Programming 1

Week 05 – Decision Structures

Introduction

- So far, we have learned the `if if…else if…else..if` structure, with this structure, even if you have more than two situations, you can still solve the problem.
- We already know that after the “if” and “else if” there will be a condition, and that condition could be anything, you can use `>, <, ==, !=, &&, || or !.`

```
if (age < 18)
        price = 0;
else if (age >= 65)
        price = 5;
else
        price = 10;
```

```
if (age >= 18 && nationality == “Canada”)
          canVote = true;
else
          canVote = false;
```

Introduction

- Because we can put as many branches as we want, and we can use whatever Boolean operators to write the condition, the “`if…else if…else`” statement is the most general structure of the selection control structure.

- There is another selection control structure, known as the "`switch case`" structure. It's important to note that the "switch case" structure has different capabilities compared to the "`if...else if...else`" structure, and it is typically used for specific situations.

The `switch` Statement

- The "`switch case`" structure may have many branches but can only use "==" in the condition. In this case, the usage of it is much narrower than the "`if…else if…else`" structure.

- Usually switch-case structure is used when executing some operations based on a state variable. There an int has more than enough options. Boolean has only two so a normal if is usually good enough. Doubles and floats aren't really that accurate to be used in this fashion.

The `switch` Statement (1 of 4)

- The `if-else` statement allows you to make true / false branches.
- The `switch` statement allows you to use an ordinal value to determine how a program will branch.
- The `switch` statement can evaluate a variable or an expression that gives a `char`, `byte`, `short`, `int`, or string value, and make decisions based on the value.
- Why use `switch` statements?
  - They are convenient when several alternative courses of action depend on a single integer, character, or string value
  - Use only when there is a reasonable number of specific matching values to be tested

The `switch` Statement (2 of 4)

- The `switch` statement takes the form:

```
switch (TestExpression)
{
  case CaseExpression:
    // place one or more statements here
    break;
  case CaseExpression:
    // place one or more statements here
    break;

    // case statements may be repeated
    //as many times as necessary
  default:
    // place one or more statements here
}
```

The `switch` Statement (3 of 4)

```
switch (TestExpression)
{
    …
}
```

- The `switch` statement will evaluate the **TestExpression**, which can be a `char`, `byte`, `short`, `int`, or string.
- If there is an associated `case` statement that matches that value, program execution will be transferred to that `case` statement.

The `switch` Statement (4 of 4)

- Each `case` statement will have a corresponding **CaseExpression** that must be unique.

```
case CaseExpression:
    // place one or more statements here
    break;
```

- If the **TestExpression** matches the **CaseExpression**, the Java statements between the colon and the `break` statement will be executed.

The `case` Statement

- The `break` statement ends the `case` statement.
- The `break` statement is optional.
- If a `case` does not contain a `break`, then program execution continues into the next `case`.
  - If a `break` statement is omitted:
    - The program finds a match for the test variable
    - All statements within the `switch` statement execute from that point forward
- The `default` section is optional and will be executed if no **CaseExpression** matches the **TestExpression**.

Example

- Consider a simplified letter grade that is given for a course project (i.e., A, B, C, D, F). Sometimes when a student receives a letter grade, he/she would like to know what percentage range corresponds to that letter. Consider code that uses if statements to compute the proper range as follows:

```
char        aLetter;

aLetter = ...;  // the code for obtaining the grade has been omitted

if (aLetter == 'A')
    System.out.println("80% - 100%");
else if (aLetter == 'B')
    System.out.println("70% - 79%");
else if (aLetter == 'C')
    System.out.println("60% - 69%");
else if (aLetter == 'D')
    System.out.println("50% - 59%");
else if (aLetter == 'F')
    System.out.println("0% - 49%");
else
    System.out.println("not defined");
```

Looking at the code, we can see that there are 5 if statements and it looks very messy. If we later decide to handle the A+, A-, B+, etc.. cases, then the code will look much longer. There is a better way to write this code.

We can use a switch statement. The switch statement is typically used in situations where we have a sequence of nested if statements in which only one of the if statements is to be executed. The switch statement has the format shown here on the right:

```
// Get the user input
System.out.print("Enter the grade: ");
letter = keyboard.next().charAt(0);

switch(letter) {
   case 'A': System.out.println("80% - 100%"); break;
   case 'B': System.out.println("70% - 79%"); break;
   case 'C': System.out.println("60% - 69%"); break;
   case 'D': System.out.println("50% - 59%"); break;
   case 'F': System.out.println("0% - 49%"); break;
   default:  System.out.println("not defined");
}
```

Multi Value `case` Statements

- With Java 12 or later, you can specify multiple values separated by commas in a `case` statement.

```
switch (number)
{
   case 1, 3, 5, 7, 9:
      System.out.println("Odd");
      break;
   case 2, 4, 6, 8, 10:
      System.out.println("Even");
      break;
   default:
      System.out.println("Number out of range");
}
```

- When any of the `case` values match the `switch` statement's **TestExpression**, the statements in the `case` section are executed.

Demo

- Implement a Simple Calculator using Java switch statement

- Writh a java code where it takes an integer representing a month and display the number of days in that month (Use both if else if and switch case)

Arrow `case` Syntax (1 of 4)

- With Java 12 or later, you can use arrow `case` syntax in a `switch` statement.

```
case value -> statement;
```

- The `case` value is followed by the arrow operator (`->`) which is followed by a single statement.
- When the `case` value matches the `switch` statement's **TestExpression**, the statement to the right of the arrow operator is executed and the program breaks out of the `switch` statement.

Arrow `case` Syntax (2 of 4)

- Example:

```
switch (month)
{
   case 1 -> System.out.println("January");
   case 2 -> System.out.println("February");
   case 3 -> System.out.println("March");
   default -> System.out.println("Error: Invalid month");
}
```

Arrow `case` Syntax (3 of 4)

- Arrow case syntax allows you to specify multiple values in a `case` statement.

```
switch (month)
{
   case 1, 2, 3, 4, 5 -> System.out.println("Spring Semester");
   case 6, 7 -> System.out.println("Summer Semester");
   case 8, 9, 10, 11, 12 -> System.out.println("Fall Semester");
   default -> System.out.println("Error: Invalid month");
}
```

Arrow `case` Syntax (4 of 4)

- If you need to write more than one statement in a `case` section, you must enclose those statements in curly braces

```
switch (month)
{
   case 1 ->
   {
      monthName = "January";
      System.out.println(monthName);
   }
   case 2 ->
   {
      monthName = "February";
      System.out.println(monthName);
   }
   case 3 ->
   {
      monthName = "March";
      System.out.println(monthName);
   }
   default ->
   {
      monthName = "Invalid";
      System.out.println("Error: Invalid month");
   }
}
```