CS620
Structured Programming
Introduction to Java

Day 3 - Lecture 1

Selection, Looping revisited

Recap

- Just to refresh and clarify some of what we covered yesterday..

Recap - Pseudocode

- Try to get into the habit of writing psuedocode every time you get a new programming task.
- If you have a task and you simply cannot imagine where to start, instead ofworrying that you don’t know what to write; juststart writing it in pseudocode.
- You might eventually see what you need to write (in real code) when you take a look at your program in ‘pseudo-form’.

Conditionals - Recap

- If
- Then
- Else
- Switch

Conditionals in Pseudo-Code

- Simple If-Else:

```
3     if(condition)
4     {
5         perform this action;
6     }
7     else
8     {
9         perform this action if the above condition is false;
10    }
```

Conditionals in Pseudo-Code

- If-Else-If:

```
15      if(variable equals one value)
16      {
17          perform this action;
18      }
19      else if(variable equals another value)
20      {
21          perform this action;
22      }
```

Conditionals in Pseudo-Code

- If-Else-If-Else:

```
29      if(variable equals one value)
30      {
31          perform this action;
32      }
33      else if(variable equals another value)
34      {
35          perform this action;
36      }
37      else if(variable equals another value)
38      {
39          perform this action;
40      }
41      else
42      {
43          if the variable was not equal to any of the values tested in the
44              if statement and all the else if statements, then perform this action;
45      }
```

Conditionals in Pseudo-Code

- Switch:

```
50      switch(variable)
51      {
52          case possible value: perform this action;
53          break;
54
55          case possible value: perform this action;
56          break;
57
58          case possible value: perform this action;
59          break;
60
61          default: perform this action if none of the values match;
62      }
```

If-Then

- The most basic control structure is **if-then**:
  - `if (someBoolean)`
  - `{`
    - `// Do stuff`
  - `}`
  - Tells your program to execute a certain section of code *only if* a particular test evaluates to true.
  - The opening brace is the equivalent of ‘then’ when saying ***if x, then do y***

If-Then

- If *isMoving* is false, the execution jumps to the end of the ‘if’ block.

```
4   // A method somewhere in a 'Bike' class
5   void applyBrakes()
6   {
7       // the "if" clause: bicycle must be moving
8       if (isMoving)
9       {
10          // the "then" clause: decrease current speed
11          currentSpeed--;
12      }
13  }
```

- Braces can be left out **if** there’s only one statement inside the block:

```
3   void applyBrakes()
4   {
5       // same as above, but without braces
6       if (isMoving)
7           currentSpeed--;
8   }
```

- It’s good practice to use braces anyway

If-Then-Else

- In this case, the action is to simply print an error message stating that the bicycle has already stopped.

```
3   void applyBrakes()
4   {
5       if (isMoving)
6       {
7           currentSpeed--;
8       }
9       else
10      {
11          System.err.println("The bicycle has already stopped!");
12      }
13  }
```

A more complex example:

```
1  class IfElseDemo
2  {
3      public static void main(String[] args)
4      {
5          int testscore = 76;
6          char grade;
7
8          if (testscore >= 90)
9          {
10             grade = 'A';
11         }
12         else if (testscore >= 80)
13         {
14             grade = 'B';
15         }
16         else if (testscore >= 70)
17         {
18             grade = 'C';
19         }
20         else if (testscore >= 60)
21         {
22             grade = 'D';
23         }
24         else
25         {
26             grade = 'F';
27         }
28         System.out.println("Grade : " + grade);
29     }
30 }
```

Switch - Example

```
4       int month = 8;
5       String monthString;
6       switch (month)
7       {
8           case 1:  monthString = "January";
9                    break;
10          case 2:  monthString = "February";
11                   break;
12          case 3:  monthString = "March";
13                   break;
14          case 4:  monthString = "April";
15                   break;
16          case 5:  monthString = "May";
17                   break;
18          case 6:  monthString = "June";
19                   break;
20          case 7:  monthString = "July";
21                   break;
22          case 8:  monthString = "August";
23                   break;
24          case 9:  monthString = "September";
25                   break;
26          case 10: monthString = "October";
27                   break;
28          case 11: monthString = "November";
29                   break;
30          case 12: monthString = "December";
31                   break;
32          default: monthString = "Invalid month";
33                   break;
34      }
35      System.out.println(monthString);
```

While

- The while statement continually executes a block of statements while a particular condition is true.

- Its syntax can be expressed as:
  - `while (expression) { statement(s) }`

- The **while** statement evaluates *expression*, which must return a boolean value.

- If the expression evaluates to true, the **while** statement executes the *statement*(s) in the while block.

- The **while** statement continues testing the expression and executing its block until the expression evaluates to false.

While - Infinite Loop

- The following code will cause an *infinite loop*; otherwise known as a **crash.**
  - `while(true) { doSomething(); }`
- The expression ‘true’ will never evaluate as ‘false’ under any circumstances, so the loop never stops.
- Because a program executes its instructions in sequence and only ever does one thing at a time, it gets *‘stuck’* inside the infinite loop and appears to freeze up!

While - Example

- The following program will print 10 messages to the screen by executing the same statements 10 times over

- The repeated statements are the ones inside the **while** loop's block.

```
1   class WhileDemo
2   {
3       public static void main(String[] args)
4       {
5           int count = 1;
6           while (count <= 10)
7           {
8               System.out.println("Count is: " + count);
9               count++;
10          }
11      }
12  }
```

Do-While

```
1   class DoWhileDemo
2   {
3       public static void main(String[] args)
4       {
5           int count = 1;
6           do
7           {
8               System.out.println("Count is: " + count);
9               count++;
10          } while (count < 11);
11      }
12  }
```

For

- **For** provides a compact way to iterate over a range of values.
- Programmers often refer to it as the *"for loop"* because of the way in which it repeatedly loops until a particular condition is satisfied.
- It can do the same things as a while loop, but in a more compact manner.

For

- A typical **for** loop:

```
1  class ForDemo
2  {
3      public static void main(String[] args)
4      {
5          for(int i=1; i<11; i++)
6          {
7              System.out.println("Count is: " + i);
8          }
9      }
10 }
```

- The above example does exactly the same as our first **while** loop:

```
1  class WhileDemo
2  {
3      public static void main(String[] args)
4      {
5          int count = 1;
6          while (count <= 10)
7          {
8              System.out.println("Count is: " + count);
9              count++;
10         }
11     }
12 }
```

Loops in Reverse

- Loops can work ‘backwards’ too
- In fact, you simply need to set them up so that they check a condition that will eventually reach ‘false’ for the loop to not be infinite.
- ‘Reverse’ **for** loop:

```
1  class ForDemo_Reverse
2  {
3      public static void main(String[] args)
4      {
5          for(int i=10; i>=0; i--)
6          {
7              System.out.println("Count is: " + i);
8          }
9      }
10 }
```

Nested Loops

- Two for loops, one nested inside another:

```
1  class NestedFor
2  {
3      public static void main(String[] args)
4      {
5          System.out.println("");
6          System.out.println("i j");
7          System.out.println("---");
8          for(int i = 1; i <= 5; i++)
9          {
10             for(int j = 1; j <= 5; j++)
11             {
12                 System.out.println(i + " " + j);
13             }
14             System.out.println("Finished " + i + " iterations of outer loop");
15         }
16     }
17 }
```

Loops in Reverse

- 'Reverse' **while** loop:

```
1   class WhileDemo_Reverse
2   {
3       public static void main(String[] args)
4       {
5           int count = 10;
6           while (count >= 1)
7           {
8               System.out.println("Count is: " + count);
9               count--;
10          }
11      }
12  }
```

Break

- We saw **break** used before with switch
- It can also be used in loops to exit the loop ‘early’

```
3   for (int i = 0; i <= 10; i++)
4   {
5       if (i == 5)
6       {
7           System.out.println("Reached i==5!");
8           break;
9       }
10  }
```

- This loop is set up to run 11 times, but will exit after 5 iterations because of **break;**