Conditionals

BOOLEAN EXPRESSIONS

Boolean Expressions

- A *boolean expression* evaluates to true or false.
- Boolean expressions often use an equality or relational operator to compare two values:

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

equality operators

relational operators

Boolean Expressions (cont.)

- Arithmetic operators have higher precedence than equality or relational operators.
- Examples:

7 < 9 – 4	evaluates to false
6 == 4 + 2	evaluates to true

Boolean Expressions (cont.)

- Boolean expressions can be combined using logical operators

!	NOT
&&	AND
||	OR

- Each operator takes two boolean expressions and returns a boolean result.

Logical NOT

- Unary operator (takes one operand)
- If boolean `b` is true, `!b` is false
- If boolean `b` is false, `!b` is true

`a`	`!a`
true	**false**
false	**true**

Logical AND and OR

- Binary operators
- If boolean `a` and `b` are **both** true, `a&&b` is true
- If **either** boolean `a` or `b` is true, `a||b` is true

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

Short-Circuited Operators

- The processing of AND and OR is *short circuited* meaning that if the left operand is enough to determine the result, the right operand is never evaluated.

```
FALSE && b
// b is never evaluated

TRUE || b
// b is never evaluated
```

Precedence Revisited

1. Parentheses
2. Multiplication, Division, Remainder (left to right)
3. Addition, Subtraction, Concatenation (left to right)
4. Equality and Relational Operators

   `==   !=   <   >   <=   >=`
5. Logical NOT

   `!`
6. Logical AND and OR (left to right)

   `&&   ||`
7. Assignment

   `=   +=   -=   /=   *=`

Before going to bed, a programmer put two glasses on his nightstand: one filled with water and one empty. Her husband asked her "why?" and she said, “The water is there in case I wake up at night and I feel thirsty.” “What about the other one?” “Oh! This is in case I wake up at night and I am *not* thirsty!”

IF/ELSE-IF/ELSE STATEMENTS

Flow of Control

- The order of statement execution is called the *flow of control.*
- Unless otherwise specified, programs execute in a linear fashion
  - Statements are executed one after another in sequence.
- Several things change this…
  - method invokations make control jump to another point in the program.
  - conditionals also change the flow of control by taking different paths based on different conditions.

Conditional Statements

- A *conditional statement* allows us to take different actions under different conditions.
- The simplest form is an if-statement.
  - If ***boolean*** is true, the code will `do something`.
  - If ***boolean*** is not true, nothing happens.

```
if (boolean) {
    // do something
}
```

Indentation and Brackets

- Without brackets, one single statement after an if-statement will execute.
- With brackets, all statements inside will execute.
  - This is called a *block statement*.
- With or without brackets, best practice is to indent code within an if-statement.
  - Java doesn’t care about this, but human readers do!

The if-else Statement

- An else clause can be added to an if.
  - If ***boolean*** is true, `statement1` is executed.
  - If ***boolean*** is false, `statement2` is executed.
- One or the other statements is executed, but not both.

```
if (boolean) {
    // statement1
} else {
    // statement2
}
```

After the Conditional…

- Once the conditional is done, you proceed to the first line of code after the full conditional.

```
if (boolean) {
   // do something- then go to below
} else {
   // do something different- then go to below
}
// continue here! (whether you executed the
  code in the if or in the else)
```

The if-else Statement (cont.)

- Just like the if-statement, without brackets, one single line of code is execute.
- With brackets, all code inside the brackets will execute.

Block Statements

- Variables declared inside of a block statement are local to that statement only.
  - They cannot be seen outside the brackets.

```
if (total > MAX)
{
   boolean error = true;
}
if(error)
   System.out.println("Error");
// COMPILER ERROR
```

Examples

- What is printed?

```
if(7.0 > -6.2) {
  System.out.println("yes");
} else {
  System.out.println("no");
}
```

Examples

- What is printed?

```
int number1 = 4;
number1 = number1 * 2;
double number2 = 8;

if(number1 != number2) {
   System.out.println("yes");
} else {
   System.out.println("no");
}
System.out.println(“moving on");
```

Examples

- What is printed?

```
int number1 = 5, number2 = 10;

if(number1 < 10)
    number1 = number1 * 3;

if(number2 > 10)
    number2 = number2 – 5;

if(number1 < 10 || number2 > 10) {
    System.out.println("yes");
} else {
    System.out.println("no");
}
System.out.println(“moving on");
```

Examples

- What is printed?

```
int number1 = 1, number2 = 10;

if(number1 < 10)
    number1 = number1 * 3;

if(number2 > 10)
    number2 = number2 - 5;

if(number1 < 10 || number2 > 10) {
    System.out.println("yes");
} else {
    System.out.println("no");
}
```

Examples

- What is printed?

```java
int number1 = 1, number2 = 10;

if(number1 < 10)
    number1 = number1 * 3;

if(number2 > 10)
    number2 = number2 - 5;

if(number1 < 10 && number2 > 10) {
    System.out.println("yes");
} else {
    System.out.println("no");
}
```

The if-else-if Statement

- You can add multiple conditions with else-if statements.
- Ending with a single else ensures that one of the statements is executed.

```
if (condition1) {
    // do something1
} else if (condition2) {
    // do something2
} else if (condition3) {
    // do something3
} else {
    // do something4
}
```

The if/else-if

```
if (condition1) {
    // do something1
} else if (condition2) {
    // do something2
} else if (condition3) {
    // do something3
} else {
    // do something4
}
// moving on
```

check if condition1 is true

if it is, do something1

then move on to the code after the **entire** conditional

note: if condition1 is true, we never even look at condition2, condition3, or the else!

The if/else-if

```
if (condition1) {
    // do something1
} else if (condition2) {
    // do something2
} else if (condition3) {
    // do something3
} else {
    // do something4
}
// moving on
```

if condition1 is **not** true, then check condition2

if it's true, do something2

then move on to the code after the **entire** conditional

note: if condition2 is true, we never even look at condition3, or the else!

The if/else-if

```
if (condition1) {
    // do something1
} else if (condition2) {
    // do something2
} else if (condition3) {
    // do something3
} else {
    // do something4
}
// moving on
```

if condition2 is **not** true, then check condition3

if it's true, do something3

then move on to the code after the **entire** conditional

note: if condition2 is true, we never even look at condition3, or the else!

The if/else-if

```
if (condition1) {
   // do something1
} else if (condition2) {
   // do something2
} else if (condition3) {
   // do something3
} else {
   // do something4
}
// moving on
```

if condition3 is **not** true, then do something4 (because it's an else, we don't check any other conditions)

then move on to the code after the **entire** conditional

The if/else-if

```
if (condition1) {
   // condition1 is true
} else if (condition2) {
  // condition1 is false
  // condition2 is true
} else if (condition3) {
  // condition1 is false
  // condition2 is false
  // condition3 is true
} else {
  // condition1 is false
  // condition2 is false
  // condition3 is false
}
```

Conditional Rules- What is Allowed

- a single if with
  - no else
- a single if with
  - a single else
- a single if with
  - any number of else-ifs
  - a single else
- a single if
  - any number of else-ifs
  - no else

- Conditionals are matched based on their *brackets*.
  - But indentation helps humans, too!

Nested Conditionals

- If statements can be nested.
- Match the conditionals based on the brackets.
  - But indentation helps humans, too!
- If brackets are not used, an else clause is matched to the last unmatched if statement.
  - Don’t do this!

Nested Conditionals

- Conditionals can be “nested” inside each other.
- The rules are the same.
  – Just treat each conditional on its own as you trace through.

Nested Conditionals (cont.)

```
if (c1) {
   // c1 is true
   if (c2) {
          // c1 and c2 are true
          if (c3) {
               // c1, c2, and c3 are true
          } else if (c4) {
               // c1 and c2 are true; c3 is false; c4 is true
          } else if (c5) {
               // c1 and c2 are true; c3 and c4 are false; c5 is true
          } else {
               // c1 and c2 are true; c3, c4, and c5 are false
          }
          // c1 and c2 are true; c3 is false; c4 is true
   } else {
   // c1 is true and c2 is false
   }
} // the c1-if has no else
```

Practice

- Write a program to determine the smallest of three values entered by a user.
- Write an interactive program that allows the user to perform basic mathematical functions including sum, difference, product, division, square, and square root.

CONDITIONAL (TERNARY) OPERATOR

The Conditional Operator

- A way to rewrite an if-else in a single statement

```
condition ?  expression1  :  expression2 ;
```

- If condition is true, expression1 is evaluated.
- If condition is false, expression2 is evaluated.

Example

```
int larger =
   num1 > num2  ?  num1  :  num2;
```

this is equivalent to:

```
int larger;
if (num1 > num2)
   larger = num1;
else
   larger = num2;
```

Example

```
System.out.println(“Your change is ” +
  count +
  ( ( count ==1 ) ? “Dime” : “Dimes” ) );
```

THE SWITCH STATEMENT

The switch Statement

- The switch statement evaluates an expression and tries to match the result to a set of possible cases.
- Each case contains a value and the statements to execute if the value matches.

The switch Statement (cont.)

```
switch (expression) {
  case value1 :
     statement-list1
  case value2 :
     statement-list2
  case value3 :
     statement-list3
}
```

The Expression

- Can contain:
  - int or Integer
  - char or Character
  - byte of Byte
  - short or Short
  - enumerated type
  - String (from Java 1.7 on)
- Is compared to each value for equality

The break Statement

- A *break statement* can be used as the last statement in each case’s statement list.
- A break causes control to jump to the end of the switch.
- Without a break, once a case is matched, *all statements* after the match will be executed.
  - Even if they are within a case that doesn’t match.
- Most times you want a break in each case.
  - Sometimes you don’t!

The default Statement

- You can optionally include a default case that has no associated value.
- If you use a default case, control will transfer to that case if no match is found.
- If you don’t use a default case and there I no match, control jumps to the statement after the switch.

Example

```
switch (grade) {
    case ‘A’ :
          aCount++;
          break;
    case ‘B’ :
          bCount++;
          break;
    case ‘C’ :
          cCount++;
          break;
    default :
          dCount++;
          break;
}
```

Practice

- Write a program to output the number of days in the month based on the month number.
  - First use an if/else-if/else.
  - Then use a switch.