Lecture 03: Flow Control

SE115: Introduction to Programming I

Previously on SE 115…

- Values
- Variables
- Evaluation
- Assignment
- Types
- Formatted output

Flow

So far, a program flowed top to bottom, line by line.

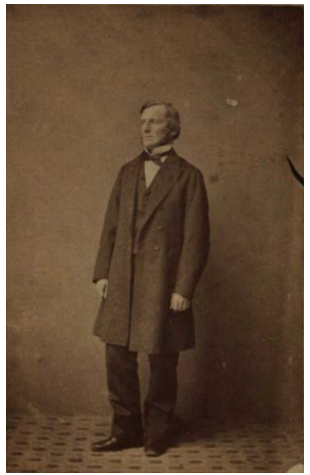
```
import java.util.Scanner;

public class Age {
    public static void main(String args[]) {
        int age = -1;
        Scanner sc = new Scanner(System.in);
        System.out.print("Please enter your age: ");
        age = sc.nextInt();
        System.out.println("You are " + age + " years old.");
    }
}
```

Flow

- It is possible to change the flow of the program depending on one or more conditions.
- For example, if the age is less than 18, then the program might refuse to continue.
- This check creates two paths of execution;
  - If the age is under 18, then the program might print a message and end.
  - Otherwise, it can continue to operate and let the user see other options.
- The same checks can be done to control if a valid value is entered.
- To do so, we have to first remember logical operations.

Boolean Logic

- George **Boole** introduced the algebra in which the values of the variables are the truth values **true** and **false**, usually denoted 1 and 0. They are called **boolean** values after him.
- In programming, we make use of true and false to evaluate statements.
- For the variable "`int x = 5;`" we may check if "`x == 5;`"
  - if x is equal to five: then the statement is TRUE,
  - if x is equal to four: then the statement is FALSE
- We have several operators for testing values.



Boolean Operators

Operator	**Java**	**Example**
Equal	`==`	`x == 5;`
Not Equal	`!=`	`x != 5;`
Less than	`<`	`x < 3;`
Greater than	`>`	`x > 4;`
Less than or equal to	`<=`	`x <= 3;`
Greater than or equal to	`>=`	`x >= 6;`
Negation	`!`	`!(x == 4);`

Boolean Operators

```
public class BooleanVals {
    public static void main(String args[]) {
        int x = 4;
        System.out.println("3 == 3 is " + (3 == 3));
        System.out.println("x == 3 is " + (x == 3));
        System.out.println("x != 3 is " + (x != 3));
        System.out.println("x < 5 is " + (x < 5));
        System.out.println("x <= 9 is " + (x <= 9));
    }
}
// What will be the output?
```

Boolean Operators

- Remember “values” - the evaluation of each statement that has a boolean operator returns a value.
  - We can store them in a variable if we want to.
- Notice how I have used the parenthesis to keep the evaluation of `x == 5` separate from the assignment of its value to **v**.
- Let’s talk about “operator precedence!”

```
public class BooleanValues {
    public static void main(String args[]) {
        int x = 5;
        boolean v = (x == 5);
        System.out.println(v);
    }
}
```

Operator Precedence

- In mathematics, there is a rule where multiplication takes precedence over addition. For example, the following statement
  2 + 3 x 5
  equals 17, because the multiplication has precedence over addition, so 3 and 5 are multiplied before the addition.
- Similarly, the compiler needs to know the order of operations.
- Since the compiler is a program, it is deterministic: there cannot be any uncertainty.
- For our current list of operators, the precedence is given in the following table.

Operator	Precedence
parentheses	`()`
postfix	`expr++ expr--`
unary	`++expr --expr !`
multiplicative	`* / %`
additive	`+ -`
relational	`< > <= >=`
equality	`== !=`
logical AND	`&&`
logical OR	`||`
assignment	`= += -= *= /= %=`

- Operators with higher precedence are evaluated before operators with relatively lower precedence.
- The closer to the top of the table an operator appears, the higher its precedence.
- Operators on the same line have equal precedence.
- When operators of equal precedence appear in the same expression,
  - all binary operators except for the assignment operators are evaluated from **left to right**;
  - assignment operators are evaluated **right to left**.

Operator	Precedence
parentheses	`()`
postfix	`expr++ expr--`
unary	`++expr --expr !`
multiplicative	`* / %`
additive	`+ -`
relational	`< > <= >=`
equality	`== !=`
logical AND	`&&`
logical OR	`||`
assignment	`= += -= *= /= %=`

- Therefore, in the expression
  `boolean v = (x == 5);`
  the parenthesis is done first.
- The equality operator (==) has precedence over assignment operator. Therefore,
  `boolean v = x == 5;`
  would have resulted in the same value.
- However,
  `boolean v = (x == 5);`
  greatly improves readability for us, humans.

Operator Precedence

- So, refrain from following mathematical notation when programming.
  - z = x - y < 5;
  - z = 3 < x < 5;
- Compiler will never get confused. Instead, you will confuse yourself.
- This is what the computers call “human error.”
- Be **clear** when you are writing code;
  - It is good for you when you read it at a later time,
  - It is also good for the compiler since it can generate more optimized code if it can understand better what you are trying to do.

Operator Precedence

- So, refrain from following mathematical notation when programming.
  - z = x - y < 5;
  - z = 3 < x < 5;
- Compiler will never get confused. Instead, you will confuse yourself.
- This is what the computers call “human error.”
- Be **clear** when you are writing code;
  - It is good for you when you read it at a later time,
  - It is also good for the compiler since it can generate more optimized code if it can understand better what you are trying to do.
- This does not mean, of course, we won’t ask them in the exams *<<insert evil laughter here>>*.

Boolean Operations

- We have a few boolean operations, such as AND and OR and NOT.
- Logical AND is true only when both statements are true.
  - `True  AND True  is True`
  - `True  AND False is False`
  - `False AND True  is False`
  - `False AND False is False`
  - It is raining AND I'm happy (check this depending on the weather and your mood).
- Logical OR is false only when both statements are false.
  - `True  OR True  is True`
  - `True  OR False is True`
  - `False OR True  is True`
  - `False OR False is False`
  - I'm old OR you are young (check this with a volunteer).

Boolean Operations

- NOT is a unary operator;
  - “NOT True” is False
  - “NOT False” is True
- It is raining.
- NOT (It is raining).
  - Or, like a normal person, “It is NOT raining.”
- You will enjoy such logical operators in CE 215 Discrete Mathematics.
- For now, let’s get back to programming.

Boolean Operations

- In Java, the AND operator is &&, OR operator is ||, and NOT operator is !.
- So the statement

  `x == 5 && x == 4;`

  is definitely **false** since x cannot be equal to both values at the same time.
- The statement

  `x == 5 || x == 4;`

  depends on the value of x.
- The statement

  `!(x == 5 && x == 4)`

  is definitely **true** since we use the NOT operator on a **false** value.

Boolean Operations

- `x == 5` returns **true** if x is 5,
- `x != 5` returns **true** if x is not 5.

- Since we are now familiar with these boolean expressions and operators, let's learn how to use them to "branch" our program.

If…

- Java is a “high-level” language. That means, it is closer to human language than machine language.
- The most basic flow control is the “if” clause.
- Similar to English, we write

```
if (condition) {
  // code that will run when the condition is true
}
```

- Notice the block: { }
- Here is one example.

If...

```
import java.util.Scanner;

public class ChangeFlowIf {
    public static void main(String[] args) {
        int grade = -1;
        Scanner sc = new Scanner(System.in);
        System.out.print("Please enter your grade: ");
        grade = sc.nextInt();
        if(grade < 60) {
            System.out.println("You have failed this course.");
        }
    }
}
```

This line is executed only if the grade is smaller than 60.

Notice the indentation to improve the visual readability of the code.

If…else

- The if clause lets the following block of code to run when the condition is true. What if the condition is false?
- Let me introduce “else”

```
if (condition) {
  // run when the condition is true
} else {
  // run when the condition is false
}
```

- Notice that there is no **condition** in “else.”

If…else

```
import java.util.Scanner;

public class ChangeFlowIf {
    public static void main(String[] args) {
        int grade = -1;
        Scanner sc = new Scanner(System.in);
        System.out.print("Please enter your grade: ");
        grade = sc.nextInt();
        if(grade < 60) {
            System.out.println("You have failed this course.");
        } else {
            System.out.println("You have passed this course!");
        }
    }
}
```

Runs for values that are less than 60.

Runs for all other values that are NOT less than 60.

If { … } else if { … } else { … }

- What if there are conditions that I want to check?
- What if instead of “pass” and “fail” I want to return the letter grade.
- I can try several if clauses, but instead we can use

```
if (condition) { … }
else if (another condition) { … }
else if (another condition) { … }
else { … }
```

Points	Grades	Coefficient	Status
90-100	AA	4,00	Successful
85-89	BA	3,50	Successful
80-84	BB	3,00	Successful
75-79	CB	2,50	Successful
70-74	CC	2,00	Successful
65-69	DC	1,50	Successful
60-64	DD	1,00	Successful
50-59	FD	0,50	Unsuccessful
49 ve below	FF	0,00	Unsuccessful

```java
import java.util.Scanner;

public class ChangeFlowIfElse {
    public static void main(String[] args) {
        int grade = -1;
        Scanner sc = new Scanner(System.in);
        System.out.print("Please enter your grade: ");
        grade = sc.nextInt();
        if (grade < 50 && grade >= 0) {
            System.out.println("You get an FF.");
        } else if (grade >= 50 && grade < 60) {
            System.out.println("You get an FD.");
        } else if (grade >= 60 && grade < 65) {
            System.out.println("You get a DD.");
        } else if (grade >= 65 && grade < 70) {
            System.out.println("You get a DC.");
        } else if (grade >= 70 && grade < 75) {
            System.out.println("You get a CC.");
        } else if (grade >= 75 && grade < 80) {
            System.out.println("You get a BC.");
        } else if (grade >= 80 && grade < 85) {
            System.out.println("You get a BB.");
        } else if (grade >= 85 && grade < 90) {
            System.out.println("You get a BA.");
        } else if (grade >= 90 && grade <=100) {
            System.out.println("You get an AA");
        } else {
            System.out.println("This is not a valid grade...");
        }
    }
}
```

A Few Remarks

- The “else if” blocks can be as many as you like.
- **Good practice:** We tend to start from the most probable case, so that the branching would be over as soon as possible.
- **Good practice:** Notice how I have protected the boundaries of grading; all values less than 0 and greater than 100 are rejected as “invalid.” Never trust a value that comes from another source, such as a user, a file, a network connection, etc.
- Let’s work on another example.

Nesting if clauses

```
import java.util.Scanner ;

public class FlowIfTime  {
    public static void main(String[] args)  {
        Scanner sc  = new Scanner(System.in);
        int t = -1;
        System.out.print("What is the time? " );
        t  = sc.nextInt();
        if (t < 10)  {
            if (t > 6)  {
                System.out.println("Good morning!" );
            } else  {
                System.out.println("Sleeping!" );
            }
        } else if (t < 14)  {
            if (t == 12)  {
                System.out.println("It is noon!" );
            } else  {
                System.out.println("Good day!" );
            }
        } else if (t < 20)  {
            System.out.println("Good evening" );
        } else  {
            System.out.println("Good night!" );
        }
    }
}
```

Switch cases

- Switch blocks are similar to “if clauses” but there are some differences.
- There is no “condition” but a set of “cases”.

```
switch(variable) {
  case 1:
    // runs when variable == 1
    break;
  case 2:
    // runs when variable == 2
    break;
  default:
    // runs when there is no match (similar to else)
    break;
}
```

Switch

```
import java.util.Scanner;

public class FlowSwitch {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int m = -1;
        System.out.print("Enter month: ");
        m = sc.nextInt();
        switch(m) {
            case 1:
                System.out.println("January");
                break;
            case 2:
                System.out.println("February");
                break;
                /* other months here... */
            default:
                System.out.println("Invalid Month");
                break;
        }
    }
}
```

```java
import java.util.Scanner;

public class FlowSwitchSeasons {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int m = -1;
        System.out.print("Enter month: ");
        m = sc.nextInt();
        switch(m) {
            case 12:
            case 1:
            case 2:
                System.out.println("Winter is coming...");
                break;
            case 3:
            case 4:
            case 5:
                System.out.println("Spring is here!");
                break;
            case 6:
            case 7:
            case 8:
                System.out.println("Summer Holiday!");
                break;
            case 9:
            case 10:
            case 11:
                System.out.println("Fall!");
                break;
            default:
                System.out.println("Invalid Month");
                break;
        }
    }
}
```

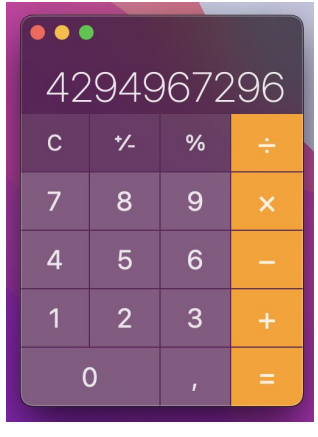
Here is a fun one.

Remarks for switch cases

- The “break” is important, because once the execution flows into a case, it keeps running all cases regardless of their values.
  - This can be used in some cases, but almost all the time we “break” out of a case when we are done.
- Also notice that there is no “condition”, but implicitly the value of the variable is compared to the values in the “case”s.

Welcome to the world of programming!

- With the introduction of the flow, you are now capable of using programming to do more than basic calculator operations!
- You can do “programming” now, depending on the input.
- Try the following.
  - Ask the user to enter two integers, then tell them which is greater than the other, or if they are equal.
  - Ask the user which operation they would like to perform: addition, subtraction, multiplication, and division. You can list them and assign a number to each. Then, ask for two integers and perform the operation the user asks for.
    - If the user enters an invalid operation, you can “return” and exit the program before asking for the numbers.



More programming

- While you are at it:
  - Write a program that asks for the day as integers (Monday is 1, Sunday is 7) and tell the user if it is a weekday or weekend. Use both “if” and “switch,” so there will be two programs.
- Another homework:
  - Consider the letter grade program. Do we really need to check if grade >= 50 if we have just checked in the previous “if clause” whether grade < 50. Since grade < 50 is not true, then maybe …? Rewrite this program with fewer conditions.
- An easy one:
  - Ask for a number, and then tell if it is negative, zero, or positive.
- Intervals are tricky:
  - The working hours are 9 to 12 and 13 to 17. The lunch break is at 12 to 13. Ask for the time, and depending on the time tell either you are working, at lunch, or not at work.

Next Week

- Conditions will always be with us.
- Next week we'll be learning "loops" which also makes use of conditions to make a change in the flow of a program.
- So, make sure you understand the conditions and the flow very well.