Lecture 10
boolean Data Type

- is a built-in (primitive) data type of Java
- is used to represent the logical values
- There are 2 logical values:
  - true
  - false

Encoding scheme used in the boolean data type:
- 0 represents false
- 1 represents true

- Boolean literals
  - There are 2 boolean literals (= logical constants) in Java:
    - true
    - false

These 2 words are keywords (reserved words) in Java.
Operations that return a *boolean* result

- There are 2 types of *operators* that return a *boolean* result (true or false):

  - **Compare operators:**
    - A *compare operator* compares 2 *numerical expressions* and *return* a *Boolean result*.
    - Example:
      - the *compare operation* `3 > 1` returns the *Boolean value* `true`

  - **Logical operators:**
    - A *logical operator* compares 2 *Boolean (logical) expressions* and *return* a *Boolean result*.
    - Example:
      - the *logical operation* `true AND false` returns the *Boolean value* `false`
- **Compare operators**
  - Compare operators
    - Compare operators compare 2 numerical values and return a Boolean (logical) value
    - A compare operator will return the value true if the test is successful
    - A compare operator will return the value false if the test is unsuccessful

- Compare operators in Java:

<table>
<thead>
<tr>
<th>Operator symbol</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>a &lt; b</td>
<td>Returns true if a &lt; b, otherwise returns false</td>
</tr>
<tr>
<td>&lt;=</td>
<td>a &lt;= b</td>
<td>Returns true if a ≤ b, otherwise returns false</td>
</tr>
<tr>
<td>&gt;</td>
<td>a &gt; b</td>
<td>Returns true if a &gt; b, otherwise returns false</td>
</tr>
<tr>
<td>&gt;=</td>
<td>a &gt;= b</td>
<td>Returns true if a ≥ b, otherwise returns false</td>
</tr>
<tr>
<td>==</td>
<td>a == b</td>
<td>Returns true if a is equal to b, otherwise returns false</td>
</tr>
<tr>
<td>!=</td>
<td>a != b</td>
<td>Returns true if a is not equal to b, otherwise returns false</td>
</tr>
</tbody>
</table>
public class Boolean01
{
    public static void main(String[] args)
    {
        boolean a;               // boolean typed variable
        a = true;
        System.out.println(a);   // prints true

        a = false;
        System.out.println(a);   // prints false

        a = (3 > 1);             // 3 > 1 evals to true, so: a = true
        System.out.println(a);   // prints true

        a = (3 <= 1);            // 3 <= 1 evals to false, so: a = false
        System.out.println(a);   // prints false
    }  
}
- Priority ranking of the **compare operators** against the **previously discussed operators**:

<table>
<thead>
<tr>
<th>Priority level</th>
<th>Operator(s)</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( )</td>
<td>Brackets</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(int) –</td>
<td>Casting, negation</td>
<td>Right to left</td>
</tr>
<tr>
<td>3</td>
<td>++, --</td>
<td>Increment, decrement</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>* / %</td>
<td>Multiple, divide, remainder</td>
<td>Left to right</td>
</tr>
<tr>
<td>5</td>
<td>+ –</td>
<td>Add, subtract</td>
<td>Left to right</td>
</tr>
<tr>
<td>6</td>
<td>&lt; &lt;= &gt; &gt;= == != !</td>
<td><strong>Compare operators</strong></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>= += -= ...</td>
<td>Assignment operators</td>
<td>Right to left</td>
</tr>
</tbody>
</table>
if (boolean-expression) {
    statement(s);
}

if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " +
                     radius + " is " + area);
}
The block braces can be omitted if they enclose a single statement. For example, the following statements are equivalent.

```
if (i > 0) {
    System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
    System.out.println("i is positive");
}
```

(b) Correct
Two-Way if-else Statements

```java
if (boolean-expression) {
    statement(s)-for-the-true-case;
}
else {
    statement(s)-for-the-false-case;
}
```
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " +
        radius + " is " + area);
} else {
    System.out.println("Negative input");
}

if (number % 2 == 0)
    System.out.println(number + " is even.");
else
    System.out.println(number + " is odd.");

if (number % 2 == 0)
    System.out.println(number + " is even.");
else
    System.out.println(number + " is odd.");
Nested **if** and Multi-Way **if-else** Statements

```plaintext
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

Equivalent:

```plaintext
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

This is better.
Common Errors in Selection Statements

- **Common Error 1: Forgetting Necessary Braces**

```java
if (radius >= 0)
    area = radius * radius * PI;
System.out.println("The area " + " is " + area);
```

(a) Wrong

```java
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area " + " is " + area);
}
```

(b) Correct

- **Common Error 2: Wrong Semicolon at the if Line**

```java
if (radius >= 0);
{
    area = radius * radius * PI;
    System.out.println("The area " + " is " + area);
}
```

(a)

```java
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area " + " is " + area);
}
```

(b) Equivalent

- Logic error

- Empty block
• Common Error 3: Redundant Testing of Boolean Values

```java
if (even == true) {
    System.out.println("It is even.");
}
```

Equivalent:

```java
if (even) {
    System.out.println("It is even.");
}
```

This is better.

• Common Error 4: Dangling else Ambiguity

```java
int i = 1, j = 2, k = 3;
if (i > j) {
    if (i > k) {
        System.out.println("A");
    } else {
        System.out.println("B");
    }
}
```

Equivalent:

```java
int i = 1, j = 2, k = 3;
if (i > j) {
    if (i > k) {
        System.out.println("A");
    } else {
        System.out.println("B");
    }
}
```

This is better with correct indentation.

```java
int i = 1, j = 2, k = 3;
if (i > j) {
    if (i > k) {
        System.out.println("A");
    } else {
        System.out.println("B");
    }
}
```
Tips

I *always* write if-statements and if-else-statements using **blocks** first:

```java
if ( ...... )
{
   (leave empty first)
}
```

I fill in the statements in the **then-part** and **else-part** later.

So *even when* the if-part and/or else-part consist of **1 statement**, I write them as **blocks**.
Generating Random Numbers

• Use the `random()` method in the `Math` class. Invoking this method returns a random double value \( d \) such that \( 0.0 \leq d < 1.0 \). Thus, `(int)(Math.random() * 10)` returns a random single-digit integer (i.e., a number between 0 and 9).
Logical Operators

<table>
<thead>
<tr>
<th>Operator symbol</th>
<th>Meaning</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>The logical AND operator</td>
<td>Binary operator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>The logical NOT operator</td>
<td>Unary operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>true &amp;&amp; true</td>
<td>true</td>
</tr>
<tr>
<td>true &amp;&amp; false</td>
<td>false</td>
</tr>
<tr>
<td>false &amp;&amp; true</td>
<td>false</td>
</tr>
<tr>
<td>false &amp;&amp; false</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( p )</th>
<th>!p</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Priority and associativity of Java's operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Level</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>access array element</td>
<td>1</td>
<td>left to right</td>
</tr>
<tr>
<td>.</td>
<td>access object member</td>
<td>1</td>
<td>left to right</td>
</tr>
<tr>
<td>( )</td>
<td>invoke a method</td>
<td>1</td>
<td>left to right</td>
</tr>
<tr>
<td>++</td>
<td>post-increment</td>
<td>1</td>
<td>left to right</td>
</tr>
<tr>
<td>--</td>
<td>post-decrement</td>
<td>1</td>
<td>left to right</td>
</tr>
<tr>
<td>++</td>
<td>pre-increment</td>
<td>2</td>
<td>right to left</td>
</tr>
<tr>
<td>--</td>
<td>pre-decrement</td>
<td>2</td>
<td>right to left</td>
</tr>
<tr>
<td>+</td>
<td>unary plus</td>
<td>2</td>
<td>right to left</td>
</tr>
<tr>
<td>-</td>
<td>unary minus</td>
<td>2</td>
<td>right to left</td>
</tr>
<tr>
<td>!</td>
<td>logical NOT</td>
<td>2</td>
<td>right to left</td>
</tr>
<tr>
<td>~</td>
<td>bitwise NOT</td>
<td>2</td>
<td>right to left</td>
</tr>
<tr>
<td>()</td>
<td>cast</td>
<td>3</td>
<td>right to left</td>
</tr>
<tr>
<td>new</td>
<td>object creation</td>
<td>3</td>
<td>right to left</td>
</tr>
<tr>
<td>*</td>
<td>multiplicative</td>
<td>4</td>
<td>left to right</td>
</tr>
<tr>
<td>/</td>
<td></td>
<td>4</td>
<td>left to right</td>
</tr>
<tr>
<td>%</td>
<td></td>
<td>4</td>
<td>left to right</td>
</tr>
<tr>
<td>+</td>
<td>additive</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>+</td>
<td>string concatenation</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>shift</td>
<td>6</td>
<td>left to right</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td></td>
<td>6</td>
<td>left to right</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td></td>
<td>6</td>
<td>left to right</td>
</tr>
</tbody>
</table>

Relational operators:

- `<  <=  >  >=`
- `instanceof`

Equality operators:

- `==`
- `!=`

Bitwise operators:

- `&`
- `^`
- `|`

Conditional operators:

- `?:`

Assignment operators:

- `=  +=  -=  *=  /=  %=  &=  ^=  |=  <<=  >>=  >>>=  >>>>>=`
Leap year

- In the Gregorian calendar, the current standard calendar in most of the world, **most years** that are evenly divisible by 4 are leap years.
- Years that are *evenly divisible by 100* are **not leap years**, unless they are also *evenly divisible by 400*, in which case **they are leap years**.

- Use nested if-else.
- Use one logical expression.
Comparing floating point numbers on equality

The floating point number 4 is not exactly equal to 4

It can be: 4.0000000000000001
Or: 3.9999999999999999

- When are 2 values equal to each other:
  - Two values are equal if they are equal in all digits

Consequently:

4.0000000000000001 != 3.9999999999999999
public class FloatEq1
{
    public static void main(String[] args)
    {
        double a, b;
        int i;

        a = 4.0;

        b = 4.0/7.0 + 4.0/7.0 + 4.0/7.0 + 4.0/7.0 + 4.0/7.0 + 4.0/7.0 + 4.0/7.0;

        System.out.println("a = " + a);
        System.out.println("b = " + b);

        if ( a == b )
        {
            System.out.println("a is equal to b");
        }
        else
        {
            System.out.println("a is NOT equal to b");
        }
    }
}
However:

- In *most applications*, 4.0 is **equal** to **3.99999999999999**

Because in the level of error tolerance in *most applications* is much higher than **0.00000000000001** part in 4 !!!

**Testing equality** within a given tolerance

- **Programming trick:**

  - When we want to test if 2 values `a` and `b` are **approximately equal** to each other, we use this test:

    ```java
    if ( absoluteValue( b - a ) < some-very-small-value )
    {
        a and b are equal
    }
    else
    {
        a and b are not equal (too far apart)
    }
    ```

- To compute the **absolute value**, you can use the `Math.abs()` method.
Example:

```java
public class FloatEq2 {
    public static void main(String[] args) {
        double a, b;
        int i;

        a = 4.0;
        b = 4.0/7.0 + 4.0/7.0 + 4.0/7.0 + 4.0/7.0 + 4.0/7.0 + 4.0/7.0 + 4.0/7.0;

        System.out.println("a = " + a);
        System.out.println("b = " + b);

        if ( Math.abs( b - a ) < 0.000000001 )
        {
            System.out.println("a is (approximately) equal to b");
        }
        else
        {
            System.out.println("a is NOT (approximately) equal to b");
        }
    }
}
```

- In Physics and Chemistry, they work with **relative errors**

You can modify the test as follows:

```java
if ( Math.abs( b - a ) < ( 0.000000001 * (Math.abs(b) + Math.abs(a)) ) )
{
    ...
}
```