Flow of Control:
Selection

•••

Lecture 2

Dylan Boliske

Boolean Data Type

- The type **boolean** is a primitive data type consisting of just two values, the constants true and false.

```
boolean t = true;

boolean f = false;
```

Relational Operators

- Relational Operators are written in the following form:

  Expression 1 Operator Expression 2

Examples

- temperature > 60
- B*B - 4.0*A*C >= 0.0
- one + two < 0
- two*three <= four
- number == 35
- input != ‘Q’

Exercise

Given int x = 4; int y = 6;

- x < y
- x + 2 < y
- x != y
- x + 3 >= y
- y == x
- y == x + 2

Logical Operators

Java Expression	Logical Expression	Description
!p	NOT p	!p is false if p is true. !p is true if p is false.
p && q	p AND q	p && q is true if both p and q are true. It is false otherwise.
p || q	p OR q	p || q is true if either p or qu or both are true. It is false otherwise.

Exercise

Given int age = 20; double temperature = 102.0; boolean isSenior = (age >= 65); boolean hasFever = (temperature > 98.6);

- isSenior
- hasFever
- isSenior && hasFever
- isSenior || hasFever
- !isSenior
- !hasFever
- hasFever && (age > 50)

“Short-Circuit” Evaluation

- Java uses short circuit evaluation of logical expressions with operators !, &&, or ||.
- This means that logical expressions are evaluated left to right and evaluation stops as soon as the final truth value can be determined.

Example 1

int age = 25; int height = 70;

(age > 50) && (height > 60)

false

Evaluation can stop now because the result of && is only true if both sides are true; it has already determined that the entire expression will be false.

Example 2

int age = 25; int height = 70;

(height > 60) || (age > 40)

true

Evaluation can stop because || is true if at least one side is true; it has already determined the entire expression will be true.

Exercise

Write an expression for each of the following:

- taxRate is over 25% and income is less than $20000
- temperature is less than or equal to 75 or humidity is less than 70%
- age is over 21 and age is less than 60
- age is 21 or 22

Precedence Chart

Precedence	Type of Operation
Higher	Arithmetic
	Relational
Lower	Logical

Short-Circuit Benefits

- One boolean expression can be placed first to “guard” a potentially unsafe operation in the second boolean expression.
- Time is saved in evaluation of complex expressions using operators || and &&.

Example

Compare the following two expressions:

1. number != 0 && x < 1/number
2. (number != 0) && (x < 1/number)

If-else (else is optional)

```
if (boolean expression) {
    //if branch
} else {
    //else branch
}
```

Else-if statements

```
if (boolean expression) {
      //statement 1
} else if (another boolean expression) {
      //statement 2
} …
} else if (final boolean expression) {
      //statement N
} else {
      //statement N+1
}
```

Exactly one of these statements will be executed.

Else-if statement

- Each expression is evaluated in sequence until some expression is found to be true.
- Only the specific statement following the true expression will be executed.
- If no expression is true, then the statement following the final else will be evaluated.
- The final else and final statement are optional. If omitted and no expression is true, then no statement is executed.

Comparing Reference Data Types

- The equator operator (==) compares object references. This is NOT the same as comparing their values.
- Example:
  - If *str1* and *str2* are two String object references, then

    ```
    (str1 == str2)
    ```

    evaluates to true only if *str1* and *str2* point to the same object, that is, the same location in memory.

Comparing Reference Data Types (values)

- To compare the value of a reference data type, use the `equals` method:

Return Type	Method Description
boolean	equals(Object obj) returns true if the data in the object *obj* is equal to the data in the object used to call the method

- Example (with *str1* and *str2* as Strings):

```
str1.equals(str2)
```

Returns true if the sequence of characters are the same in *str1* and *str2*.

equalsIgnoreCase method

- Sometimes, for Strings, it is not necessary that the two be completely identical. In these cases, it is better to use the `equalsIgnoreCase` method to compare the two.

Return Type	Method Description
boolean	equalsIgnoreCase(String str) compares the values of two Strings, treating upper and lower case characters as equal. Returns true if the Strings are equal, false otherwise.

compareTo method

- Sometimes, for Strings, finding out which comes first alphabetically is needed. For situations like these, the compareTo method can be used.

Return Type	Method Description
int	compareTo(String str) compares the values of two Strings. If the String object is less than the String arguement, *str*, a negative integer is returned. If the String object is greater than *str*, a positive integer is returned. If the two are equal, then 0 is returned.

- A character with a lower Unicode numeric value is considered less than a character with a higher Unicode numeric value.
- *a* is less than *b* and *A* is less than *a*.

Conditional (Ternary) Operator

- The Conditional Operator is written in the following form:

  condition ? trueExpression : falseExpression

- If condition evaluates to true, then the value of the entire expression is trueExpression. Otherwise, the value is falseExpression.

Example

```
double min;

double x;

double y;

…

min = (x < y) ? x : y;
```

Switch Statements

- Switch statements are a selection control structure for multi-way branching

```
switch (expression) {
    case constant1 : statements; //optional
    case constant2 : statements; //optional
    …
    default : statements;        //optional (but a good idea)
}
```

Switch Statements

- The value of the `expression` should be an Integral data type (byte, int, char, etc.). However, in newer versions of Java, Strings have also become acceptable.
- Case labels are constant expressions; several cases can precede a single statement.
- Control branches to the statement following the case label that matches the value of the `expression`.
- Control proceeds through all following statements unless redirected with a `break`.
- If no case label matches the value of `expression`, control branches to the `default` label, if it exists; otherwise, control leaves the switch structure.
- Forgetting to use `break` can cause logical errors because after a branch is taken, control proceeds sequentially until either it hits a break or the end of the switch structure occurs.