Object-Oriented Programming

Java Programming Essentials Cont.

Computer Science and Technology
United International College

Outline

- **Java Packages**
- **Flow Control**
  - **Conditionals** (if-then-else)
  - **Switch Statements** (switch)
  - **Iteration Statements** (while, do-while)
  - **For Loops** (for)
  - **Transfer Statement** (continue)
  - **Break Statement** (break)

Importing Packages and Classes

- Code libraries in Java are called ***packages.***
  - A package is a **collection of classes** that is stored in a manner that makes it easily accessible to any program.
  - In order to use a class that belongs to a package, the class must be brought into a program using an **`import`** statement.
  - Classes found in the package **`java.lang`** are imported automatically into every Java program.

```
import java.text.NumberFormat;
  // import theNumberFormat class only
import java.text.*;
  //import all the classes in package java.text
```

- Let's check Java's Math API!

Example: java.lang.Math

```
// java.lang.Math is imported automatically.
public class Test {
   public static void main(String[] args) {
      double i = -3.24;
      double j = 56.2;
      System.out.println(Math.abs(i));
      System.out.println(Math.max(i, j));
      System.out.println(Math.sqrt(j));
   }
}
```

Flow of Control

- **`if-else`**, **`if`**, and **`switch`** statements.
- **`while`**, **`do-while`**, and **`for`** statements.
- A Boolean expression evaluates to either **`true`** or **`false`** -- **used to control the flow**

`if-else` Statement

- An **`if-else`** statement chooses between two alternative statements based on the value of a Boolean expression:

```
if(Boolean_Expression)
   Yes_Statement
else
   No_Statement
```

- Each ***`Yes_Statement`*** and ***`No_Statement`*** branch of an **`if-else`** can be a made up of a single statement or a **compound statement.**

Compound Statements

Compound Statement: a statement that is made up of a list of statements and enclosed in a pair of braces ({ }).

```
if(myScore > yourScore) {
    System.out.println("I win!");
    wager = wager + 100;
} else {
    System.out.println("So sad...");
    wager = 0;
}
```

Multiway `if-else` Statement

```
if(Boolean_Expression)
    Statement_1
else if(Boolean_Expression)
    Statement_2
        .
        .
        .
else if(Boolean_Expression_n)
    Statement_n
else
  Statement_For_All_Other_Possibilities
```

Example: if-else

```
public class Test {
    public static void main(String[] args) {
        int i = 20;
        if(i < 20) {
                System.out.println("<20");
        } else if(i < 40) {
                System.out.println("<40");
        } else if(i < 60) {
                System.out.println("<60");
        } else {
                System.out.println(">=60");
        }
    }
}
```

The `switch` Statement

- The **`switch`** statement is the only other kind of Java statement that implements **multiway** branching.
  - When a **`switch`** statement is evaluated, one of a number of different branches is executed.
  - The choice of which branch to execute is determined by a **controlling expression** enclosed in parentheses after the keyword **`switch`** .
    - The controlling expression must evaluate to a **`char`**, **`int`**, **`short`**, or **`byte`**.
  - Syntax is similar to C.

The `switch` Statement

```
switch(numberOfFlavors) {
  case 32:
       System.out.println("So many yummy flavors!! ");
       break;
  case 1:
       System.out.println("I bet it is vanilla");
       break;
  case 2:
  case 3:
  case 4:
       System.out.println(numberOfFlavors + " is ok");
       break;
  default:
       System.out.println("How many do you have?");
       break;
}
```

Example: switch

```
public class Test {
    public static void main(String[] args) {
        int i = 3;
        switch(i) {
            case 8:
            case 3:
            case 2:
                System.out.println("C");
                break;
            case 9:
                System.out.println("D");
                break;
            default:
                System.out.println("error");
                break;
        }
    }
}
```

Boolean Expressions

- A **`boolean`** expression is an expression that is either **`true`** or **`false`**.
- The simplest **`boolean`** expressions compare the value of two expressions:

```
time < limit
yourScore == myScore
```

  - Note that Java uses two equal signs (**`==`**) to perform equality testing: a single equal sign (**`=`**) is used only for assignment.
  - Use parentheses in practice, although a **`boolean`** expression does not have to be enclosed in parentheses.

Java Comparison Operators

Display 3.3 Java Comparison Operators

MATH NOTATION	NAME	JAVA NOTATION	JAVA EXAMPLES
=	Equal to	==	`x + 7 == 2*y` `answer == 'y'`
≠	Not equal to	!=	`score != 0` `answer != 'y'`
>	Greater than	>	`time > limit`
≥	Greater than or equal to	>=	`age >= 21`
<	Less than	<	`pressure < max`
≤	Less than or equal to	<=	`time <= limit`

Pitfall: Using == with Float point

- When comparing the equality of two float point numbers, we usually do not use == for this purpose.
- We usually compare their difference to a very small number.

```
float a = 2.345678f;
float b = 2.345678f;
float c = b - a;
if(c < 1e-6) {
    System.out.println("a equals to b");
}
```

Example: ==

```
public class Test {
    public static void main(String[] args) {
        float a = 3.141234567f;
        double b = 3.141234567;
        //float b = 3.141234567f;
        if(a == b) {
            System.out.println("a is equal to b");
        } else {
            System.out.println("a is not equal to b");
        }
        if((a - b) < 1e-4) {
            System.out.println("Equal!");
        }
    }
}
```

Building Boolean Expressions

- Conjuction: **p and q** Java we write: `p && q`
- Disjunction: **p or q** Java we write: `p || q`
- Negation: **not p** Java we write: `!p`

- Mulitple inequalities must be joined by `&&`
  - Use `(min < result) && (result < max)` to represent `min < result < max`

- Boolean expressions evaluate to `true` or `false`

```
boolean madeIt = (time < limit) && (limit < max);
```

Loops

- **Loops** in Java are similar to those in other high-level languages (like C).
- Java has three types of loop statements: the **`while`**, the **`do-while`**, and the **`for`** statements.
  - The code that is repeated in a loop is called the **body** of the loop.
  - Each repetition of the loop body is called an **iteration** of the loop.

`while` statement

- A **`while`** statement is used to repeat a portion of code (i.e., the loop body) based on the evaluation of a boolean expression.
- The boolean expression is checked **before** the loop body is executed.

`while` Syntax

```
while(Boolean_Expression)
    Statement
```

Or

```
while(Boolean_Expression)  {
    Statement_1
    Statement_2

    Statement_Last
}
```

do-while Statement

- A **do-while** statement is similar to a while statement, with the exception that the boolean expression is checked **after** the loop body is executed.

```
do
    Statement
while(Boolean_Expression);
```

Or

```
do {
    Statement_1
    Statement_2
        :

    Statement_Last
} while(Boolean_Expression);
```

Example: while, do-while

```
public class Test {
	public static void main(String[] args) {
		int i = 0;
		while(i < 10) {
			System.out.println(i);
			i++;
		}
		i = 0;
		do {
			i++;
			System.out.println(i);
		} while(i < 10);
	}
}
```

The `for` Statement

- The **`for`** statement is most commonly used to step through an integer variable in equal increments.
- It begins with the keyword **`for`**, followed by three expressions in parentheses that describe what to do with one or more **controlling variables**.
  - The first expression tells how the control variable or variables are **initialized** before the first iteration.
  - The second expression determines **when the loop should continue**, based on the evaluation of a boolean expression **before** each iteration.
  - The third expression tells how the control variable or **variables** are **updated** after each iteration of the loop body.

`for` Statement Syntax

Display 3.10 **for Statement Syntax and Alternate Semantics *(Part 1 of 2)***

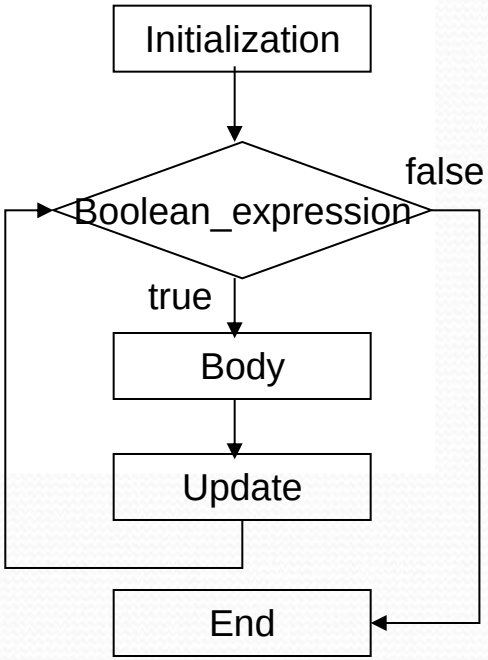
for STATEMENT SYNTAX:

SYNTAX:

```
for (Initialization; Boolean_Expression; Update)
    Body
```

EXAMPLE:

```
for (number = 100; number >= 0; number--)
    System.out.println(number
        + " bottles of beer on the shelf.");
```



TestFor.java

Example: for loop

```
public class Test {
	public static void main(String[] args) {
		long result = 0;
		long f = 1;
		for(int i = 1; i <= 10; i++) {
			f = f * i;
			result += f;
		}
		// result: 1! + 2! + ... + 10!
		System.out.println("result=" + result);
	}
}
```

Write and Show Time

Write a while loop equivalent to this for loop:

```
for(int number = 100; number >= 1; number--) {
   System.out.println(number + " bottles of beer left!");
}
```

Infinite Loops

- A **`while`**, **`do-while`**, or **`for`** loop should be designed so that the value tested in the boolean expression is changed in a way that eventually makes it false, and terminates the loop.
- If the boolean expression remains true, then the loop will run forever, resulting in an **infinite loop**.

Note: loops that check for numerical equality or inequality (**`==`** or **`!=`**) are especially prone to this error and should be avoided if possible (floating point equality is difficult to determine!)

The `break` and `continue` Statements

- The **`break`** statement: **`break;`**
  - When executed, the **`break`** statement ends the nearest enclosing switch or loop statement.
- The **`continue`** statement: **`continue;`**
  - When executed, the **`continue`** statement ends the current loop body iteration of the nearest enclosing loop statement.
  - Note that in a **`for`** loop, the **`continue`** statement transfers control to the **update** expression.

The **break** and **continue** Statements

Example:

```
for(int i = 0; i < 100; i++) {
    if(i == 47)
        break; // Exit out of for loop
    if(i % 9 != 0)
        continue; // Go to next iteration immediately
    System.out.println(i);
}
int j = 0; // An "infinite loop":
while(true) {
    j++;
    if(j == 47)
        break; // Exit out of loop
    if(j % 10 != 0)
        continue; // Go to top of loop immediately
    System.out.println(j);
}
```

Example: break

```
public class Test {
   public static void main(String[] args) {
      int stop = 4;
      for(int i = 1; i <= 6; i++) {
         if(i == stop)
            break;
         System.out.println("i = " + i);
      }
   }
}
```

Example: continue

```
public class Test {
    public static void main(String[] args) {
        int skip = 4;
        for(int i = 1; i <= 6; i++) {
            if(i == skip)
                continue;
            System.out.println("i = " + i);
        }
    }
}
```

General Debugging Techniques

- Examine the system as a whole – don't assume the bug occurs in one particular place.
- Try different test cases and check the input values.
- Comment out blocks of code to narrow down the offending code.
- Check common pitfalls.
- Take a break and come back later.
- DO NOT make random changes just hoping that the change will fix the problem!

Tips for Productive Coding

- **Plan Globally**
  - Design the overall function of the program first.
- **Develop Incrementally**
  - Write a little bit of code at a time and test it before moving on.

Write and Show Time - MultiTable

```
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
1*4=4   2*4=8   3*4=12  4*4=16
1*5=5   2*5=10  3*5=15  4*5=20  5*5=25
1*6=6   2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7   2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8   2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9   2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

Write and Show Time

```
public class Test {
    public static void main(String[] args) {
        for(int i = 1; i < 10; i++) {      //loop row
            for(int j = 1; j <= i; j++) { //loop column
                System.out.print(j + "*" + i + "=" + i*j);
                if(i * j < 10) // only one digit of output
                    System.out.print("   ");
                else
                    System.out.print("  ");
            }
            System.out.println();
        }
    }
}
```

Summary

- Java Packages
- Flow Control
  - Conditionals (if-then-else)
  - Switch Case Statement (switch)
  - Iteration Statement (while, do-while)
  - For Loops (for)
  - Transfer Statement (continue)
  - Break Statement (break)