3.3: Flow Controls

IT1406 - Introduction to Programming

Level I - Semester 1

3.3.1 Selection statements

Flow Control

- Control flow refers to the order in which statements are executed in an algorithm.

- Usually, an algorithm executes sequentially; that is the first statement executes, then the second and so on. However, it is often useful to be able to alter this flow.

- A flow control statement is a statement that changes the order of execution of subsequent statements.

- We will be talking about 3 types of flow control statements
  - Selection Statements
  - Iterative Statements
  - Jump Statements

Selection Statements

- A Selection Statement, evaluates some expression, and depending on the expression’s value, selects one of several possible sets of statements to execute.

- In Java, we have two types of selection statements:
  - **If**
  - **switch**

Selection Statements
If Statements

- An If Statement has the form:

```
if (<Boolean expression>)
{
  <statement1.1>;
  <statement1.2>;
  …
}
else
{
  <statement2.1>;
  <statement2.2>;
  …
}
```

- If <Boolean expression> is true, then <statement1.1>, <statement1.2> etc. are executed; else <statement2.1>, <statement2.2> etc. are executed. The "else" part is optional if there is nothing to be executed if <Boolean expression> is false.

Selection Statements
If Statements (cont…)

- If only one statement needs to be executed, the curly brackets can be omitted.

```
if (<Boolean expression>)
  <statement1.1>;
else
  <statement2.1>;
```

Selection Statements
If Statements (cont…)

- If statements can be nested inside other if statements to give multiple if statements.

```
if (<Boolean expression1>)
  <statement1.1>;
else if (<Boolean expression2>)
  <statement2.1>;
else if (<Boolean expression3>)
  <statement3.1>;
else
  <statement4.1>;
```

Selection Statements
Switch Statements

- An Switch Statement has the form:

```
switch (<Expression>)
{
  case <value 1>:
      <statement1.1>;
      <statement1.2>;
  case <value 2>:
      <statement2.1>;
      <statement2.2>;
  case <value 3>:
      <statement3.1>;
      <statement3.2>;
  case <value 4>:
      <statement4.1>;
      <statement4.2>;
      …
  default:
      <statementD.1>;
      <statementD.2>;
}
```

- The expression <Expression> must evaluate to a byte, int or a char.

Selection Statements
Switch Statements (cont…)

- <Expression> is compared with each of the case values <value 1>, <value 2>, <value 3> etc.

- If a match is found, the statements that correspond to the first match, and all the other case statements further down are executed
  - For example if <Expression> is equal to <value 2>, <statement2.1>, <statement2.2> etc. are executed; then, <statement3.1>, <statement3.2> etc. are executed, and so on.

- If a match is not found, default statements <statementD.1>, <statementD.2> etc. are executed.

- The default statements are optional. If it is not there, case statement completes without doing anything.

Selection Statements

```
// Demonstrate if-else-if statements.
class IfElse {
 public static void main(String args[]) {
  int month = 4; // April
  String season;

  if(month == 12 || month == 1 || month == 2)
   season = "Winter";
  else if(month == 3 || month == 4 || month == 5)
   season = "Spring";
  else if(month == 6 || month == 7 || month == 8)
   season = "Summer";
  else if(month == 9 || month == 10 || month == 11)
   season = "Autumn";
  else
   season = "Bogus Month";

  System.out.println("April is in the " + season + ".");
 }
}
```

Selection Statements

```
// A simple example of the switch.
class SampleSwitch {
  public static void main(String args[]) {
    for(int i=0; i<6; i++)
      switch(i) {
        case 0:
          System.out.println("i is zero.");
          break;
        case 1:
          System.out.println("i is one.");
          break;

        case 2:
          System.out.println("i is two.");
          break;
        case 3:
          System.out.println("i is three.");
          break;
        default:
          System.out.println("i is greater than 3.");
      }
  }
}
```

Selection Statements

- The break statement is optional. If you omit the break, execution will continue on into the next case. It is sometimes desirable to have multiple cases without break statements between them. For example, consider the following program:

Selection Statements

```
// In a switch, break statements are optional.
class MissingBreak {
  public static void main(String args[]) {
    for(int i=0; i<12; i++)
      switch(i) {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
          System.out.println("i is less than 5");
          break;
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:
          System.out.println("i is less than 10");
          break;
       default:
          System.out.println("i is 10 or more");    }  }
}
```

Selection Statements

After JDK 7, you can use a string to control a switch statement. For example,

```
// Use a string to control a switch statement.
class StringSwitch {
  public static void main(String args[]) {
    String str = "two";
    switch(str) {
      case "one":
        System.out.println("one");
        break;
      case "two":
        System.out.println("two");
        break;      case "three":
        System.out.println("three");        break;      default:
System.out.println("no match");        break;    }  }
```

3.3.2 Iterative statements

- Iterative statements execute a set of statements over and over again, until some condition is satisfied.

- Java has 3 types of Iterative Statements:
  - For loops
  - While loops
  - Do while loops

Iterative Statements
For loops

- A For loop has the form:

```
for (<Initialization statement>; <Boolean Expression>;
  <Increment statement>)
{
  <statement1.1>;
  <statement1.2>;
  ...
}
```

- First **<Initialization statement>** is executed.
- Next **<Boolean Expression>** is evaluated.
  - If **<Boolean Expression>** is true, **<statement1.1>**, **<statement1.2>** etc. and **<Increment statement>** are executed. The **<Boolean Expression>** is evaluated again, and the loop continues.
  - If **<Boolean Expression>** is false, executing the iterative statement is concluded.

Iterative Statements
For loops (cont…)

- For example, the following loop finds the sum of the numbers 0, 1, 2, …99.

```
int sum = 0;
for (int i = 0; i<100; i=i+1)
{
  sum = sum +i;
}
```

Iterative Statements
While loops (cont...)

- A While loop has the form:

```
while (<Boolean Expression>)
{
  <statement1.1>;
  <statement1.2>;
  ...
}
```

- First **<Boolean Expression>** is evaluated.
  - If **<Boolean Expression>** is true, **<statement1.1>, <statement1.2>** etc. are executed. The **<Boolean Expression>** is evaluated again, and the loop continues.
  - If **<Boolean Expression>** is false, executing the iterative statement is concluded.

Iterative Statements
While loops (cont...)

- For example, the following loop finds the sum of the numbers 0, 1, 2, ...99.

```
int sum = 0;
int i=0;
while (i<100)
{
 sum = sum +i;
 i = i+1;
}
```

Iterative Statements
Do While loops

- A Do While loop has the form:

```
do
{
  <statement1.1>;
  <statement1.2>;
  …
}
while (<Boolean Expression>);
```

- First **<statement1.1>, <statement1.2>** etc. are executed.
- Next, the **<Boolean Expression>** is evaluated.
  - If **<Boolean Expression>** is true, **<statement1.1>, <statement1.2>** etc. are executed. The **<Boolean Expression>** is evaluated again, and the loop continues.
  - If **<Boolean Expression>** is false, executing the iterative statement is concluded.

Iterative Statements
Do While loops (cont...)

- For example, the following loop finds the sum of the numbers 0, 1, 2, ...99.

```
int sum = 0;
int i=0;
do
{
  sum = sum +i;
  i = i+1;
}
while (i<100);
```

Applying the Enhanced for

- Since the for-each style **for** can only cycle through an array sequentially, from start to finish, you might think that its use is limited, but this is not true. A large number of algorithms require exactly this mechanism. One of the most common is searching. For example, the following program uses a **for** loop to search an unsorted array for a value. It stops if the value is found.

Applying the Enhanced for

```
// Search an array using for-each style for.
class Search {
  public static void main(String args[]) {
    int nums[] = { 6, 8, 3, 7, 5, 6, 1, 4 };
    int val = 5;
    boolean found = false;

    // use for-each style for to search nums for val
    for(int x : nums) {
      if(x == val) {
        found = true;
        break;
      }
    }

    if(found)
      System.out.println("Value found!");
  }
}
```

Lesson 3.3.4:
Jump statements

- Jump statements move the point of execution in a program from one place to another place.

- In java we have 3 types of Jump Statements.
  - **Break Statement**
  - **Continue Statement**
  - **Return Statement**

Jump Statements
Break Statement

- A Break Statement jumps out of a loop and effectively bypasses the loop condition.

- For example, the following loop finds the sum of the numbers 0, 1, 2, ...55.

```
int sum = 0;
int i=0;
while (i<100)
{
 sum = sum +i;
 i = i+1;
 if (i==55)
   break;
}
```

Jump Statements
Continue Statement

- A Continue Statement jumps out of the current iteration of a loop.
- For example, the following loop finds the sum of the numbers 0, 1, 2, …99 excluding the numbers that are multiples of 5.

```
int sum = 0;
for (int i = 0; i<100; i=i+1)
{
 if (i%5==0)
  continue;
 sum = sum +i;
}
```

Jump Statements
Return Statement

- A Return Statement terminates the current method (function) and jumps to the place immediately after the function call.

Jump Statements
Labeled loops

- With a label, program controls transfer to the statement just after the enclosing statement or block of statements carrying the same label.

Jump Statements
Labeled loops (cont…)

```
for (int i=0; i<3; i++)
{
    resume:
    for (...............)
      {
         ........
         if (........) break resume;
      }
  }
```