Programming Fundamentals

Control Flow Statements

Week 4| Iresh Bandara

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

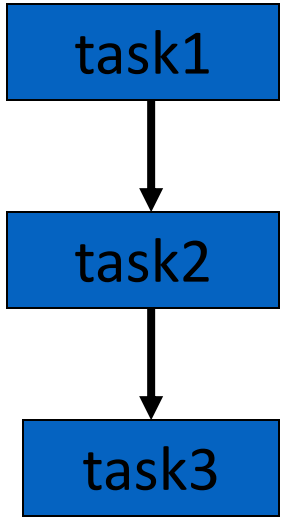
Learning Outcomes

- Covers part of LO1, LO2 & LO3 for Module

- On completion of this lecture, students are expected to be able to:
  - Recognize the appropriate places where a conditional operator is required.
  - Apply conditional statements to a given problem.
  - Substitute If-else with switch statements.

No `goto` in Java

- Java executes one statement after the other in the order they are written.



Control Flow Statements

- Control flow statements, break up the flow of execution by:
  - Decision making
  - Looping
  - Branching
- These are used to conditionally execute particular blocks of code.

Types of Control Flow Statements

- Decision-making statements:

  if, if-else, switch

- Looping statements:

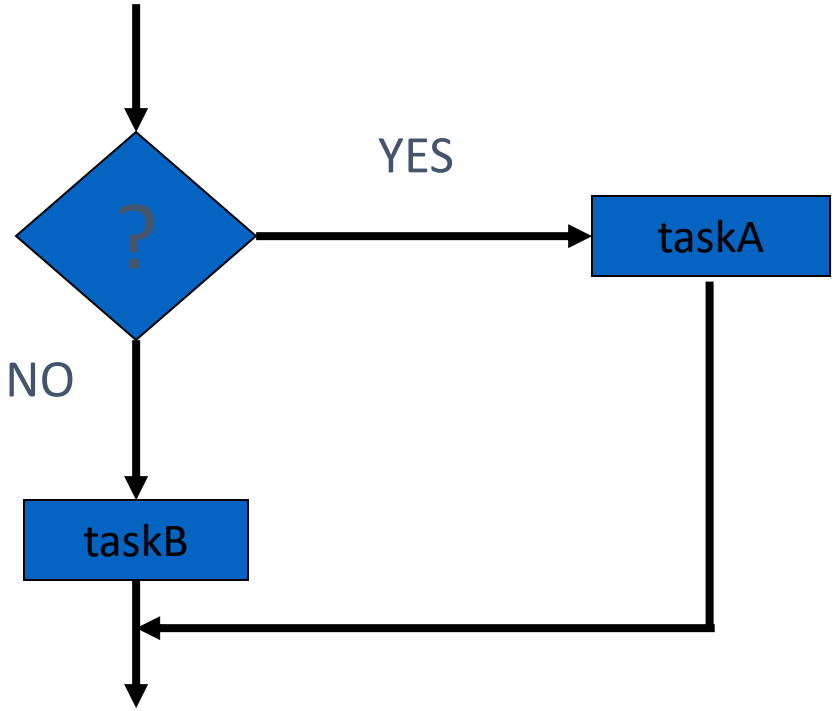
  while, do-while, for

- Branching statements:

  break, continue, return

Decision Making/Selection

- Often we want the computer to carry out a test on some data and then take one of a choice of actions depending on the result of the test.



The `if` Statement

- The `if` statement is intended for selecting a choice between two possible paths.
- Hence it associates with "boolean expressions".

Comparing Values: Relational Operators

- To compare numbers we can use relational and logical operators.

<, >, <=, >=, ==, != ← Relational

&&, ||, ! ← Logical

- Eg: The **==** denotes equality testing

```
a = 5; // Assign 5 to a
if (a == 5) . . . //Test whether
          a equals 5
```

Comparing Floating-Point Numbers

- To avoid round off errors in floating point values, don't use `==` and `!=` to compare floating-point numbers.

```
double x = 7.0;
double y = 3.5;
if (x / y == 2.0)
        ...
```

May be false!

- To compare floating-point numbers test whether they are close enough.

Comparing Strings

- Don't use **==** for strings!

```
if (input == "Y") // WRONG!!!
```

- Use **equals()** method:

```
if (input.equals("Y"))
```

- For case insensitive test ("Y" or "y") use **equalsIgnoreCase()**

```
if (input.equalsIgnoreCase("Y"))
```

- **s.compareTo(t) < 0** means:
  s comes before **t** in the dictionary

Testing for null

- Use **==** or **!=** when testing for **`null`**.

```
if ( text != null )    ...
```

- **`null`** is not the same as the empty string "”

Exercise 1

- What is the value of **`s.length()`** if **`s`** is
  a. *the empty string* `""`?
  b. *the string* `" "` *containing a space?*
  c. ***`null`***?
- **Answer:** (a) `0`; (b) `1`; (c) an exception is thrown.

Different `if` Statements

- Use the if statement depending on the complexity of conditions to be tested.
  - Simple if statement
  - If-else statement
  - Nested if - else statement
  - else if ladder

Simple `if` Statement

- The most simple form:

```
if ( <condition> )
{
    < statements >
}
```

- If action involves only one statement, the **{ }** braces are not needed, but you are advised to include them as a matter of habit.

if-else Statement

- Often we find that we want to specify two courses of action,
  - One to carry out if the condition is true, and
  - The other to perform if it is false

```
if ( <condition> ){
    < statements >
}
else{
    < other statements >
}
```

Nesting of **if-else** Statements

```
if ( <condition> ){
    if ( <sub condition> )
     < code block 1 >
    else
     < code block 2 >
}
else{
    if ( <sub condition> )
     < code block 3 >
    else
     < code block 4 >
}
```

The **if-else-if** Ladder

```
if ( <condition 1> ){
    < statements >
}
else  if ( <condition 2> ){
    < statements >
}
else{
    < other statements >
}
```

Default Statement

Exercise 1b

Exercise 1:
What is the output of each of the following code fragments?
(given the declaration int a=1, b=2, c=3;):

1.
```
if (6 < 2 * 5)
System.out.print("Hello");
System.out.print(" There");
```

2.
```
if(a>b)
if(a>c)
System.out.println("1111");
else
System.out.println("2222");
```

3.
```
if (a < c)
System.out.println("*");
else if (a == b)
System.out.println("&");
else
System.out.println("$");
```

4.
```
if(a<b)
System.out.println("####");
else
System.out.println("&&&&");
System.out.println("****");
```

5.
```
if(a>b)
System.out.println("####");
else
{System.out.println("&&&&");
System.out.println("****");}
```

6.
```
int x = 100; int y = 200;
if (x > 100 && y <=200)
System.out.print(x+" "+y+" "+(x+y));
else
System.out.print(x+" "+y+" "+(2*x-y));
```

7.
```
if (a < c)
System.out.println("*");
else if (a == c)
System.out.println("&");
else
System.out.println("$");
```

8.
```
if(++a > b++ || a-- > 0)
c++;
else
c--;
System.out.println(a+" "+b+" "+c);
```

9.
```
if(a<b){
System.out.println("####");
System.out.println("****");
}
else
System.out.println("&&&&");
```

10.
```
if ('a' > 'b' || 66 > (int)('A'))
System.out.println("#*#");
```

Exercise 2

- Find errors, if any, in each of the following sements:

```
1. if(x+y = z && y>0)
2. if(code>1);
        a = b+c
   else
        a = 0
1.  if(p<0) | | (q<0)
```

The `switch` Statement

- For decisions involving many possible paths and values we use the `switch` statement.
- At the end of each sequence we insert a `break` statement that means 'go to the end of the switch'.

The **switch** Statement

```
switch (variable)
{
  case value1:
        ...
        break;

  case value2:
        ...
        break;
   ...
   ...
  default:
        ...
        break;
}
```

switch

case

default

break

Reserved words

Don’t forget breaks!

switch – Example

- The same **case** can have two or more labels. For example:

```
switch (num)
{
 case 1:
 case 2:System.out.println ("Buckle your shoe");
        break;
 case 3:
 ...
}
```

Rules that applies to a `switch` statement

- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch, and it must be a constant or a literal (character).
- When the variable being switched on is equal to a case, the statements following that case will execute until a *break* statement is reached.

Rules that applies to a `switch` statement

- When a *break* statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A *switch* statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Exercise 2b

- Convert the following switch statement into if-else statements then into if-then statements:

```
String dayString1, dayString2, dayString3;
int day = KB.nextInt();
switch (day) {
  case 1: dayString1 = "Saturday";
  case 2: dayString2 = "Sunday";
          break;
  case 3: dayString3 = "Monday";
          break;
  case 4: dayString1 = "Tuesday";
  case 5: dayString2 = "Wednesday";
          break;
  default: dayString3 = "Invalid day";
           break;
}
```

The Conditional Operator (?:)

- Useful for making two-way decisions.
- When this operator is used, the code becomes more concise and more efficient.
- **BUT**, readability is poor.
- **Better** to use `if` statements when more than a single nesting of conditional operator is req

Exercise 3

- Write a program that will read the value of x and evaluate the following function

$$y = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{for } x = 0 \\ -1 & \text{for } x < 0 \end{cases}$$

- Using
  - nested **if** statements,
  - **else if** statements,
  - conditional operator **?:**

Look at the given code…

```
import java.util.*;
class ScannerDemo {

    public static void main(String[] args) {
        int age;
        String name;
        Scanner s = new Scanner(System.in);

        System.out.print("Enter first and last name:");
        name = s.nextLine();

        System.out.print("How old are you? ");
        age = s.nextInt();

        System.out.println(name + "\t" + age);
    }
}
```

Input Errors

- What happens if the user doesn't enter an integer when asked for the age?
- There are a couple of ways to handle it.
  - Look ahead to see if the user entered an integer before we read it.

    or
  - Read the input and handle the resulting error.

Solution 1 - Look Ahead

- Scanner provide the ability to look at the next token in the input stream before we actually read it into our program.
  - hasNextInt()
  - hasNextDouble()
  - hasNext()
  - etc…

```
if (s.hasNextInt())
{
   age = s.nextInt(); //read user input(age)

}
else
{
   age = 30; //assign a default value
   String junk = s.next();
}
```

Exercise 4

- Write a program to read the number of novels you have as an integer.
  - If it is greater than 20 – display “Wow!”
  - If it is less than or equals to 20 – display “Not Bad”
  - If it is equal to zero – display “Buy One Now!!”
- Validate the input for its data type. In case of a wrong data type, exit from the program.

Summery

- Java executes one statement after the other in the order they are written.
- Decision-making statements: if, if-else, switch
- The if statement is intended for selecting a choice between two possible paths.
- Use the if statement depending on the complexity of conditions to be tested.
  - Simple if statement
  - If-else statement
  - Nested if - else statement
  - else if ladder

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.print("Enter the number of novels you have: ");
            int numNovels = scanner.nextInt();
            if (numNovels > 20) {
                System.out.println("Wow!");
            } else if (numNovels <= 20 && numNovels > 0) {
                System.out.println("Not Bad");
            } else if (numNovels == 0) {
                System.out.println("Buy One Now!!");
            }
        } catch (Exception e) {
            System.out.println("Invalid input! Please enter a valid integer.");
        }
    }
}
```

Thank you