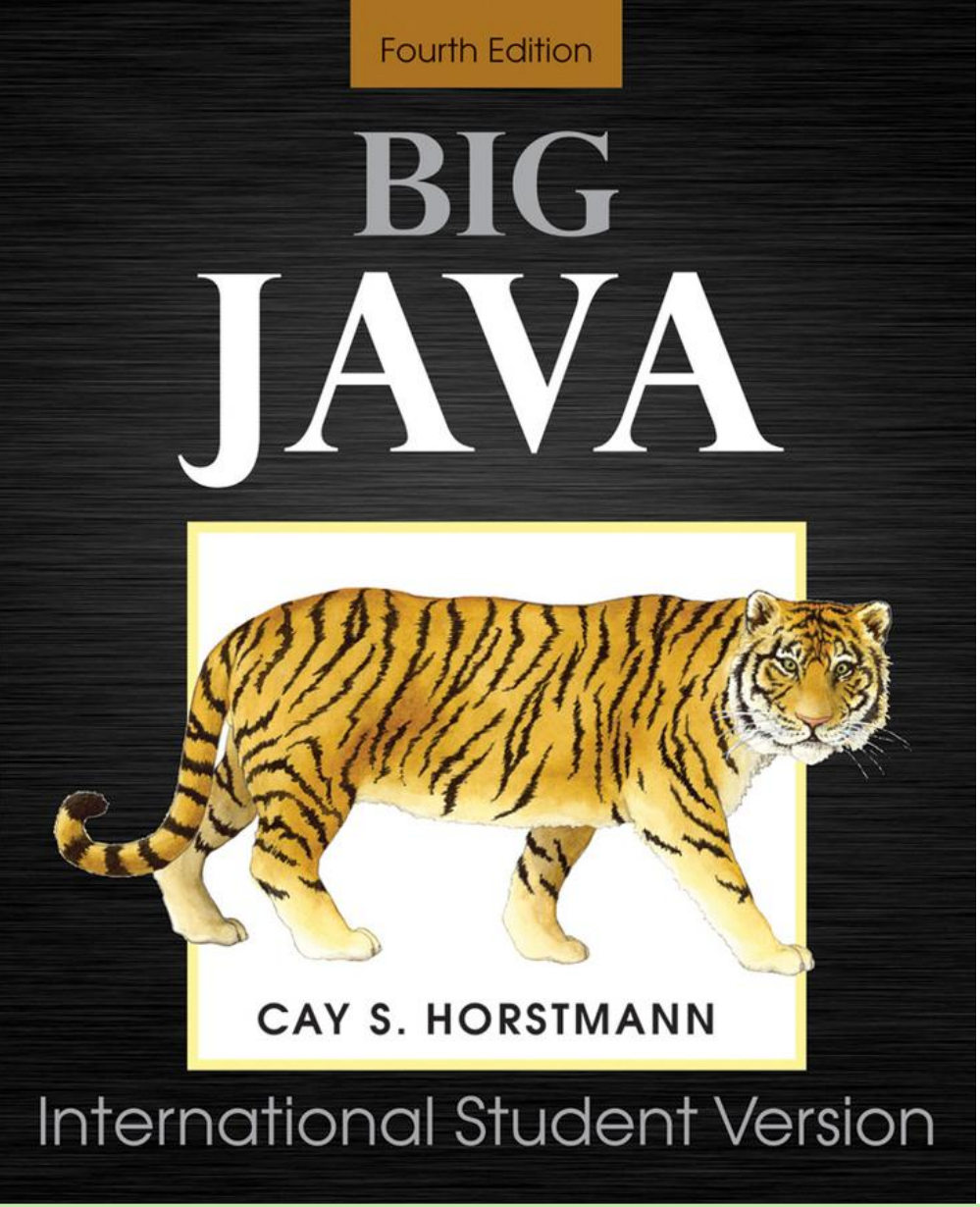


Chapter 5 - Iteration

Chapter Goals

- To be able to program loops with the `while` and `for` statements
- To avoid infinite loops and off-by-one errors
- To be able to use common loop algorithms
- To understand nested loops
- To implement simulations
- To learn about the debugger

`while` Loops

- A `while` statement 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 `{ }`)

`do-while` Loops

- A `do-while` statement executes a block of code repeatedly
- A condition controls **after** the statement how often the loop is executed:

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

- The statement is therefore executed at least once.
- Most commonly, the statement is a block statement (set of statements delimited by `{ }`)

```
do
{
   ...
}
while (condition)
```

Calculating the Growth of an Investment

- Want to know when has the bank account reached a particular balance:

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

ch05/invest1/Investment.java

```
1  /**
2     A class to monitor the growth of an investment that
3     accumulates interest at a fixed annual rate.
4  */
5  public class Investment
6  {
7     private double balance;
8     private double rate;
9     private int years;
10
11    /**
12       Constructs an Investment object from a starting balance and
13       interest rate.
14       @param aBalance the starting balance
15       @param aRate the interest rate in percent
16    */
17    public Investment(double aBalance, double aRate)
18    {
19       balance = aBalance;
20       rate = aRate;
21       years = 0;
22    }
23
```

Continued

ch05/invest1/Investment.java (cont.)

```
24      /**
25         Keeps accumulating interest until a target balance has
26         been reached.
27         @param targetBalance the desired balance
28      */
29      public void waitForBalance(double targetBalance)
30      {
31         while (balance < targetBalance)
32         {
33            years++;
34            double interest = balance * rate / 100;
35            balance = balance + interest;
36         }
37      }
38
39      /**
40         Gets the current investment balance.
41         @return the current balance
42      */
43      public double getBalance()
44      {
45         return balance;
46      }
47
```

Continued

ch05/invest1/Investment.java (cont.)

```
48      /**
49         Gets the number of years this investment has accumulated
50         interest.
51         @return the number of years since the start of the investment
52      */
53      public int getYears()
54      {
55         return years;
56      }
57   }
```

ch05/invest1/InvestmentRunner.java

```
1  /**
2     This program computes how long it takes for an investment
3     to double.
4  */
5  public class InvestmentRunner
6  {
7     public static void main(String[] args)
8     {
9        final double INITIAL_BALANCE = 10000;
10       final double RATE = 5;
11       Investment invest = new Investment(INITIAL_BALANCE, RATE);
12       invest.waitForBalance(2 * INITIAL_BALANCE);
13       int years = invest.getYears();
14       System.out.println("The investment doubled after "
15             + years + " years");
16    }
17 }
```

ch05/invest1/InvestmentRunner.java (cont.)

Program Run:

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

`while` Loop Flowchart

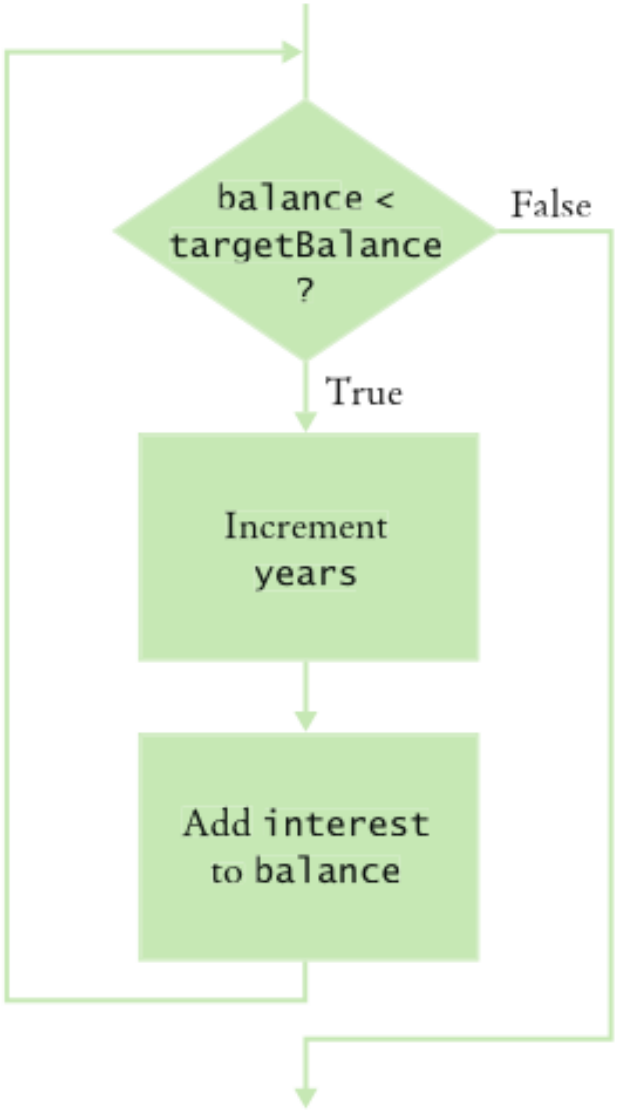


Figure 2 Flowchart of a `while` Loop

Self Check 5.1

How often is the following statement in the loop executed?

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

Answer: Never.

Self Check 5.2

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

- Example:

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

- Loop runs forever — must kill program

Common Error: Infinite Loops

- Example:

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

- Loop runs forever — must kill program

Common Error: Off-by-One Errors

- **Off-by-one error:** a loop executes one too few, or one too many, times
- Example:

```
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

`for` Loops

- Example:

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

- Use a `for` loop when a variable runs from a starting value to an ending value with a constant increment or decrement

`for` Loop Flowchart

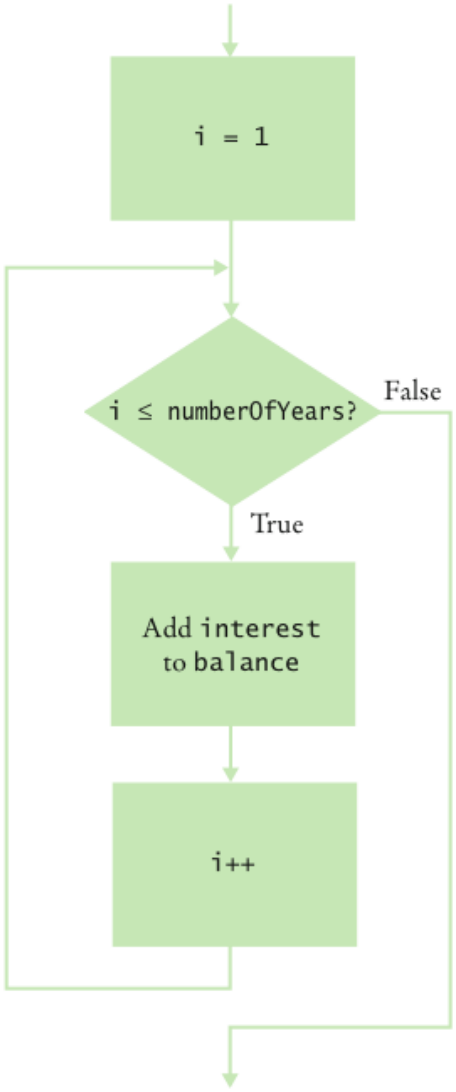


Figure 3 Flowchart of a for Loop

ch05/invest2/Investment.java

```
1  /**
2     A class to monitor the growth of an investment that
3     accumulates interest at a fixed annual rate
4  */
5  public class Investment
6  {
7     private double balance;
8     private double rate;
9     private int years;
10
11    /**
12       Constructs an Investment object from a starting balance and
13       interest rate.
14       @param aBalance the starting balance
15       @param aRate the interest rate in percent
16    */
17    public Investment(double aBalance, double aRate)
18    {
19       balance = aBalance;
20       rate = aRate;
21       years = 0;
22    }
23
```

Continued

ch05/invest2/Investment.java (cont.)

```
24      /**
25         Keeps accumulating interest until a target balance has
26         been reached.
27         @param targetBalance the desired balance
28      */
29      public void waitForBalance(double targetBalance)
30      {
31         while (balance < targetBalance)
32         {
33            years++;
34            double interest = balance * rate / 100;
35            balance = balance + interest;
36         }
37      }
38
39      /**
40         Keeps accumulating interest for a given number of years.
41         @param numberOfYears the number of years to wait
42      */
43      public void waitYears(int numberOfYears)
44      {
45         for (int i = 1; i <= numberOfYears; i++)
46         {
47            double interest = balance * rate / 100;
48            balance = balance + interest;
49         }
50         years = years + n;
51      }
```

Continued

ch05/invest2/Investment.java (cont.)

```
52
53      /**
54         Gets the current investment balance.
55         @return the current balance
56      */
57      public double getBalance()
58      {
59         return balance;
60      }
61
62      /**
63         Gets the number of years this investment has accumulated
64         interest.
65         @return the number of years since the start of the investment
66      */
67      public int getYears()
68      {
69         return years;
70      }
71   }
```

ch05/invest2/InvestmentRunner.java

```
1  /**
2     This program computes how much an investment grows in
3     a given number of years.
4  */
5  public class InvestmentRunner
6  {
7     public static void main(String[] args)
8     {
9        final double INITIAL_BALANCE = 10000;
10       final double RATE = 5;
11       final int YEARS = 20;
12       Investment invest = new Investment(INITIAL_BALANCE, RATE);
13       invest.waitYears(YEARS);
14       double balance = invest.getBalance();
15       System.out.printf("The balance after %d years is %.2f\n",
16             YEARS, balance);
17    }
18 }
```

Program Run:

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

Self Check 5.3

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 5.4

How many times does the following for loop execute?

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

Answer: 11 times.

`for` Loop Examples

Table 2 for Loop Examples

Loop	Values of i	Comment
`for (i = 0; i <= 5; i++)`	0 1 2 3 4 5	Note that the loop is executed 6 times. (See Quality Tip 6.4 on page 240.)
`for (i = 5; i >= 0; i--)`	5 4 3 2 1 0	Use `i--` for decreasing values.
`for (i = 0; i < 9; i = i + 2)`	0 2 4 6 8	Use `i = i + 2` for a step size of 2.
`for (i = 0; i != 9; i = i + 2)`	0 2 4 6 8 10 12 14 ... (infinite loop)	You can use `<` or `<=` instead of `!=` to avoid this problem.
`for (i = 1; i <= 20; i = i * 2)`	1 2 4 8 16	You can specify any rule for modifying `i`, such as doubling it in every step.
`for (i = 0; i < str.length(); i++)`	0 1 2 ... until the last valid index of the string `str`	In the loop body, use the expression `str.charAt(i)` to get the `i`th character.

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);
```

Common Loop Algorithm: Computing a Total

- Example — keep a *running total*: a variable to which you add each input value:

```
double total = 0;
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   total = total + input;
}
```

Common Loop Algorithm: Counting Matches

- Example — count how many uppercase letters are in a string:

```
int upperCaseLetters = 0;
for (int i = 0; i < str.length(); i++)
{
   char ch = str.charAt(i);
   if (Character.isUpperCase(ch))
   {
      upperCaseLetters++;
   }
}
```

Common Loop Algorithm: Finding the First Match

- Example — find the first lowercase letter in a string:

```
boolean found = false;
char ch = '?';
int position = 0;
while (!found && position < str.length())
{
   ch = str.charAt(position);
   if (Character.isLowerCase(ch)) { found = true; }
   else { position++; }
}
```

FIND - Methode der Klasse String anschauen!

Common Loop Algorithm: Comparing Adjacent Values

- Example — check whether a sequence of inputs contains adjacent duplicates such as 1 7 2 9 9 4 9:

```
double input = in.nextDouble();
while (in.hasNextDouble())
{
   double previous = input;
   input = in.nextDouble();
   if (input == previous) { System.out.println("Duplicate input"); }
}
```

Common Loop Algorithm: Processing Input with Sentinel Values

- Example — process a set of values
- **Sentinel value:** Can be used for indicating the end of a data set
- `0` or `-1` make poor sentinels; better to 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
   }
}
```

ch05/dataset/DataAnalyzer.java

```
1  import java.util.Scanner;
2
3  /**
4     This program computes the average and maximum of a set
5     of input values.
6  */
7  public class DataAnalyzer
8  {
9     public static void main(String[] args)
10    {
11       Scanner in = new Scanner(System.in);
12       DataSet data = new DataSet();
13
14       boolean done = false;
15       while (!done)
16       {
17          System.out.print("Enter value, Q to quit: ");
18          String input = in.next();
19          if (input.equalsIgnoreCase("Q"))
20             done = true;
21          else
22          {
23             double x = Double.parseDouble(input);
24             data.add(x);
25          }
26       }
27
```

Continued

ch05/dataset/DataAnalyzer.java (cont.)

```
28          System.out.println("Average = " + data.getAverage());
29          System.out.println("Maximum = " + data.getMaximum());
30       }
31    }
```

ch05/dataset/DataSet.java

```
1  /**
2     Computes information about a set of data values.
3  */
4  public class DataSet
5  {
6     private double sum;
7     private double maximum;
8     private int count;
9
10    /**
11       Constructs an empty data set.
12    */
13    public DataSet()
14    {
15       sum = 0;
16       count = 0;
17       maximum = 0;
18    }
19
```

Continued

ch05/dataset/DataSet.java (cont.)

```
20      /**
21         Adds a data value to the data set
22         @param x a data value
23      */
24      public void add(double x)
25      {
26         sum = sum + x;
27         if (count == 0 || maximum < x) maximum = x;
28         count++;
29      }
30
31      /**
32         Gets the average of the added data.
33         @return the average or 0 if no data has been added
34      */
35      public double getAverage()
36      {
37         if (count == 0) return 0;
38         else return sum / count;
39      }
40
```

Continued

ch05/dataset/DataSet.java (cont.)

```
41      /**
42         Gets the largest of the added data.
43         @return the maximum or 0 if no data has been added
44      */
45      public double getMaximum()
46      {
47         return maximum;
48      }
49   }
```

Program Run:

```
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 5.5

How do you compute the total of all positive inputs?

Answer:

```
double total = 0;
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   if (input > 0) total = total + input;
}
```

The *break* statement

You can also leave a while or for loop with the break statement but it is not recommended because it can lead to hard to find errors.

```
while (true)
{   String input = in.next();
   if (input.equalsIgnoreCase("Q"))
      break;
   double x = Double.parseDouble(input);
   data.add(x);
}
```

The *continue* statement

There is a second statement to leave a while or for loop.

With continue you can jump to the end of the while or for loop:

```
while (!done)
{  String input = in.next();
   if (input.equalsIgnoreCase("Q"))
   {  done = true;
      continue; // Jump to the end of the loop body
   }
   double x = Double.parseDouble(input);
   data.add(x);
   // continue statement jumps here
}
```

Self Check 5.7

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.

Nested Loops

- Create triangle shape:

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

- 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

Nested Loop Examples

Table 3 Nested Loop Examples

Nested Loops	Output	Explanation

`for (i = 1; i <= 3; i++)`
`{`
`  for (j = 1; j <= 4; j++)  { Print "*" }`
`  System.out.println();`
`}`
	`****`
	`****`
	`****`
		Prints 3 rows of 4 asterisks each.

`for (i = 1; i <= 4; i++)`
`{`
`  for (j = 1; j <= 3; j++) { Print "*" }`
`  System.out.println();`
`}`
	`***`
	`***`
	`***`
	`***`
		Prints 4 rows of 3 asterisks each.

Nested Loop Examples

Table 3 Nested Loop Examples, continued

Nested Loops	Output	Explanation

`for (i = 1; i <= 4; i++)`
`{`
`  for (j = 1; j <= i; j++) { Print "*" }`
`  System.out.println();`
`}`
	`*`
	`**`
	`***`
	`****`
		Prints 4 rows of lengths 1, 2, 3, and 4.

`for (i = 1; i <= 3; i++)`
`{`
`  for (j = 1; j <= 5; j++)`
`  {`
`  if (j % 2 == 0) { Print "*" }`
`  else { Print "-" }`
`  }`
`  System.out.println();`
`}`
	`-*-*-`
	`-*-*-`
	`-*-*-`
		Prints asterisks in even columns, dashes in odd columns.

`for (i = 1; i <= 3; i++)`
`{`
`  for (j = 1; j <= 5; j++)`
`  {`
`  if ((i + j) % 2 == 0) { Print "*" }`
`  else { Print " " }`
`  }`
`  System.out.println();`
`}`
	`* * *`
	`  * *`
	`* * *`
		Prints a checkerboard pattern.

Self Check 5.9

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 5.10

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

```
i:1, j: 0, n: 0
i:2, j: 0, n: 0
i:2, j: 1, n: 1
i:3, j: 0, n: 1
i:3, j: 1, n: 2
i:3, j: 2, n: 4
i:4, j: 0, n: 4
i:4, j: 1, n: 5
i:4, j: 2, n: 7
i:4, j: 3, n: 10
i:5, j: 0, n: 10
i:5, j: 1, n: 11
i:5, j: 2, n: 13
i:5, j: 3, n: 16
i:5, j: 4, n: 20
```

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);
```

- A random generator is by default seeded with the current time. To create reproducable result you must seed with a constant

```
Random generator = new Random(0);
```

ch05/random1/Die.java

```
 1  import java.util.Random;
 2
 3  /**
 4     This class models a die that, when cast, lands on a random
 5     face.
 6  */
 7  public class Die
 8  {
 9     private Random generator;
10     private int sides;
11
12     /**
13        Constructs a die with a given number of sides.
14        @param s the number of sides, e.g. 6 for a normal die
15     */
16     public Die(int s)
17     {
18        sides = s;
19        generator = new Random();
20     }
21
```

Continued

ch05/random1/Die.java (cont.)

```
22      /**
23         Simulates a throw of the die
24         @return the face of the die
25      */
26      public int cast()
27      {
28         return 1 + generator.nextInt(sides);
29      }
30   }
```

ch05/random1/DieSimulator.java

```
1  /**
2     This program simulates casting a die ten times.
3  */
4  public class DieSimulator
5  {
6     public static void main(String[] args)
7     {
8        Die d = new Die(6);
9        final int TRIES = 10;
10       for (int i = 1; i <= TRIES; i++)
11       {
12          int n = d.cast();
13          System.out.print(n + " ");
14       }
15       System.out.println();
16    }
17 }
```

Continued

ch05/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

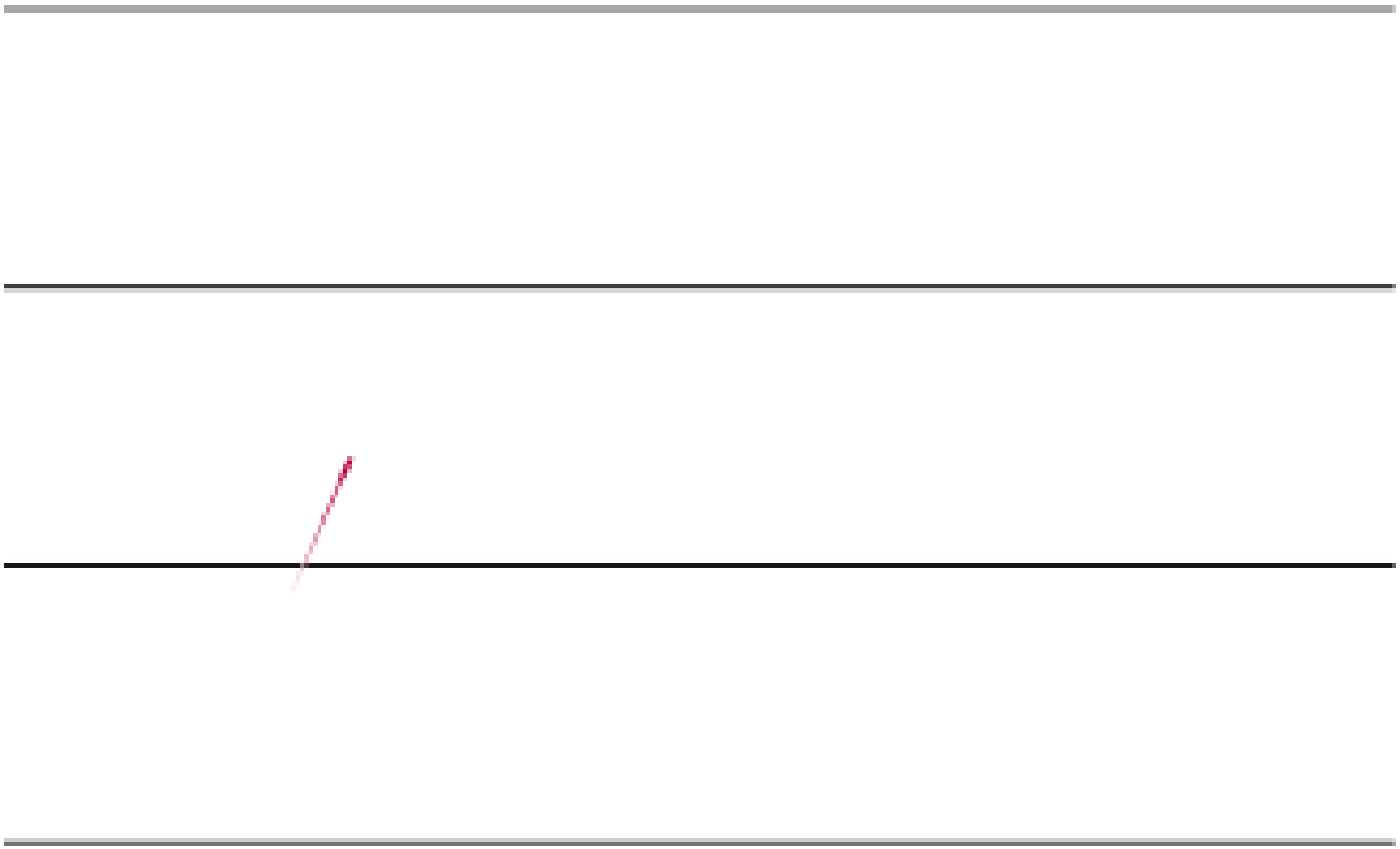
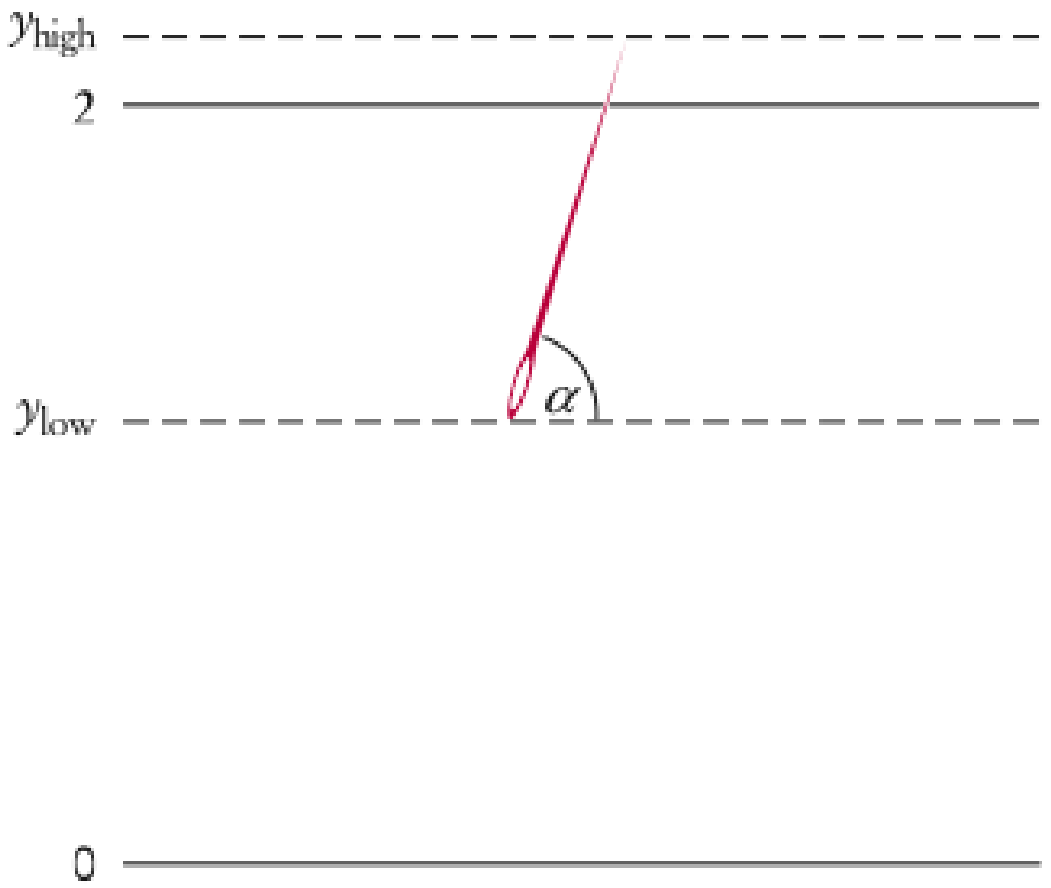


Figure 5 The Buffon Needle Experiment

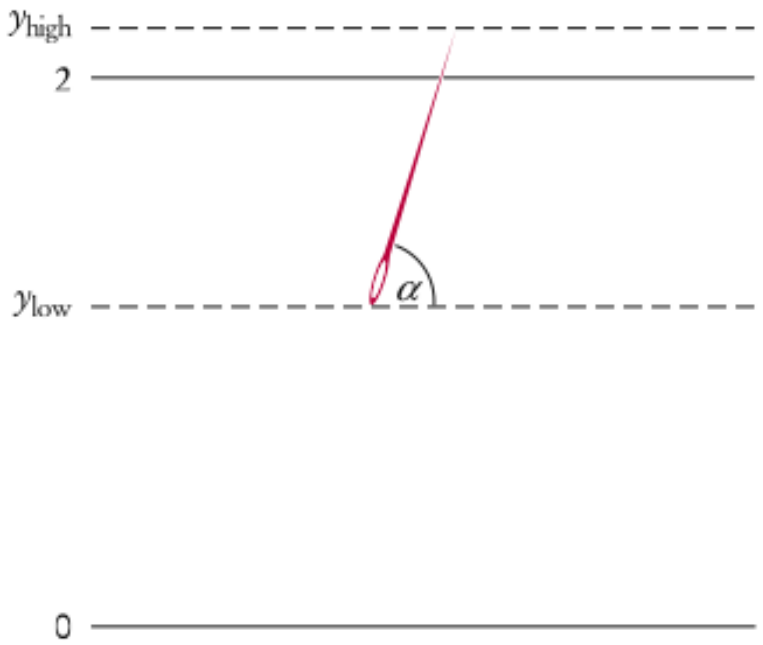
Buffon Needle Experiment

Figure 6
When Does the Needle Fall on a Line?



Needle Position

- Needle length = 1, distance between lines = 2
- Generate random y_{low} between 0 and 2
- Generate random angle α between 0 and 180 degrees
- $y_{high} = y_{low} + \sin(\alpha)$
- Hit if $y_{high} \geq 2$



ch05/random2/Needle.java

```
1  import java.util.Random;
2
3  /**
4     This class simulates a needle in the Buffon needle experiment.
5  */
6  public class Needle
7  {
8     private Random generator;
9     private int hits;
10    private int tries;
11
12    /**
13       Constructs a needle.
14    */
15    public Needle()
16    {
17       hits = 0;
18       tries = 0;
19       generator = new Random();
20    }
21
```

Continued

ch05/random2/Needle.java (cont.)

```
22      /**
23         Drops the needle on the grid of lines and
24         remembers whether the needle hit a line.
25      */
26      public void drop()
27      {
28         double ylow = 2 * generator.nextDouble();
29         double angle = 180 * generator.nextDouble();
30
31         // Computes high point of needle
32
33         double yhigh = ylow + Math.sin(Math.toRadians(angle));
34         if (yhigh >= 2) hits++;
35         tries++;
36      }
37
38      /**
39         Gets the number of times the needle hit a line.
40         @return the hit count
41      */
42      public int getHits()
43      {
44         return hits;
45      }
```

Continued

ch05/random2/Needle.java (cont.)

```
46
47       /**
48          Gets the total number of times the needle was dropped.
49          @return the try count
50       */
51       public int getTries()
52       {
53          return tries;
54       }
55    }
```

ch05/random2/NeedleSimulator.java

```
1  /**
2     This program simulates the Buffon needle experiment
3     and prints the resulting approximations of pi.
4  */
5  public class NeedleSimulator
6  {
7     public static void main(String[] args)
8     {
9        Needle n = new Needle();
10       final int TRIES1 = 10000;
11       final int TRIES2 = 1000000;
12
13       for (int i = 1; i <= TRIES1; i++)
14          n.drop();
15       System.out.printf("Tries = %d, Tries / Hits = %8.5f\n",
16             TRIES1, (double) n.getTries() / n.getHits());
17
18       for (int i = TRIES1 + 1; i <= TRIES2; i++)
19          n.drop();
20       System.out.printf("Tries = %d, Tries / Hits = %8.5f\n",
21             TRIES2, (double) n.getTries() / n.getHits());
22    }
23 }
```

Continued

ch05/random2/NeedleSimulator.java (cont.)

Program Run:

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

Self Check 5.11

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 5.12

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:** a program to execute 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

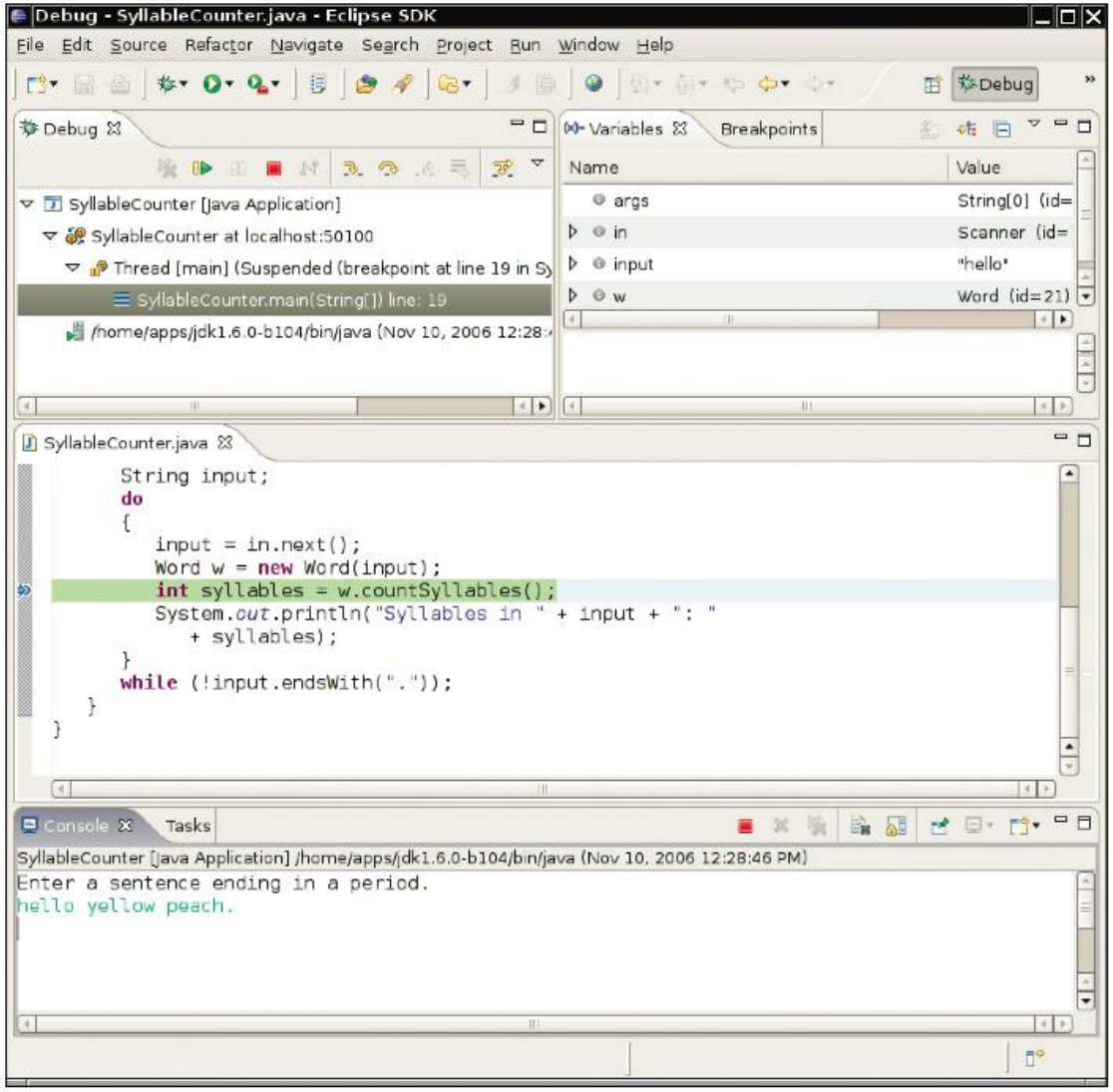
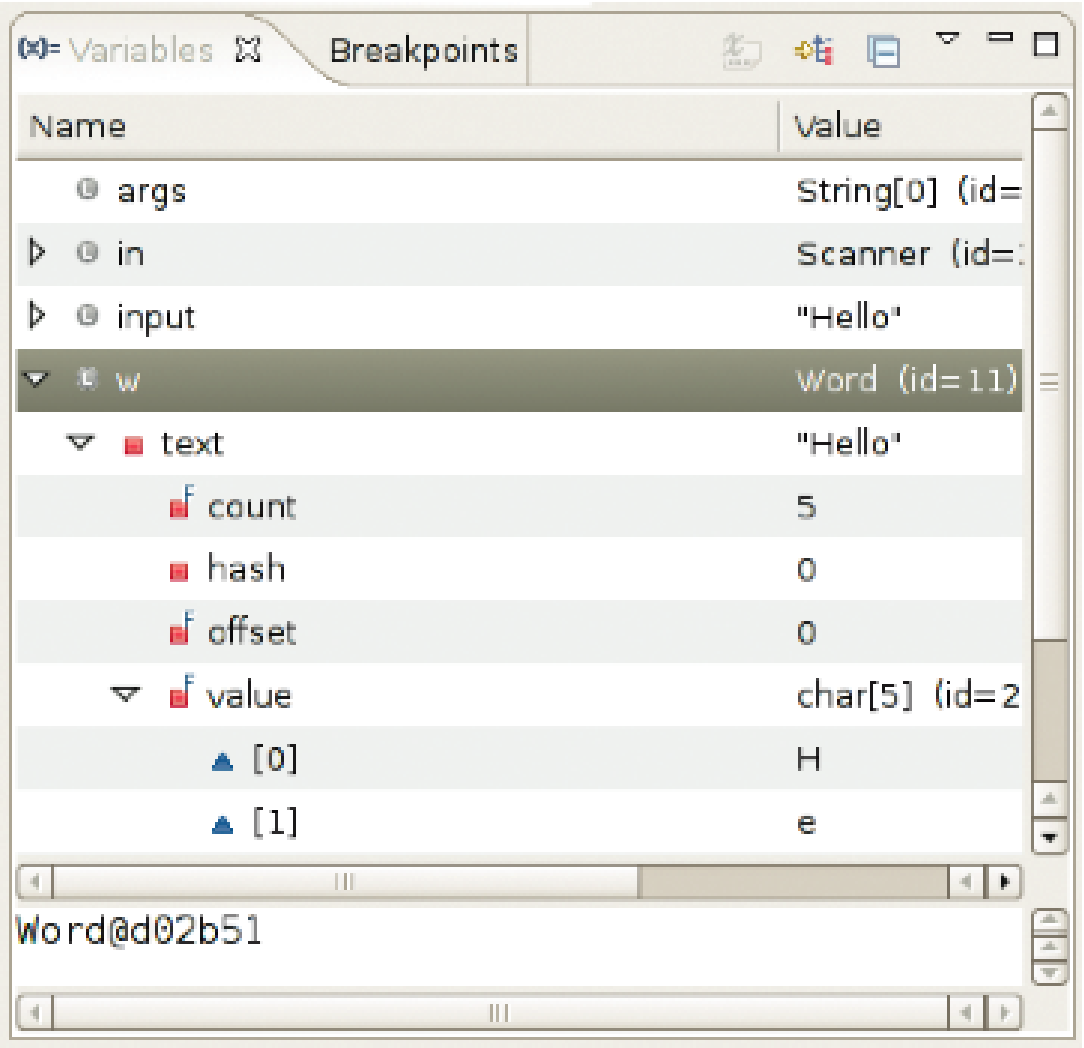


Figure 7 Stopping at a Breakpoint

Inspecting Variables

Figure 8
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 5.13

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 5.14

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.

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.
1700 closed down.

The First Bug

Programming Exercise for chapter 5: Iterations

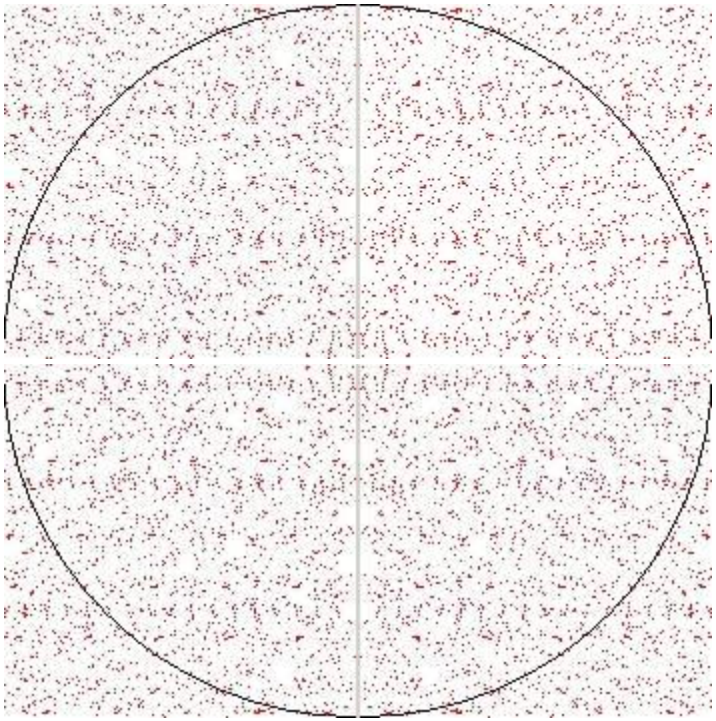
Implementieren Sie **2 Aufgaben** aus dem Bereich **String3** von

http://codingbat.com/java/String-3

Programming Exercise for chapter 5: Iterations

Berechnen Sie die Zahl PI, indem Sie Zufallspunkte in einem Quadrat zählen. Über das Verhältnis der Punkte innerhalb des Quadrats zur Anzahl Punkte innerhalb des Kreises können Sie die Zahl PI berechnen.

1. Generieren Sie eine grafische Ausgabe.
2. Konvergiert diese Methode schneller als die Methode von Buffon?
3. Wie viele Punkte und Sekunden braucht es für Double-Genauigkeit?
4. Wer schafft die höchste Genauigkeit innerhalb einer Bestimmten Zeit (ohne grafischen Outputs)?



5. Abgabe und Wettbewerb am Dienstag 22.10.2013, 18:15 Uhr