Java Selection Statement

Prepared by

Dr. Rajesh Kumar Ojha
Asst. Prof., CSE, Silicon University

Control Flow Statement?

- *Control flow statements*, break up the flow of execution by employing decision making, looping, and branching, enabling your program to *conditionally* execute particular blocks of code.

if-then Statement

- The if-then statement is the most basic of all the control flow statements.
- It tells your program to execute a certain section of code *only if* a particular test evaluates to true.

Example

- For example, the Bicycle class could allow the brakes to decrease the bicycle's speed *only if* the bicycle is already in motion. One possible implementation of the applyBrakes method could be as follows:

```
void applyBrakes() {
// the "if" clause: bicycle must be moving
 if (isMoving){
// the "then" clause: decrease current speed currentSpeed--;
 }
}
```

if-then-else Statement

- The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false.
- You could use an if-then-else statement in the applyBrakes method to take some action if the brakes are applied when the bicycle is not in motion. In this case, the action is to simply print an error message stating that the bicycle has already stopped.

Example

```
void applyBrakes() {
if (isMoving) {
currentSpeed--;
} else {
System.err.println("The bicycle has already stopped!");
 }
 }
```

Example

```
class IfElseDemo {
public static void main(String[] args) {
int testscore = 76;
char grade;
if (testscore >= 90) {
grade = 'A';
} else
if (testscore >= 80) {
 grade = 'B'; } else
if (testscore >= 70) {
grade = 'C'; } else
if (testscore >= 60) {
 grade = 'D'; } else
{ grade = 'F'; }
System.out.println("Grade = " + grade);
}}
```

Switch?

- The switch statement can have a number of possible execution paths.
- A switch works with the byte, short, char, andint primitive data types. It also works with *enumerated types* (discussed in Enum Types), the String class, and a few special classes that wrap certain primitive types:Character, Byte, Short, and Integer

Example

```
public class SwitchDemo {
public static void main(String[] args) {
 int month = 8;
String monthString;
 switch (month) {
case 1: monthString = "January";
break;
case 2: monthString = "February";
break;
case 3: monthString = "March"; break;
case 4: monthString = "April"; break;
case 5: monthString = "May"; break;
case 6: monthString = "June"; break;
case 7: monthString = "July"; break;
case 8: monthString = "August";
break;
case 9: monthString = "September";
break;
case 10: monthString = "October";
break;
case 11: monthString = "November";
break;
case 12: monthString = "December";
break;
default: monthString = "Invalid
month"; break;
}
System.out.println(monthString);
}
}
```

while and do-while Statements

- 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.

Example

```
class WhileDemo {
public static void main(String[] args){
int count = 1;
while (count < 11) {
 System.out.println("Count is: " + count); count++;
}
}
 }
```

Do-while Statement

- The Java programming language also provides a do-while statement, which can be expressed as follows:

```
do {
statement(s)
} while (expression);
```

- The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within thedo block are always executed at least once

Example

```
class DoWhileDemo {
public static void main(String[] args){
 int count = 1;
do {
System.out.println("Count is: " + count);
count++;
} while (count < 11);
}
 }
```

for Statement

- The for statement provides a compact way to iterate over a range of values.
- The general form of the for statement can be expressed as follows:

for (*initialization*; *termination*; *increment*) {

statement(s)

}

- The *initialization* expression initializes the loop; it's executed once, as the loop begins.
- When the *termination* expression evaluates to false, the loop terminates.
- The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment *or* decrement a value.

Example

```
class ForDemo {
public static void main(String[] args){
 for(int i=1; i<11; i++){
System.out.println("Count is: " + i);
}
}
}
```

foreach Statement

- The for statement also has another form designed for iteration through Collections and arrays.
- This form is sometimes referred to as the *enhanced for* statement, and can be used to make your loops more compact and easy to read

Example of foreach

```
class Test {
 public static void main(String[] args) {
int[] arr={10,20,30,40};
 for(int x:arr) {
System.out.println(x);
 }
}
}
```

output: 10
20
30
40

Example of foreach

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

Thank you