Boolean Expressions & Selection Structures (`if && if/else`)
Norton CS139

- Relational Operators
    - A Boolean expression in Java is an expression that returns either true or false. Boolean expressions use Java's *relational operators*.

`==`	equal to
`!=`	not equal to
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to

- Note the difference between the equality operator (`==`) and the assignment operator (`=`)
- The result of a Boolean expression can be assigned to a `boolean` variable:

```
int a;
int b;
boolean isGreaterThan = a > b;
```

- A Boolean expression can also be used to return a `boolean` value from a method:

```
public boolean isGreaterThan()
{
      return a > b;
}
```

- The method, then can be used in place of a Boolean expression or a boolean variable:

```
boolean myBool = isGreaterThan();
```

- Comparing Things (Primitive Types)
    - We can use the relational operators on integer types and character data
    - When comparing **chars**, the results are based on the Unicode character set
    - The following expression is true because the character '+' comes before the character 'J' in Unicode:

      ```
      boolean equalTo = '+' < 'J‘;
      ```

    - The uppercase alphabet (A-Z) and the lowercase alphabet (a-z) both appear in alphabetical order in Unicode
    - We have to be careful, though, when comparing two floating point values (**float** or **double**) for equality
        - You should rarely use the equality operator (==) when comparing two floats
        - In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal
        - Therefore, to determine the equality of two floats, you may want to use the following technique:

          ```
          boolean floatsAreEqual =
               Math.abs( f1 - f2 ) < 0.00001;
          ```

- Comparing Things (Strings & Other Objects)
    - Objects cannot be compared using the relational operators (why?)
    - Since character strings in Java are objects, we cannot use the relational operators to compare their contents
    - The **equals()** method can be called on a string to determine if two strings contain exactly the same characters in the same order
    - The String class also contains a method called **compareTo()** to determine if one string comes before another alphabetically (as determined by the Unicode character set)

      ```
      String myString =     “Hello”;
      String yourString =   “Hello”;

      boolean stringsAreEqual =
           myString.equals( yourString );

      int comparison = myString.compareTo( yourString );
      ```

- What happens if we use relational operators with Objects?

  ```
  String stringOne = "Hello";
  String stringTwo = "Hello";

  boolean oops = ( StringOne == stringTwo );
  ```

  Hint: what is actually stored in `stringOne` and `stringTwo`?

- Logical Operators
  - Boolean expressions can also use the following logical operators:

`!`	Logical NOT
`&&`	Logical AND
`||`	Logical OR

  - They all take `boolean` operands and produce `boolean` results
  - Logical NOT is a unary operator (it has one operand), but logical AND and logical OR are binary operators (they each have two operands)
  - The logical NOT operation is also called logical negation or logical complement
    - If some `boolean` condition `a` is true, then `!a` is `false`; if `a` is false, then `!a` is `true`
  - The logical AND expression:
    - `a && b`
      - is true if both `a` and `b` are `true`, and `false` otherwise
  - The logical OR expression:
    - `a || b`
      - is true if either a or b (or both) are true, and false otherwise

- Truth Tables
  - A truth table shows the possible true/false combinations for the logical AND and logical OR expressions:
  - Since `&&` and `||` each have two operands, there are four possible combinations of true and false

`a`	`b`	`a && b`	`a || b`
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

  - Since the logical NOT operator has only a single operand, its truth table has only two combinations of true and false

`a`	`!a`
true	false
false	true

- Complex logical statements
    - Logical operators can be used to join Boolean expressions to form complex expressions

      ```
      boolean complexBoolean = total < MAX && !found;
      ```

    - Logical operators have precedence relationships between themselves and other operators
        - Arithmetic operators have higher precedence than relational operators

          ```
          total != stock + warehouse
          ```

            - The addition will be evaluated first!
        - Relational operators have higher precedence than logical operators
        - Logical operators have precedence relationships among themselves

`!`	- highest
`&&`	
`||`	- lowest

        - Parentheses can be used to alter the normal precedence:

          ```
          ( a || b ) && c
          ```

    - Specific expressions can be evaluated using truth tables:

total < MAX	found	!found	total < MAX && !found
false	false	true	false
false	true	false	false
true	false	true	true
true	true	false	false

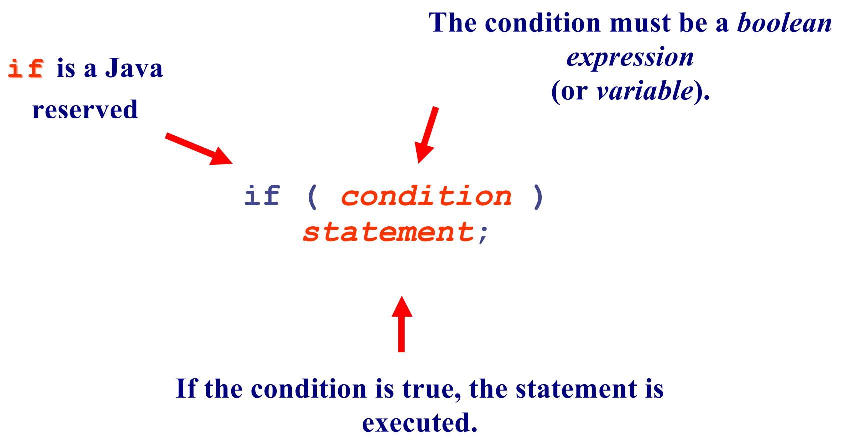
- Complex Boolean expressions are short circuited:

  `( x != 0 ) && ( y / x > 5 )` – ***whew!***
  `( y / x > 5 ) && ( x! = 0 )` – ***oops!***

- Selection Structures
  - Unless indicated otherwise, the order of statement execution through a method is linear:
    - one after the other in the order they are written
    - We call these "Sequential Structures" (or "Linear Structures")
  - Some programming statements modify that order, allowing us to:
    - decide whether or not to execute a particular statement,

      or
    - perform a statement over and over repetitively
  - The order of statement execution is called the flow of control
  - A *selection statement* (or conditional statement) lets us choose which statement will be executed next
  - Selection statements give us the power to make basic decisions
  - Java provides 3 selection statements:
    - the `if` *statement*,
    - he `if-else` *statement*, and
    - the `switch` *statement*

- The `if` statement
  - The `if` *statement* has the following syntax:


The condition must be a *boolean expression* (or *variable*).
`if` is a Java reserved
If the condition is true, the statement is executed.

```
if ( condition )
   statement;
```

- An example of an if statement:

  ```
  if ( sum > MAX )
       delta = sum - MAX;
  System.out.println ( "The sum is " + sum );
  ```

  1. First the condition is evaluated -- the value of sum is either greater than the value of MAX, or it is not
  2. If the condition is true, the assignment statement is executed -- if it isn't, it is skipped.
  3. Either way, the call to `println` is executed next

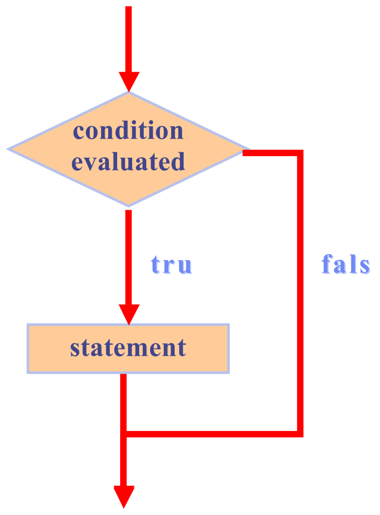
- See `Age.java`

- Indentation
  - The statement controlled by the if statement is indented to indicate that relationship
  - The use of a consistent indentation style makes a program easier to read and understand
  - Although it makes no difference to the compiler, proper indentation is crucial

  "Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live."

  Martin Golding

- Logic of an if statement

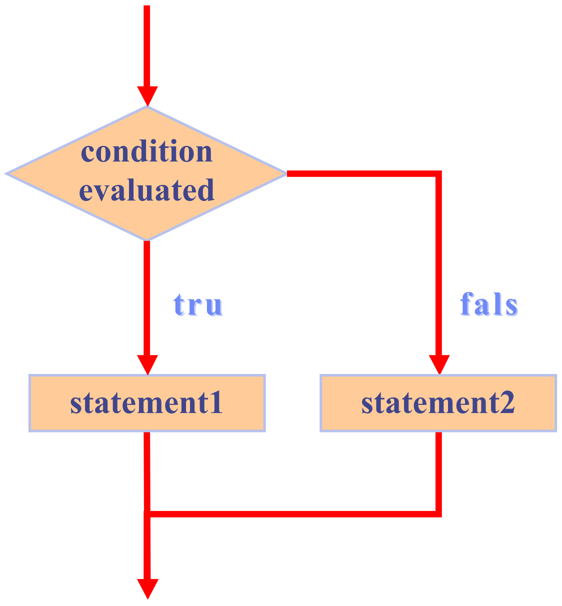


- The if/else statement
  - An *else clause* can be added to an if statement to make it an *if-else statement*:

    ```
    if ( condition )
          statement1;
    else
          statement2;
    ```

  - If the *condition* is true, *statement1* is executed; if the condition is false, *statement2* is executed
  - One or the other will be executed, but not both
  - See `Wages.java`

- Logic of an if/else statement



- The `Coin` class
  - Let's examine a class that represents a coin that can be flipped
  - Instance data is used to indicate which face (heads or tails) is currently showing
    - See `CoinFlip.java`
    - See `Coin.java`

- Complex `if/else` statements
  - Several statements can be grouped together into a block statement
    - A block is delimited by braces (`{ … }`)
    - A block statement can be used wherever a statement is called for in the Java syntax
    - For example, in an `if-else` statement, the `if` portion, or the `else` portion, or both, could be block statements

```
if ( someCondition )
{
	int temp;
	temp = 6;
	doSomething( temp );
	doSomethingElse( temp );
}
else
{
	int temp;
	temp = 3;
	doSomethingTotallyDifferent( temp );
}
```

    - See `Guessing.java`
  - The statement executed as a result of an `if` statement or `else` clause could be another `if` statement
    - These are called nested if statements
    - See `MinOfThree.java`
    - An `else` clause is matched to the last unmatched `if` (no matter what the indentation implies)

```
if ( someCondition )
	if ( someOtherCondition )
		doStuff();
	else
		doOtherStuff();
else
	doStillOtherStuff();
```