Basics of Java

Lecture 3

Java vs other programming languages
• Basic syntax very similar to C, C++
• Fully object-oriented: all code and data is within a class
• Java handles memory management: no need to allocate or free memory!
• No pointers, no segmentation faults!!
• Easy to learn and use
• Execution time slightly slower than C or C++

Programming cycle in Java

Integrated Development Environment
• IDE: Program that facilitates writing code, compiling it, running it, and debugging it.
• Recommended IDE: Eclipse
  – You also need to install a Java Runtime Environment (JRE), from the same URL.
  – Runs on all OS: Windows, Mac, Linux, etc.
  – Installed on all machines in Trottier

My first Java program

Variables and types

```java
// Variables and types

public class VariablesExample {
    public static void main(String args[]) {
        int age;  // age can store an integer
        float pi;  // float can store a decimal number
        age = 29;
        pi = 3.14;
    }
}
```

```java
/* This program prints a welcoming statement */
public class Welcome {
    // Every executable class have to contain a method called main like below
    // When the class is executed, main is the first method to be called
    public static void main(String args[]) {
        System.out.println("Welcome to Java!");
        System.out.println("This is easy!");
    }
}
```

File: Welcome.java

1) Compile Welcome.java
2) Run the program
Output: Welcome to Java!
This is easy!

Notes on java byte-code:
Advantages: - Byte-code is platform independent
            - Very important for internet!
Disadvantage: - Byte-code has to be interpreted by the JVM so it runs slightly slower
### Primitives

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8-bit</td>
<td>signed integer</td>
<td>[-128,127]</td>
</tr>
<tr>
<td>char</td>
<td>16-bit</td>
<td>integer</td>
<td>[0, 65536] (encodes 'a', 'b' ...)</td>
</tr>
<tr>
<td>short</td>
<td>16-bit</td>
<td>signed integer</td>
<td>[-32768,32767]</td>
</tr>
<tr>
<td>int</td>
<td>32-bit</td>
<td>signed integer</td>
<td>[-2147483648, 2147383647]</td>
</tr>
<tr>
<td>long</td>
<td>64-bit</td>
<td>signed integer</td>
<td>[-9223372036854775807, 9223372036854775806]</td>
</tr>
<tr>
<td>float</td>
<td>32-bit</td>
<td>decimal number</td>
<td>1.40239e-45 to 3.402823e+38</td>
</tr>
<tr>
<td>double</td>
<td>64-bit</td>
<td>decimal number</td>
<td>4.9406e-324 to 1.79769e+308</td>
</tr>
<tr>
<td>boolean</td>
<td>8-bit</td>
<td>boolean</td>
<td>true or false</td>
</tr>
</tbody>
</table>

### Expressions and Assignments

- **Expression**: Piece of code that has a value of a certain type
- **Assignment**: Storing the value of an expression into a variable
- **Syntax**: `<variable> = <expression>`

### Examples:

```java
public class Expressions {
    public static void main(String args[]) {
        int i, j, k;
        float f, g;
        i = 5;
        j = i + 1;
        k = k * 2;  // Compiling error: Why? _________________________________
        j = j / 2;
        j + 10;       // Legal, but useless. Why? _________________________________
        g = j + 3.14; // Note the implicit conversion of j into a float
        f = 15 / 2;    // f now has value 7.0. Why?__________________________________
        f = ((float) i) / 2 // Explicit type conversion (casting) of i into a float. f is now 7.5
    }
}
```

### Memory (RAM)

- **Boolean expressions**
  - Boolean expressions have value true or false.
  - Operations on booleans:
    - `NOT` : `!`
    - `AND` : `&&`
    - `OR` : `||`

### Exercise

```java
public class Exercise {
    public static void main(String args[]) {
        int i, j, f;
        boolean a, b, c;
        char c = 'f';
        f = i;      // compilation error: i is not initialized
        i = 9;
        a = (f > 100);    // compilation error: f is not initialized
        f = i;
        b = true;
        a = (b || (12345.67*i - f/0.02345 == 0.003464)); // a is __________
        j = i;
        j = j + 1;     // value of j: 10,   value of i is still 9
        i = f + 3.3;  // error: a float value cannot be stored in an int
        i = (int) (f + 3.3);    // the float value 12.3 is cast into an int.
        // It becomes 12, so i becomes 12
        b = b && ( (i == j) || (!b || f > 10) );
        System.out.println("The point is inside the circle");
        if ( (x==a) && (y==b) ) System.out.println("It is the center");
        else {
            System.out.println("The point is outside the circle");
            // other statements could be here
        }
    }
}
```

### Conditionals

- **Syntax**: `if (<boolean expression>) <statementBlock1> [else <statementBlock2>]`
- **Executes** `<statementBlock1>` only if `<boolean expression>` is true. Otherwise `<statementBlock2>` is executed.

### while loops

- **Syntax**: `while (<boolean expression>) <statementBlock>`
- **Keeps executing** `<statementBlock>` repeatedly as long as `<boolean expression>` is true.
- **Example**: If `<boolean expression>` is false from the beginning, then `<statement>` is never executed.
do-while loops

• do <statementBlock> while (<boolean expression>)
• Same as while-loop but <boolean condition> is checked after executing <statementBlock>, so <statement> is always executed at least once.

```java
// Keep asking for a price as long as the number entered is not positive
double price = 0;
String line;     // String is a special type of variable. More about strings next week
do {
    System.out.println("Enter price of item:");
    line = stdin.readLine();     // Read a line from keyboard
    price = Double.parseDouble(line);    // Parse the line to get a double
} while (price<=0);
```

For loops

• for (<statement1>; <boolean expression>; <statement2>)
• <statementBlock3>
• Equivalent to:
  <statement1>
  while (<boolean expression>) {
   <statementBlock3>
  <statement2>
}

```java
int n=5;
int s=0;
int i;
for (i = 0; i < n; i++ ) {
s = s + i;
}
System.out.println("Value of s: "+ s);
```

Arrays

• Arrays are used to store and manipulate several variables of the same type

```java
int n=5;
for(int i = 1; i <= n; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print( i );
    }
    System.out.println(" ");    // print the end of line
}
```

Arrays in Java

• Declaration:
  ```java
  double[] a;   // a will be array of double.
  int[] x,y;    // x and y will both be arrays of ints
  // IMPORTANT: At this point, a, x, and y are references to null arrays
  x[ a[ 1 ] ] = 3; // compiling error. Why? _______________________
  a[ 10 ] = 3; // run-time error. ArrayIndexOutOfBoundsException gets thrown
  ```

• Assignments with arrays
  ```java
  y=x;
  y[ 3 ]=9;    // the value of x[3] is now __________
  y[ 2 ]=4;    // the value of y[2] is now _________
  a = x;      // compl. error: a and x have different types
  x = new int[ 0 ];
  x[3] = 5;    // the value of y[3] is now ________
  y = new int[3];
  ```

Memory (stack) | Memory (heap)
null | null

Arrays in Java

• Accessing elements
  ```java
  a[ 3 ] = 3.1;
  a[ 0 ] = 1;
  a[ a[ 0 ] + 1 ] = 1.5;
  if ( 3 > 2 ) { // run-time error. ArrayIndexOutOfBoundsException gets thrown
      a[ 0 ] = 3;      // run-time error. ArrayIndexOutOfBoundsException gets thrown
      a[ 10 ] = 3;    // run-time error. ArrayIndexOutOfBoundsException gets thrown
  }
  ```

```java
Arrays
• Arrays are used to store and manipulate several variables of the same type
  ```java
  int[] X;
  ```
  To access the i-th element:
  ```java
  int X[i];
  ```
  Note: first element is at index 0.
  ```java
  X.length
  ```
  is the number of elements in X (here, it’s 13)
  ```java
  Java makes sure you don’t write outside arrays:
  // ArrayIndexOutOfBoundsException gets thrown if you try
  ```
Multi-dimensional arrays

• Arrays can have more than one dimension:
  
  ```java
  double matrix[][] = new double[10][10];
  // initialize the matrix to zero
  for (int i=0; i<10; i++) {
      for (int j=0; j<10; j++) {
          matrix[i][j] = 0;
      }
  }
  // make it an identity matrix
  for (int i=0; i<10; i++) matrix[i][i]=1;
  ```