Lecture 9
Honor Code

- [http://www.mathcs.emory.edu/spca.php](http://www.mathcs.emory.edu/spca.php)
- Copy the following at the very front of all your programs.
- /* THIS CODE IS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING A TUTOR OR CODE WRITTEN BY OTHER STUDENTS - YOUR NAME */
Programming Style and Documentation

- Appropriate Comments and Comment Styles
  - line comments  // ......
  - block comments  /* ...... */

- Proper Indentation and Spacing
  - A single space should be added on both sides of a binary operator

```
System.out.println(3+4*4);  // Bad style
System.out.println(3 + 4 * 4);  // Good style
```
• Block Styles

  – A block is a group of statements surrounded by braces. There are two popular styles, next-line style and end-of-line style, as shown below.

```java
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

Next-line style

```java
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

End-of-line style
String data type

• The `char` type represents only one character. To represent a string of characters, use the data type called `String`.

```java
String message = "Welcome to Java";
```
We write **string literals** between **double quotes** "..." (as discussed above)

**Example:**

```
"Hello World"
```

**Escape character:**

- **Escape character** = a **special character** that allow **Java** to **change the meaning** of the **next character**
- \ (backslash) = the **escape character** for **strings**
  - The **escape character** is usually used to express "unprintable" **characters**

**Escape sequence:**

- **Escape sequence** = the **escape character** \ followed by **one character**

**Example:**

```
\n
```

- A **escape sequence** denotes **one character**

The **character** denoted by an **escape sequence** is usually one that you cannot type in with the keyboard
Commonly used *escape sequences*:

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Denoted character</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>\t</td>
<td>Tab</td>
<td>&quot;\t&quot; (string with a TAB character)</td>
</tr>
<tr>
<td>\n</td>
<td>New line (NL)</td>
<td>&quot;\n&quot; (string with a NL character)</td>
</tr>
<tr>
<td>\ \</td>
<td>Backslash ()</td>
<td>&quot;\&quot; (string with a )</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote (&quot;)</td>
<td>&quot;\&quot; (string with a &quot;)</td>
</tr>
</tbody>
</table>

Example:

```java
public class Escape01
{
    public static void main(String[] args)
    {
        System.out.println("He said \t "Hello\nI said \\\\\\\n    ");
    }
}
```
The + operation

```java
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB

message += " and Java is fun";
```

Java's automatic conversion rule for `number ⇒ string`:

- When the `+` operator is used between
  - a `number` and
  - a `string`

Then:

- the `number` is automatically converted to a `string`
- The `+` operator is then applied on 2 `strings` (i.e., the `+` operator is a concatenation!)
```java
public class String05
{
    public static void main(String[] args)
    {
        String a, b;
        int x;

        a = "abc";
        x = -12;

        b = a + x;
        System.out.println(b);   // Prints "abc-12"

        b = x + a;
        System.out.println(b);   // Prints "-12abc"
    }
}
```

```java
public class String06
{
    public static void main(String[] args)
    {
        System.out.println( "a" + 1 + 2 );
        System.out.println( 1 + 2 + "a" );
    }
}
```
How to convert: \( \text{int} \Rightarrow \text{String} \)

```java
String a;
int x;

a = Integer.toString(x);  // Returns the String representation
// for the integer value in variable x
```

How to convert: \( \text{double} \Rightarrow \text{String} \)

```java
String a;
double x;

a = Double.toString(x);  // Returns the String representation
// for the double value in variable x
```

Converting: \( \text{float} \Rightarrow \text{String} \)

```java
String a;
float x;

a = Float.toString(x);  // Returns the String representation
// for the float value in variable x
```
public class String04
{
    public static void main(String[] args)
    {
        String s1, s2; // Strings

        int x = 12;
double y = 3.1415;

        System.out.print("x + y = ");
System.out.println( x + y ); // Add numbers

        s1 = Integer.toString(x);
s2 = Double.toString(y);
System.out.print("s1 + s2 = ");
System.out.println( s1 + s2 ); // + on String is concatenation
    }
}
Converting: \texttt{int} \Leftrightarrow \texttt{String}

\begin{verbatim}
String s;
int x;

s = Integer.toString(x); // Returns the String representation
// for the integer value in variable x

x = Integer.parseInt(s); // Returns the binary number representation
// for the numeric string in variable s
\end{verbatim}

Converting: \texttt{double} \Leftrightarrow \texttt{String}

\begin{verbatim}
String s;
double x;

s = Double.toString(x); // Returns the String representation
// for the double value in variable x

x = Double.parseDouble(s); // Returns the IEEE 754 double
// precision number representation
// for the numeric string in variable s
\end{verbatim}

Converting: \texttt{float} \Leftrightarrow \texttt{String}

\begin{verbatim}
String a;
float x;

s = Float.toString(x); // Returns the String representation
// for the float value in variable x

x = Float.parseFloat(s); // Returns the IEEE 754 single
// precision number representation
// for the numeric string in variable s
\end{verbatim}
public class String04
{
    public static void main(String[] args)
    {
        String s1, s2;       // Strings

        int x = 12;
        double y = 3.1415;

        System.out.print("x + y = ");
        System.out.println( x + y );  // Add numbers

        s1 = Integer.toString(x);
        s2 = Double.toString(y);
        System.out.print("s1 + s2 = ");
        System.out.println( s1 + s2 );  // + on String is concatenation
    }
}
Obtaining the *length information* of strings in Java:

- Let `a` be a string typed variable
- The expression `a.length()` returns the length of the string stored in the string variable `a`

```java
class String02 {
    public static void main(String[] args) {
        String a;
        a = "abc"; // a = location + length of "abc"
        System.out.println(a); // Print the string at a
        System.out.println(a.length()); // Print the length of the string at a
    }
}
```
charAt(i): return the character at position \( i \) in the string.

Example:

```
String s = "Hello World";

char x;

x = s.charAt(0);  // x = 'H';
x = s.charAt(1);  // x = 'e';
x = s.charAt(2);  // x = 'l';
```
substring(i, j): return the substring consisting of the characters between positions i and j-1 in the string.

Example:

```java
String s = "Hello World";
char  x;

y = s.substring(0,1);  // y = "H"
y = s.substring(2,4);  // y = "ll"
```
Reading String from keyboard

To read a string from the console, invoke the `next()` method on a `Scanner` object. For example, the following code reads three strings from the keyboard:

```java
Scanner input = new Scanner(System.in);
System.out.println("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

The `next()` method reads a string that ends with a whitespace character. The characters ' ', \t, \f, \r, or \n are known as whitespace characters.

You can use the `nextLine()` method to read an entire line of text. The `nextLine()` method reads a string that ends with the `Enter` key pressed. For example, the following statements read a line of text.

```java
Scanner input = new Scanner(System.in);
System.out.println("Enter a line: ");
String s = input.nextLine();
System.out.println("The line entered is " + s);
```
**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`

- Uses 1 byte of memory (to store 0 or 1)

**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 <strong>true</strong> if ( a &lt; b ), otherwise returns <strong>false</strong></td>
</tr>
<tr>
<td>&lt;=</td>
<td>a &lt;= b</td>
<td>Returns <strong>true</strong> if ( a \leq b ), otherwise returns <strong>false</strong></td>
</tr>
<tr>
<td>&gt;</td>
<td>a &gt; b</td>
<td>Returns <strong>true</strong> if ( a &gt; b ), otherwise returns <strong>false</strong></td>
</tr>
<tr>
<td>&gt;=</td>
<td>a &gt;= b</td>
<td>Returns <strong>true</strong> if ( a \geq b ), otherwise returns <strong>false</strong></td>
</tr>
<tr>
<td>==</td>
<td>a == b</td>
<td>Returns <strong>true</strong> if ( a ) is equal to ( b ), otherwise returns <strong>false</strong></td>
</tr>
<tr>
<td>!=</td>
<td>a != b</td>
<td>Returns <strong>true</strong> if ( a ) is not equal to ( b ), otherwise returns <strong>false</strong></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>Compare operators</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>= += -= ...</td>
<td>Assignment operators</td>
<td>Right to left</td>
</tr>
</tbody>
</table>
if Statements

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

(a) Wrong

\[
\begin{array}{l}
\text{if (i > 0) \{}
\text{System.out.println("i is positive");}
\text{\}}
\end{array}
\]

(b) Correct

The block braces can be omitted if they enclose a single statement. For example, the following statements are equivalent.

\[
\begin{array}{l}
\text{if (i > 0) \{}
\text{System.out.println("i is positive");}
\text{\}}
\end{array}
\]

(a)

\[
\begin{array}{l}
\text{if (i > 0) \{}
\text{\}}
\end{array}
\]

(b) Equivalent
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.");