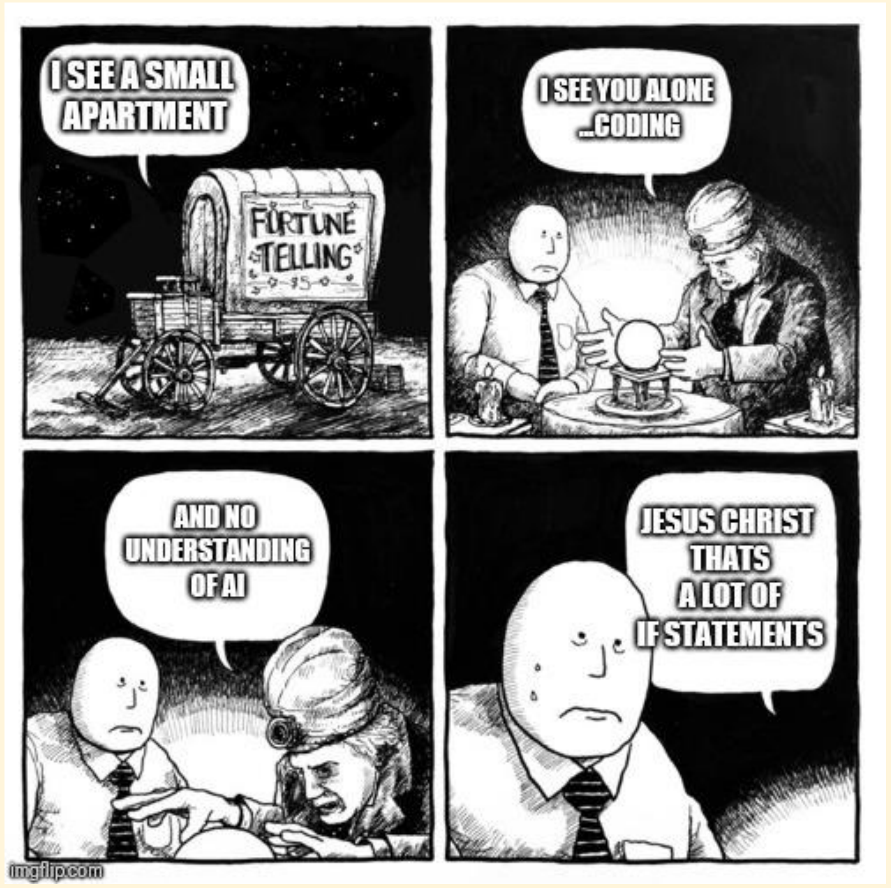
Module 03
Basic Java: Selection and Repetition


I SEE A SMALL APARTMENT
FORTUNE TELLING $5
I SEE YOU ALONE ...CODING
AND NO UNDERSTANDING OF AI
JESUS CHRIST THATS A LOT OF IF STATEMENTS
imgflip.com

CS 225 Computer Science II
Embry-Riddle Aeronautical University
Daytona Beach, FL

Module 03 Outcomes

After completing this module you should be able to …

1. Distinguish basic blocks of code from branching or repeated code.
2. Explain the three components required for all selection constructs (if-then-else).
3. Write, compile, and execute if-then-else constructs, including nested if-then-else forms.
4. Identify the true and false branches of an if-then-else block.
5. Explain the three components required for all repetition constructs (for and while loops).
6. Write, compile and execute for loops and while loops.
7. Identify the three repetition components in for and while loops.
8. Identify the body of a loop.

5 Things Every Programming Language MUST do

- Create / declare variables
- Bind values to variables (assignment)
- ➡ Select sections of code to execute (if-then-else)
- ➡ Repeat sections of code (for, while loops, recursion)
- Input/Output (I/O)

This module discusses code selection and repetition.

Selection – If-Then-Else

- Selection: the ability to select which sections of code will or will not execute.
  - Not a standard term – instructor's terminology. Covers all possibilities.
- A program that does not respond to user input is useless (and maddening!)
- Selection allows the program to change behavior based on user input.
- User input may be …
  - actions by a human sitting at the computer/controller (word processing, game play)
  - file input providing data (databases for schools/businesses/organizations)
  - data or requests sent from other computers (browsing/shopping/texting online)
- Typically achieved using If-Then-Else forms
  - "If something is true, do this, else do that."
- Basic If-Then-Else form is required, other forms exist for convenience.
  - This is called "syntactic sugar" – doesn't add to computational power, but sweet for programmers.

3 Things Needed for Selection

1. Boolean Test: An expression that evaluates to True or False
2. Something to do if the Boolean test is True (the True branch)
3. Something to do if the Boolean test is False (the False Branch)

Everything else is simply formatting.

Matlab

```
If ( <test> )
   <True Branch>
 else
  <False Branch>
end
```

Java, C, C++

```
If ( <test> ) {
   <True Branch>
} else {
  <False Branch>
}
```

Python

```
If ( <test> )
   <True Branch>
else
  <False Branch>
```

Excel

```
= If( <test>, <True Branch>, <False Branch>)
```

The Boolean Test – Boolean Operators

- Boolean refers to the mathematical system having only two values: True and False
- Basic Boolean tests in Java:

a == b	“is equal to”
a != b	“is not equal to” (the ! is called the “not operator”)
a > b	“is greater than”
a >= b	“is greater than or equal to”
a < b	“is less than”
a <= b	“is less than or equal to”

- All of the above are QUESTIONS not STATEMENTS.
- Boolean operations: AND, OR, and NOT

A && B	“A AND B” True if and only if both A and B are true
A || B	“A OR B” True if and only if A is true, or B is true, or both are true
!A	“NOT A” True if and only if A is False

If-Then-Else

```
If ( x > y ) {
   System.out.println(“Yes”);
} else {
   System.out.println(“No”);
}
```

```
If ( x > y ) {
   System.out.println(“Yes”);
}
```

The False branch may be omitted.

```
If ( x > y ) {

} else {
   System.out.println(“No”);
}
```

The True branch may be empty.

Never do this!

Nested If-Then-Else

- You can place an If-Then-Else inside of another If-Then-Else (nesting)
- This can be done repeatedly.
- Usually it means bad code design, sometimes you can’t help it.

```
If ( x > y ) {

    if ( x> z) {
        System.out.println(“Yes”);
    } else {
        System.out.println(“Huh?);
    }

} else {
   System.out.println(“No”);
}
```

ElseIf

- Additional Boolean tests may be added using the “else if” form.
- Not terribly common

```
If ( x > y ) {
    System.out.println(“Yes”);
} else if (x > z) {
    System.out.println(“Why?”);
} else if ( x > p) {
    System.out.println(“Why not?”);
} else {
   System.out.println(“No”);
}
```

Switch Statements

- Used when an attribute/variable may take on a finite number of specific values.
- Specify the attribute/variable to be tested.
- Each “case” represents a specific value that attribute/variable may contain.
- Include a “break;” statement at the end of each choice, or all cases after the current one execute.
- Include a default case even when you know you don’t need it.

```
switch ( x ) {
 case 1:   System.out.println(“Tinkers”);
           break;
 case 2:   System.out.println(“Evars”);
           break;
 case 3:   System.out.println(“Chance”);
           break;
 default;  System.out.println(“No triple play today.”);
}
```

Boolean Operators to Combine Logical Tests

Read this as “If (x>y) AND (x>z) then ….”

```
If (( x > y )  && (x > z)) {
    System.out.println(“Yes”);
} else {
   System.out.println(“No”);
}
```

Code clarity tip: replace complicated Boolean tests with a method call:

```
If ( myBooleanTest() ) {
    System.out.println(“Yes”);
} else {
   System.out.println(“No”);
}

public boolean myBooleanTest() {
     return (( x > y )  && (x > z)) ;
}
```

Give the method a descriptive name.

Repetition – Iteration and Recursion

- Repetition: the ability to execute sections of code repeatedly.
  - Not a standard term – instructor's terminology. Covers all possibilities.
- Repetition allows the program to change behavior based on user input.
  - Typically to respond to changes in the number of items in a data set.
  - Also used to repeat actions until some condition is met.
- Typically achieved using the For Loop (iteration)
  - Used when the required number of iterations is known
- Sometimes achieved using While Loops (iteration)
  - Used when the required number of iterations is not known
- Occasionally achieved using recursion
  - Recursive forms are when methods call themselves repeatedly
  - Some languages only allow for recursion
  - Not covered in this module

3 Things Needed for Repetition

1. Starting condition: controls when and how the repetition starts.
2. Stopping condition: controls when the repetition stops (important!).
3. Progress: each repetition moves toward the stopping condition

Everything else is simply formatting.

For Loops

```
for ( start condition; stopping condition; progress ) {
    <Loop body>
}
```

While Loops

```
<starting conditions set before the loop>
while ( stopping condition ) {
    <Loop body – hopefully has progress!>
}
```

A note about terminology.

The “stopping condition” in the loops are expressed as “continuing conditions.”

The loops continue as long as the condition is met.

Confusing? A little at first.

Java For Loop Example

1. Calculate the sum of the first n integers.

```
n = input.nextInt();
sum = 0;

for ( i = 1; i < n + 1; i = i + 1 ) {
   sum = sum + i;
}
```

<u>Starting Condition: “i = 1”</u>

- Variable “i” is commonly used.
- If nested loops, use i, j, k, l, …
- Why? Tradition!
- Contrary to my “descriptive names” advice
- Sets variable i to 1 at start of loop.
- Will almost always be “i = 0;” for arrays.
- Ensure you start at the right place! Easy error.

Java For Loop Example

1. Calculate the sum of the first n integers.

```
n = input.nextInt();
sum = 0;

for ( i = 1; i < n + 1; i = i + 1 ) {
   sum = sum + i;
}
```

Stopping Condition: “i < n + 1”

- Technically, it’s the continuing condition.
- Loop continues as long as “i < n + 1” is True.
- This form is typical, will change for arrays.
- Any Boolean test can go here!
- Ensure you stop at the right place! Easy error.

Java For Loop Example

1. Calculate the sum of the first n integers.

```
n = input.nextInt();
sum = 0;

for ( i = 1; i < n + 1; i = i + 1 ) {
   sum = sum + i;
}
```

<u>Progress: “i = i + 1”</u>

- Java shortcut: i++ (means to increment i).
- Almost always look this way, but ….
- Anything can go here!

Java While Loop Example

1. Calculate the sum of the first n integers.

```
n = input.nextInt();
sum = 0;
i = 1;

while ( i < n + 1 ) {
   sum = sum + i;
   i = i + 1;
}
```

<u>Starting condition: “I = 1;”</u>

<u>Stopping condition: “I < n + 1”</u>

<u>Progress: “i = i + 1”</u>