Honors Computer Programming 1-2
Introduction To Chapter 6
Iteration

Chapter Goals

- To be able _____
- To avoid _____
- To understand _____
- To _____

While Loops

In this chapter we will look at programs that _____ execute one or more statements. Suppose we open a bank account with an initial deposit of $10,000. The account earns 5% interest with the interest calculation at the end of each year and then deposited into the bank account. How many years does it take for the balance to reach $20,000?

In Java, the _____ statement implements a repetition. A while statement executes a _____ repeatedly. A _____ condition controls _____ the loop is executed. The general form of the while statement is:

```
while(condition)
   statement
```

In our case we want to know when the bank account has reached a _____ . While the balance is _____ we keep _____ interest and incrementing the _____ counter:

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

Here is the complete program that solves our investment problem:

```
public class Investment
{
  public Investment(double balance, double rate)      // constructor
  {
     this.balance = balance;
     this.rate = rate;
     years = 0;
  }

  // accumulates interest until a target balance has been reached
  public void waitForBalance(double targetBalance)
  {
      while (balance < targetBalance)
      {
         years++;
         double interest = balance * rate / 100;
         balance = balance + interest;
      }
   }
```

continued on the next page . . . .

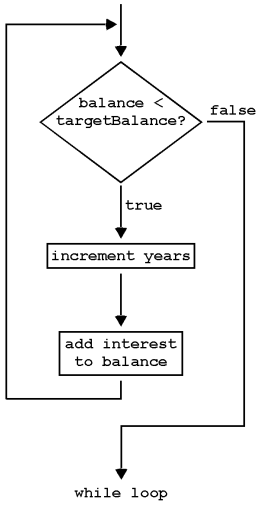
```
   // gets the current balance
   public double getBalance( )
   {
      return balance;
   }

   // gets the number of years this investment has accumulated interest
   public int getYears( )
   {
      return years;
   }

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

```
public class InvestmentTest1
{
   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 would be doubled after " + years + " years");
    }
}
```

A while statement is often called a _____ . The flowchart shows that the control loops _____ to the _____ after every _____ .



The while loop shown below

```
while (true)
{
   body
}
```

executes the _____ over and over without ever _____ . Some programs never exit (examples _____ or _____) but our programs are not usually of that kind. But even if you can't terminate the loop, you can _____ from the method that contains it.

Infinite Loop Error

The most annoying loop error is an _____ which is a loop that can only be stopped by killing the program or restarting the computer.

A common reason for infinite loops is forgetting to advance the variable that _____ the loop:

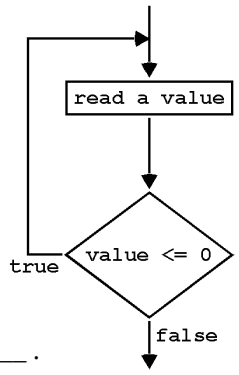
```
int years = 0;
while (years < 20)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

Here the programmer forgot to add a _____ command in the loop. As a result the value of **`years`** always stays _____ , and the loop never comes to an _____ .

do Loops

Sometimes you want the body of a loop to execute _____ and perform the _____ after the body was executed. The _____ loop serves that purpose.

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



do while loop

For example, suppose you want to make sure that a user enters a positive number. As long as the user enters a _____ number or _____ just keep prompting for a correct input. In this case, a _____ makes sense because you need to get a user input _____ you can _____.

```
double value;
do
{
   String input = JOptionPane.showInputDialong("Enter a positive number");
   value = Double.parseDouble(input);
}
while (value <= 0);
```

In practice, this situation is _____. You can always replace a _____ loop with a _____ loop by introducing a _____ control variable.

```
boolean done = false;
while (!done)
{
   String input = JOptionPane.showInputDialog("Enter a positive number");
   value = Double.parseDouble(input);
   if (value > 0)
     done = true;
}
```

For Loops

The most common loop has the form:

```
i = start;
while (i <= end)
{
   ...
   i++;
}
```

Because this form is so common there is a special form for it that emphasizes the patterns

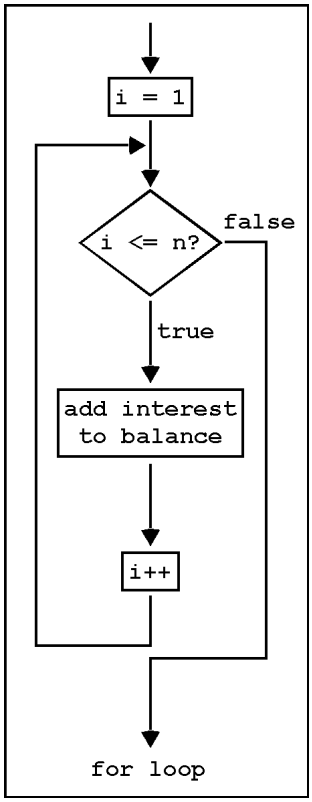
```
for (i = start;  i <= end;  i++)
{
   ...
}
```

You can also _____ the loop counter inside the for loop header:

```
for (int i = start;  i <= end;  i++)
{
   ...
}
```

Let us use this loop to find out the size of our $10,000 investment if 5% interest is compounded for 20 years. Remember that $500 is added every year.

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



Below is the code for Investment.java and **`InvestmentTest.java`** with an additional method **`waitYears`** that contains a **`for`**-loop.

```
public class Investment
{
 public Investment(double balance, double rate)      // constructor
 {
    this.balance = balance;
    this.rate = rate;
    years = 0;
 }

 // accumulates interest until a target balance has been reached
 public void waitForBalance(double targetBalance)
 {
      while (balance < targetBalance)
      {
         years++;
         double interest = balance * rate / 100;
         balance = balance + interest;
      }
  }

  // keeps accumulating interest for a given number of years
  public void waitYears(int n)
  {
     for (int i = 1;  i <= n;  i++)
     {
        double interest = balance * rate / 100;
        balance = balance + interest;
     }
  }

  // gets the current balance
  public double getBalance( )
  {
     return balance;
  }

  // gets the number of years this investment has accumulated interest
  public int getYears( )
  {
     return years;
  }

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

```
public class InvestmentTest2
{
   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 %1d years is $%,1.2f%n", YEARS, balance);
   }
}
```

The three slots in the for header can contain any three expressions. You can count _____ instead of ____ :

```
for (years = n;  years > 0;  years--) ...
```

The increment or decrement need not be in steps of _____ :

```
for (x = -10;  x <= 10;  x = x + 0.5) ...
```

It is possible, but a sign of _____ , to put _____ into the loop:

```
for (rate = 5;  years-- > 0;  System.out.println(balance)) ... // Bad taste
```

You should stick with for loops that _____ , _____ , and _____ a single variable.

Use for Loops For Their Intended Purpose Only

A for loop is an _____ for a _____ loop of a particular form. A _____ runs from the _____ to the _____ with a _____ increment:

```
for (set counter to start;  test whether counter at end;  update counter by increment)
{
   ...
   // counter, start, end, increment not changed here
}
```

If your loop doesn't match this pattern, don't use the _____ construction.

Scope of Variables Defined in a for Loop Header

It is legal in Java to declare a variable in the _____ of a for loop. Here is the most common form of this syntax:

```
for (int i = 1;  i <= n;  i++)
{
   ...
}

// i no longer defined here
```

The scope of the variables extends to the _____ of the for loop. Therefore, ___ is no longer defined when the loop ends. If you need to use the value of the variable beyond the end of the loop, then you need to define it _____ the loop.

In the loop header, you can declare multiple variables, as long as they are of the _____ and you can include multiple _____ separated by _____ :

```
for (int i = 0, j = 10;  i <= 10;  i++, j--)  ...
```

Many people find it _____ if a for loop controls more than one _____ . It is not recommended to use this type of for statement. Instead, make the for loop control a _____ counter and _____ the other variable explicitly.

```
int j = 10;
for (int i = 0;  i <= 10;  i++)
{
   ...
   j--;
}
```

A Semicolon Too Many

What does the loop at the right print? This loop is supposed to compute 1 + 2 + · · · + 10 which is 55. But actually, the loop prints _____ .

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

Did you spot the _____ at the end of the for loop? The loop really is a loop with an _____ .

```
for (i = 1;  i <= 10;  i++)
   ;
```

The loop does _____ 10 times and when finished, **sum =** ____ and **i =** _____ . Then the statement **sum =sum + i;** makes **sum =** _____ .

Nested Loops

Suppose you need to print the following triangle shape:

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

You have to generate a number of rows as shown at the right.

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

How do you make a triangle row? Use another _____ for the squares in that row. Then add a _____ at the end of the row. The **ith** row has **i** symbols so the loop counter goes from _____ . The code for a row is shown at the right.

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

Putting these two loops together yields two _____ as shown at the right.

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

The complete program is shown below.

```
public class Triangle
{
   public Triangle(int aWidth)     // constructor
   {
      width = aWidth;
   }

   // computes a string representing the triangle
   public String toString( )
   {
      String r = "";
      for (int i = 1;  i <= width;  i++)
      {
         // make a triangle row
         for (int j = 1;  j <= i;  j++)
            r = r + "[ ] ";
         r = r + "\n";
      }
      return r;
   }

   private int width;
}
```

```
public class TriangleTest
{
   public static void main(String[ ] args)
   {
      Triangle small = new Triangle(3);
      System.out.println(small.toString( ));

      Triangle large = new Triangle(6);
      System.out.println(large.toString( ));
   }
}
```

Processing Input

Suppose you want to process a set of values. For reading input, you can use the _____ method of the **JOptionPane** class. Or you can use the _____ method to read an **int**, the _____ method to read a **double**, the _____ method to read a word , or the _____ method to read a line of text all from the **Scanner** class.

The loop shown at the right reads through input data. This loop is a little different from earlier examples because the test condition is a variable _____ . That variable stays _____ until you reach the end of _____ ; then it is set to _____ . The next time the loop starts at the top, done is _____ and the loop _____ .

There is a reason for using a variable. The test for loop termination occurs in the _____ of the loop, not at the top or the bottom. You must first try to _____ before you can test whether you have reached the _____ .

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

Let's write a program that analyzes a set of values. This will use a class DataSet. You add values to a DataSet object with the ____ method. The _____ method returns the average of all added data and the _____ method returns the largest.

```
public class DataSet
{
   public DataSet( )  // creates an empty set
   {
      sum = 0;
      count = 0;
      maximum = 0;
   }

   public void add(double x)
   {
      sum = sum + x;
      if (count == 0 || x > maximum)
         maximum = x;
      count++;
   }

   public double getAverage( )
   {
      if (count == 0) return 0;
      else return sum / count;
   }

   public double getMaximum( )
   {
      return maximum;
   }

   public int getCount( )
   {
      return count;
   }

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

```
public class JOptionPaneInputTest
{
   public static void main(String[ ] args)
   {
      DataSet data = new DataSet( );
      boolean done = false;
      while (!done)
      {
         String input = JOptionPane.showInputDialog("Enter value, Cancel to quit");
         if (input == null)
            done = true;
         else
         {
            double x = Double.parseDouble(input);
            data.add(x);
         }
      }

      System.out.println("Number of data values: " + data.getCount( ));
      System.out.println("Average = " + data.getAverage( ));
      System.out.println("Maximum = " + data.getMaximum( ));
   }
}
```

The method of exiting the loop using the _____ is called the "Loop and a Half" method since loop exit is in the middle of the loop. Another technique of exiting a loop that is preferred by some programmers involves the use of the _____ statement. The **break** statement was used in chapter 5 to exit a _____ statement. A **break** can also be used to exit a _____ , _____ , or ____ loop. In this example, the **break** statement is used to _____ the loop when the _____ is reached.

```
while(true)
{
   String input = JOptionPane.showInputDialog("Enter value, Cancel to quit");

   if (input == null)
      break;
   double x = Double.parseDouble(input);
   data.add(x);
}
```

Reading Data from the Console

Reading from the console is done with the _____ class. The code at the right is a modified version of the input test with input from the console.

Note that there is a _____ to the user _____ the **while** loop. The loop continues to run until _____ is changed to _____ .

```
public class ConsoleInputTest
{
   public static void main(String[] args)
   {
      DataSet data = new DataSet( );
      Scanner console = new Scanner(System.in);
      boolean done = false;
      while (!done)
      {
         System.out.print("Enter value, Q to quit: ");
         String input = console.next( );
         if (input.equalsIgnoreCase("Q"))
            done = true;
         else
         {
            double x = Double.parseDouble(input);
            data.add(x);
         }
      }

      System.out.println("Number of data values: " + data.getCount( ));
      System.out.println("Average = " + data.getAverage( ));
      System.out.println("Maximum = " + data.getMaximum( ));
   }
}
```

Reading Data Values from a File

The loop needs to be modified when reading an _____ number of data values from a _____ . We will not use a _____ variable to control the loop. Instead, we will use the _____ method or the _____ method of the **`Scanner`** class.

Code for the input test has been modified so that an unknown number of data items can be read from a file.

```
public class FileInputTest
{
   public static void main(String[ ] args) throws FileNotFoundException
   {
      DataSet data = new DataSet( );

      FileReader reader = new FileReader("Data.txt");
      Scanner file = new Scanner(reader);

      while (file.hasNext( ))
      {
         int number = file.nextInt( );
         data.add(number);
      }

      System.out.println("Number of data values: " + data.getCount( ));
      System.out.println("Average = " + data.getAverage( ));
      System.out.println("Maximum = " + data.getMaximum( ));
   }
}
```

Note that when reading data from a file, no _____ are needed. And loop exit will eventually occur at the _____ of the loop.

String Tokenization

Sometimes it is convenient to have an input line that contains _____ items of input data. Suppose an input line contains two numbers: _____ . You can't convert the string **`"5.5  10000"`** to a number but you can break the string into a _____ of strings, each of which represents a separate input item. There is a special class _____ that can break up a string into items, or as they are called _____ . By default, the string tokenizer uses _____ (_____ , _____ , _____) as delimiters. For example, the string "5.5 10000" will be decomposed into two tokens _____ and _____ .

To tokenize a string, you need to construct a **`StringTokenizer`** object and supply the string to be broken up in the _____ : `StringTokenizer tokenizer = new StringTokenizer(input);` . Then keep calling the _____ method to get the next token.

The loop below shows the proper technique. It uses the _____ method to ensure that there are still tokens to be processed.

```
while (tokenizer.hasMoreTokens( ))
{
   String token = tokenizer.nextToken( );
   ...  // do something with token
}
```

If you want to use another separator, such as a _____ to separate the individual values, you need to specify a second argument when you construct the **StringTokenizer** object:

```
StringTokenizer tokenizer = new StringTokenizer(input, ",");
```

Here is a modified version of the input test using the tokenizers:

```
public class TokenizerInputTest
{
   public static void main(String[] args)
   {
      DataSet data = new DataSet( );
      String input = JOptionPane.showInputDialog("Enter several values:");

      StringTokenizer tokenizer = new StringTokenizer(input);

      while (tokenizer.hasMoreTokens( ))
      {
         String token = tokenizer.nextToken( );
         double x = Double.parseDouble(token);
         data.add(x);
      }

      System.out.println("Number of data values: " + data.getCount( ));
      System.out.println("Average = " + data.getAverage( ));
      System.out.println("Maximum = " + data.getMaximum( ));
   }
}
```

Traversing the Characters in a String

The _____ method of the **String** class returns an individual character as a value of type _____. Recall that string positions are numbered from ____. The pattern for transversing a string is shown below.

```
for (int i = 0;  i < s.length( );  i++)
{
   char ch = s.charAt(i);
   do something with ch
}
```

Suppose you want to count the number of vowels in a string. The loop below carries out the task. Here we use the _____ method of the **String** class. The call `str.indexOf(ch);` returns the first occurrence of ch in **str** or ____ if **ch** doesn't occur in **str**.

```
int vowelCount = 0;
String vowels = "aeiouy";
for (int i = 0;  i < s.length( );  i++)
{
   char ch = Character.toLowerCase(s.charAt(i));
   if (vowels.indexOf(ch) >= 0)
      vowelCount++;
}
```

Symmetric and Asymmetric Bounds

It is easy to write a loop with i going from 1 to n: `for (i = 1;  i <= n;  i++) . . .`

The values for i are bounded by the relation _____ . Because there are _____ comparisons on both bounds, the bounds are called _____ .

When traversing the characters of a string, the bounds are _____ :

`for (i = 0;  i < s.length( );  i++) . . .` . The values of i are bounded by _____ with a $\leq$ on the left and a $<$ on the right. That is appropriate because _____ is not a valid position.

Random Numbers and Simulations

In a _____ you generate _____ events and evaluate their outcomes. The _____ class of the Java library implements a random number generator which produces numbers that appear to be completely random. To generate random numbers, you construct an object of the _____ class and then apply one of the methods shown in the chart.

Method	Returns
nextInt(n)	a random integer between the integers 0 (inclusive) and n (exclusive)
nextDouble(n)	a random floating-point number between 0 (inclusive) and n (exclusive)

For example, you can simulate the cast of a die as shown. The call `generator.nextInt(6)` gives you a random number between _____ . Add 1 to obtain a number between _____ .

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

The following is a dice program to give you a feeling of how to use random numbers.

```
public class Die
{
   public Die(int s)
   {
      sides = s;
      generator = new Random( );
   }

   // simulates the throw of a die
   public int cast( )
   {
      return 1 + generator.nextInt(sides);
   }

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

```
public class DieTest
{
   public static void main(String[ ] args)
   {
      Die d = new Die(6);

      // toss the die 10 times
      for(int i = 1;  i <= 10;  i++)
      {
         int n = d.cast( );
         System.out.print(n + " ");
      }
      System.out.println( );
   }
}
```