Lecture Set #5: If Statements

Last time:
1. Variables and types
2. Expressions in Java
3. User input with Scanner objects

This set:
1. if statements
Control Flow and Conditionals

- **Control flow**: the order in which statements are executed
  
  General rule: top to bottom

- **Conditional statements** permit control flow to be dependent on (true/false) conditions
  
  - `if`
  
  - `if-else`
The if Statement

- Form:
  
  ```java
  if ( <boolean-expression> ) {
    <statement>
  }
  ```

- Example:
  
  ```java
  if (inchesOfSnow > 7) {
    System.out.println( "Go home" );
  }
  ```

- The `println` statement is executed only if the variable “inchesOfSnow” is greater than 7
- Otherwise, it is skipped
Example 6

```java
public class Example6 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int i = sc.nextInt();

        if (i < 0){
            System.out.println("That was a negative number!");
        }
        System.out.println("The number was " + i);
    }
}
```
The if-else Statement

- Form:
  ```
  if ( <condition> ){
    <statements 1>;
  }else{
    <statements 2>;
  }
  ```

- Example:
  ```java
  if (inchesOfSnow > 7) {
    System.out.println("Go home");
  } else {
    System.out.println("Go to school");
  }
  ```

- If “inchesOfSnow” > 7, the first println statement is executed and the second is skipped.
- Otherwise (i.e. inchesOfSnow ≤ 7), the first println statement is skipped and the second is executed.
Indentation Convention for if-else

The if-else class of statements should have the following form:

- if (condition) {
  
  statements;

} else {

  statements;

}
Blocks

- What happens?
  
  ```
  if (i > 10)
    i = 10;
    saturate = true;
  ```

- Desired: both \( i \), \( saturate \) are set only when \( i > 10 \)
- Actual: only the \( i=10 \) statement is dependant
  - Only one statement can be associated with `if`
  - The `saturate` assignment statement is not part of the `if`

- Blocks solve this problem
Blocks

- What happens?
  
  ```
  if (i > 10)
      i = 10;
      saturate = true;
  else
      k = 100;
  ```

- Desired: both `i`, `saturate` are set only when `i > 10`
- Actual: syntax error
  - Only one statement can be associated with `if`
  - The `saturate` assignment statement is not part of the `if`
  - The else can’t find the if it belongs to
- Blocks solve this problem also
What Blocks Are

- Blocks are sequences of statements “glued together” into one
- Form:

\[
\begin{align*}
&\{ \\
&\quad \text{<statement 1>;} \\
&\quad \text{<statement 2>;} \\
&\quad \ldots
\end{align*}
\]

- Example:

\[
\begin{align*}
\text{if (i > 10) } & \{ \\
\quad & \text{i = 10;} \\
\quad & \text{saturate = true;} \\
\text{else } & \{ \\
\quad & \text{i = i+1;}
\end{align*}
\]
Indention Conventions for Blocks

• Either
  
  ```
  if (...) {
    statement 1;
    statement 2;
    ...
  }
  ```

• Or
  
  ```
  if (...) {
    statement 1;
    statement 2;
    ...
  }
  ```

  This is what we will use (Sun code convention)
See Sun Code Conventions on Resource Page!

Java and White Space

- You can add:
  - carriage returns
  - spaces
  - tabs

  wherever you want in Java

- Properly used, this makes your program easier to read and understand
public class Example7 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int i = sc.nextInt();

        if (i < 0) {
            System.out.println("That was a negative number!");
        } else {
            System.out.println("That was a non-negative number!");
        }
        System.out.println("The number was: " + i);
    }
}
Logical Operators

Used for forming more complex conditions.

- “and”  
  
  ```java
  if ( temp >= 97 && temp <= 99 ) {
    System.out.println( "Patient is healthy" );
  }
  ```

- “or”  
  
  ```java
  if ( months >= 3 || miles >= 3000 ) {
    System.out.println( "Change your oil" );
  }
  ```

- “not”:  
  
  ```java
  if ( ! phone.equals( "301-555-1212" ) ) {
    System.out.println( "Sorry, wrong number" );
  }
  ```
Example 8

```
public class Example8 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int i = sc.nextInt();

        if (i < 0) {
            System.out.println("That was a negative number!");
            System.out.println("I prefer positive ones, so I'll fix it...");
            i = -i;
        } else {
            System.out.println("That was a positive number!");
            System.out.println("That makes me happy.");
        }
        System.out.println("The number is now " + i);
    }
}
```
Example 9

```java
public class Example9 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter an integer from 1 to 10: ");
        int i = sc.nextInt();

        if (i >= 1 && i <= 10) {
            System.out.println("Good job!");
        } else {
            System.out.println("You didn't follow instructions!");
        }
    }
}
```
Statement Constructors and Nesting

- if, if-else, {...} are *statement constructors*
  - They take statement(s) and convert them into a new statement
  - Example:
    ```java
    if (i >= 1 && i <= 10) {
        System.out.println("Good job!");
    } else {
        System.out.println("Oops!");
    }
    ```
  - Two “sub-statements” come in
  - A single big statement (if … else …) comes out
  - Implications: if statements, etc. can also appear inside (“be nested within”) one another
Java, Eclipse and Uninitialized Variables

- Eclipse will complain if you try to use an uninitialized variable:
  ```java
  int i;
  System.out.println("i is "+i);
  What is value of i?
  ```
- This feature interacts strangely with if/else statements sometimes
- Good programming practice: always initialize new variables