Introduction to Programming

with Java, for Beginners

Control Structures

Sequential Control Flow

- Consists of just a list of commands to be done in order

Welcome to Dr. Java
> int num = 2;
> int sqNum;
> sqNum = num * num;
> System.out.println(sqNum); //Or just type: sqNum

2

What are control Structures?

- Limitations of sequential control flow
  - Cannot choose whether or not to perform a command/instruction
  - Cannot perform the same command more than once

- Such programs are extremely limited!

- Control structures allow a program to base its behavior on certain conditions

Recap: Boolean

- Boolean is one of the eight primitive types
  - Only 2 value: true, or false
  - Booleans are used to make yes or no decisions
  - All control structures use Booleans

- The following expression each give a Boolean result:
  - (25 > 24) && (12 == 13) //results to false
  - (25 > 24) || (12 == 13) //results to true

- Thus based on certain conditions we can alter the outcome or flow of the program
Conditionals ("if" statements)

- An "if" statement is a flow control statement.
- It is also called a conditional, or a branch.
- We'll see several "flavors":
  - An "if" all by itself
  - An "if" with an "else" part
  - An "if" with an "else if" part

"if" statement

```java
if ((x % 2) == 0) {
    System.out.println(x + " is even");
}
```

If the condition is true, then the statement(s) (i.e. instructions) will be executed. Otherwise, it/they won't.

"if-else" statement

```java
// Assume x is an integer
if ((x % 2) == 0) {
    System.out.println(x + " is even");
} else {
    System.out.println(x + " is odd");
}
```

Example: "if-else" statement

<table>
<thead>
<tr>
<th>Example</th>
<th>Syntax &amp; Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>//Determine smaller of two integers numbers &amp; put it variable &quot;min&quot;</td>
<td>if (condition) {</td>
</tr>
<tr>
<td></td>
<td>int min;</td>
</tr>
<tr>
<td></td>
<td>int x = 5;</td>
</tr>
<tr>
<td></td>
<td>int y = 10;</td>
</tr>
<tr>
<td></td>
<td>if (x &lt;= y) {</td>
</tr>
<tr>
<td></td>
<td>min = x;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>else {</td>
</tr>
<tr>
<td></td>
<td>min = y;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

If the condition is true, then the statement(s) in the "if block" are executed. Otherwise, if there is an "else" part, the statement(s) in it are executed.
if with else-if statement

```java
if (condition){
    statement(s)
}
else if (condition){
    statement(s)
} else { ...
    ...
    else {
}
```

- No limit on “else if” statements
- The “else” condition is usually a default condition
- For completeness

Cascading “if-else”

```java
class Example {
    public static void main(String[] args) {
        char userChoice;
        // Ask user for input and store it in userChoice
        if (userChoice == 'q')
            System.out.println("quitting.");
        else if (userChoice == 'a')
            System.out.println("adding.");
        else if (userChoice == 's')
            System.out.println("saving.");
        else
            System.out.println("unrecognized choice.");
    }
}
```

//Note: You can avoid the curly brace after condition if only one statement is to be performed

Nested if-statements

**Truth Table**

<table>
<thead>
<tr>
<th>An if within an if</th>
<th>Truth Table</th>
</tr>
</thead>
</table>
| if (condition1){
  if (condition2){
    statement(s) A
  } else{
    statement(s) B
  }
} else {
  statement(s) C
} | What values must the conditions have in order for block A to run? B? C? |

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition1</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>condition2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The infamous “dangling else”

<table>
<thead>
<tr>
<th>Code</th>
<th>Notes</th>
</tr>
</thead>
</table>
| if (x > y){
  if (y < z)
      statementA;
  else
      statementB;
} else{
  statementB;
} | When is statementB executed? |
|      | In other words, which if is the else paired with? |

- An else is paired with the last else-less if, regardless of spacing, unless {} dictate otherwise.
A "Loop"

- A simple but powerful mechanism for "making lots of things happen!"
- Performs a statement (or block) over & over
- Usually set up to repeat an action until some condition is satisfied
- Computing Scenarios Examples
  - Run an application until user hits "quit" button
  - Read characters until end of file reached
  - Deal card hands until game over

Syntax of the **while** statement

```java
while (condition) {
    statement(s)
}
```

- **condition** is a true/false (boolean) expression
- If **condition** is initially false, the statement is never executed
- If **condition** is true, **statement** is executed and **condition** is re-evaluated
- **The statement** should eventually make the loop stop

A **while** Loop to Print Numbers

```java
// Print the numbers 1 thru 10
int x = 1;
while (x <= 10){
    System.out.println(x);
    x = x + 1;
}
```

What happens if you forget the statement \(x = x + 1\)?

- We print value 1 forever
- Known as **infinite loop**

More **Infinite Loops**

```java
// Some infinite loops are intentional
while (true){
    statement(s)
}
```

// Others are not
```java
int x = 5;
while (x < 10){
    statement(s) which don't change x
}
```
Compute Square of first 10 numbers

//In Square.java
int num = 1;
int sqNum = 0;
while (num <= 10) {
    sqNum = num * num;
    System.out.println(num + "  " + sqNum);
    num = num + 1;
}

For Loop

for (init; end-test; re-init){
    statement
}
- Executes loop body as long as end-test evaluates to TRUE
- Initialization and re-initialization code included in loop statement
- Note: Test is evaluated before executing loop body

While vs. For

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| int x = 1;
while (x <= 10){
    System.out.println(x);
    x = x + 1;
} | An example of a while loop that has this pattern |
| for (int x = 1; x <= 10; x = x + 1){
    System.out.println(x);
} | A for loop that does the same thing |

Note: For loops are used generally for bounded iteration

Summary of Loops

<table>
<thead>
<tr>
<th>Type of Loop</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| while | while (condition){
    statement(s)
} |
| for | for (expr1; condition; expr3){
    statement(s)
} |