Selections, and Mathematical Functions, Characters, and Strings

Introduction to Programming and Computational Problem Solving - 2

CSE 8B

Lecture 3
Announcements

• Assignment 1 is due Oct 14, 11:59 PM
• Quiz 1 is Oct 16
• Assignment 2 will be released Oct 14
  – Due Oct 21, 11:59 PM
• Educational research study
  – Oct 16, weekly reflection
• Reading
  – Chapters 3 and 4
The boolean type and operators

• Often in a program you need to compare two values, such as whether \( i \) is greater than \( j \)
• Java provides six comparison operators (also known as relational operators) that can be used to compare two values
• The result of the comparison is a Boolean value: true or false
• For example
  ```java
  boolean b = (1 > 2);
  ```
## Relational operators

<table>
<thead>
<tr>
<th>Java Operator</th>
<th>Mathematics Symbol</th>
<th>Name</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>less than</td>
<td>radius &lt; 0</td>
<td>false</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
<td>less than or equal to</td>
<td>radius &lt;= 0</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>greater than</td>
<td>radius &gt; 0</td>
<td>true</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
<td>greater than or equal to</td>
<td>radius &gt;= 0</td>
<td>true</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
<td>equal to</td>
<td>radius == 0</td>
<td>false</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
<td>not equal to</td>
<td>radius != 0</td>
<td>true</td>
</tr>
</tbody>
</table>
if statements

```java
if (boolean-expression) {
    statement(s);
}
```
if statements

if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " + radius + " is " + area);
}

(area >= 0)
if-else statements

if (boolean-expression) {
    statement(s)-for-the-true-case;
}
else {
    statement(s)-for-the-false-case;
}
if (radius >= 0) {
    area = radius * radius * 3.14159;

    System.out.println("The area for the " + "circle of radius " + radius + " is " + area);
}
else {
    System.out.println("Negative input");
}
Multiple if-else statements

```java
if (score >= 90.0)
    System.out.print("A");
else
    if (score >= 80.0)
        System.out.print("B");
    else
        if (score >= 70.0)
            System.out.print("C");
        else
            if (score >= 60.0)
                System.out.print("D");
            else
                System.out.print("F");
```

(a) Equivalent

This is better

```java
if (score >= 90.0)
    System.out.print("A");
else if (score >= 80.0)
    System.out.print("B");
else if (score >= 70.0)
    System.out.print("C");
else if (score >= 60.0)
    System.out.print("D");
else
    System.out.print("F");
```

(b)
Multiple if-else statements

```
if score >= 90:
    grade is A
else:
    if score >= 80:
        grade is B
    elif score >= 70:
        grade is C
    elif score >= 60:
        grade is D
    else:
        grade is F
```
Nested statements

• The else clause matches the most recent if clause in the same block

```
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

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

Nothing is printed
Nested statements

• To force the else clause to match the first if clause, you must add a pair of braces

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

B is printed
Less error prone

```java
if (number % 2 == 0)
    even = true;
else
    even = false;
```

Equivalent:
```java
boolean even = number % 2 == 0;
```

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

Equivalent:
```java
if (even)
    System.out.println("It is even.");
```
## Logical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>logical negation</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>logical conjunction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or (xor)</td>
<td>logical exclusion</td>
</tr>
</tbody>
</table>
Truth table for operator `!`

<table>
<thead>
<tr>
<th>p</th>
<th>!p</th>
<th>Example: age = 24 and weight = 140</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td><code>(age &gt; 18)</code> is false, because <code>(age &gt; 18)</code> is true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td><code>(weight == 150)</code> is true, because <code>(weight == 150)</code> is false</td>
</tr>
</tbody>
</table>
### Truth table for operator &&

<table>
<thead>
<tr>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_1 &amp;&amp; p_2 )</th>
<th><strong>Example:</strong> ( \text{age} = 24 \ \text{and} \ \text{weight} = 140 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>((\text{age} \leq 18) &amp;&amp; (\text{weight} &lt; 140)) is false, because both conditions are false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>((\text{age} \leq 18) &amp;&amp; (\text{weight} \geq 140)) is false, because ((\text{age} \leq 18)) is false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>((\text{age} &gt; 18) &amp;&amp; (\text{weight} &gt; 140)) is false, because ((\text{weight} &gt; 140)) is false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>((\text{age} &gt; 18) &amp;&amp; (\text{weight} \geq 140)) is true, because both conditions are true</td>
</tr>
</tbody>
</table>
Truth table for operator $\lor$

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_1 \lor p_2$</th>
<th>Example: age = 24 and weight = 140</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 34) $\lor$ (weight $\geq$ 150) is false, because both conditions are false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 34) $\lor$ (weight $\leq$ 140) is true, because (weight $\leq$ 140) is true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(age &gt; 14) $\lor$ (weight $\geq$ 150) is false, because (age &gt; 14) is true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 14) $\lor$ (weight $\leq$ 140) is true, because both conditions are true</td>
</tr>
</tbody>
</table>
## Truth table for operator \(^\)\\

<table>
<thead>
<tr>
<th>(p_1)</th>
<th>(p_2)</th>
<th>(p_1 \land p_2)</th>
<th>Example: age = 24 and weight = 140</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 34) (^) (weight &gt; 140) is false, because both conditions are false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 34) (^) (weight &gt;= 140) is true, because (age &gt; 34) is false and (weight &gt;= 140) is true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(age &gt; 14) (^) (weight &gt; 140) is true, because (age &gt; 14) is true and (weight &gt; 140) is false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>(age &gt; 14) (^) (weight &gt;= 140) is false, because both conditions are true</td>
</tr>
</tbody>
</table>
switch statements

• When the value in a case statement matches the value of the switch-expression, the statements starting from this case are executed until either a break statement or the end of the switch statement is reached.

```java
switch (switch-expression) {
    case value1:  statement(s)1;
        break;
    case value2: statement(s)2;
        break;
    ...
    case valueN: statement(s)N;
        break;
    default: statement(s)-for-default;
}
```
switch statements

• The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses

• The value1, ..., and valueN must have the same data type as the value of the switch-expression

• The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression

• Note that value1, ..., and valueN are constant expressions (i.e., they cannot contain variables in the expression, such as 1 + x)

```c
switch (switch-expression) {
    case value1:  statement(s)1;  break;
    case value2: statement(s)2;  break;
    ...
    case valueN: statement(s)N;  break;
    default: statement(s)-for-default;
}
```
switch statements

• The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement
  – If the break statement is not present, the next case statement will be executed

• The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression

```plaintext
switch (switch-expression) {
  case value1:  statement(s)1;
    break;
  case value2: statement(s)2;
    break;
  ...
  case valueN: statement(s)N;
    break;
  default: statement(s)-for-default;
}
```
switch statements

switch (status) {
    case 0:  compute taxes for single filers;
            break;
    case 1:  compute taxes for married file jointly;
            break;
    case 2:  compute taxes for married file separately;
            break;
    case 3:  compute taxes for head of household;
            break;
    default: System.out.println("Errors: invalid status");
             System.exit(1);
}

switch statements

- status is 0: Compute tax for single filers -> break
- status is 1: Compute tax for married jointly or qualifying widow(er) -> break
- status is 2: Compute tax for married filing separately -> break
- status is 3: Compute tax for head of household -> break
- default: Default actions
switch statements

```
switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        System.out.println("Weekday");
        break;
    case 0:
    case 6:
        System.out.println("Weekend");
}
```
Conditional operator

• (boolean-expression) ? expression1 : expression2

```python
if (x > 0)
    y = 1
else
    y = -1;
```

is equivalent to

```python
y = (x > 0) ? 1 : -1;
```
Operator precedence

- `var++`, `var--`
- `+`, `-` (unary plus and minus), `++var`, `--var`
- (type) casting
- `!` (not)
- `*`, `/`, `%` (multiplication, division, and remainder)
- `+`, `-` (binary addition and subtraction)
- `<`, `<=`, `>`, `>=` (relational operators)
- `==`, `!=` (equality)
- `^` (exclusive OR)
- `&&` (conditional AND) short-circuit AND
- `||` (Conditional OR) short-circuit OR
- `=`, `+=`, `-=` `*=` `/=`, `%=` (assignment operator)
Operator associativity

• When two operators with the same precedence are evaluated, the associativity of the operators determines the order of evaluation

• All binary operators except assignment operators are left-associative
  \[ a - b + c - d \text{ is equivalent to } (((a - b) + c) - d) \]

• Assignment operators are right-associative
  \[ a = b += c = 5 \text{ is equivalent to } a = (b += (c = 5)) \]
Operator precedence and associativity

• The expression in the parentheses is evaluated first
  – Parentheses can be nested, in which case the expression in the inner parentheses is executed first

• When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule

• If operators with the same precedence are next to each other, their associativity determines the order of evaluation
Mathematical functions

• Java provides many useful methods in the Math class for performing common mathematical functions
• Math class constants
  – PI
  – E
• Math class methods
  – Trigonometric methods
  – Exponent methods
  – Rounding methods
  – min, max, abs, and random methods
Trigonometric methods

- Math.toDegrees(radians)
- Math.toRadians(degrees)
- Math.sin(radians)
- Math.cos(radians)
- Math.tan(radians)
- Math.acos(a)
- Math.asin(a)
- Math.atan(a)
Exponent methods

• Math.exp(a) $e^a$
• Math.log(a) $\log_e(a)$
• Math.log10(a) $\log_{10}(a)$
• Math.pow(a, b) $a^b$
• Math.sqrt(a)
Rounding methods

- Math.ceil(x)
- Math.floor(x)
- Math.rint(x)
- If you want to return an integer type, then
  - int Math.round(float x)
    - Returns (int)Math.floor(x + 0.5f)
  - long Math.round(double x)
    - Returns (long)Math.floor(x + 0.5)
min, max, abs, and random methods

- Math.min(a, b)
- Math.max(a, b)
- Math.abs(a)
- Math.random()
  - Returns a random double value in the range [0.0, 1.0)
char data type

char letter = 'A'; // ASCII
char numChar = '4'; // ASCII
char letter = '\u0041'; // Unicode
char numChar = '\u0034'; // Unicode

• Java characters use Unicode, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world’s diverse languages

• Unicode takes two bytes, preceded by \u, expressed in four hexadecimal numbers that run from \u0000 to \uFFFF
  – Unicode can represent 65535 + 1 characters
## Common and special characters

<table>
<thead>
<tr>
<th>Characters</th>
<th>Code Value in Decimal</th>
<th>Unicode Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'0' to '9'</td>
<td>48 to 57</td>
<td>\u0030 to \u0039</td>
</tr>
<tr>
<td>'A' to 'Z'</td>
<td>65 to 90</td>
<td>\u0041 to \u005A</td>
</tr>
<tr>
<td>'a' to 'z'</td>
<td>97 to 122</td>
<td>\u0061 to \u007A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Name</th>
<th>Unicode Code</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>Backspace</td>
<td>\u0008</td>
<td>8</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
<td>\u0009</td>
<td>9</td>
</tr>
<tr>
<td>\n</td>
<td>Linefeed</td>
<td>\u000A</td>
<td>10</td>
</tr>
<tr>
<td>\f</td>
<td>Formfeed</td>
<td>\u000C</td>
<td>12</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage Return</td>
<td>\u000D</td>
<td>13</td>
</tr>
<tr>
<td>&quot;</td>
<td>Backslash</td>
<td>\u005C</td>
<td>92</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double Quote</td>
<td>\u0022</td>
<td>34</td>
</tr>
</tbody>
</table>
Casting between char and numeric data types

```java
int i = 'a'; // Same as int i = (int)'a';

char c = 97; // Same as char c = (char)97;
```
Comparing and testing characters

if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");

• Methods in the char class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isDigit(ch)</td>
<td>Returns true if the specified character is a digit.</td>
</tr>
<tr>
<td>isLetter(ch)</td>
<td>Returns true if the specified character is a letter.</td>
</tr>
<tr>
<td>isLetterOfDigit(ch)</td>
<td>Returns true if the specified character is a letter or digit.</td>
</tr>
<tr>
<td>isLowerCase(ch)</td>
<td>Returns true if the specified character is a lowercase letter.</td>
</tr>
<tr>
<td>isUpperCase(ch)</td>
<td>Returns true if the specified character is an uppercase letter.</td>
</tr>
<tr>
<td>toLowerCase(ch)</td>
<td>Returns the lowercase of the specified character.</td>
</tr>
<tr>
<td>toUpperCase(ch)</td>
<td>Returns the uppercase of the specified character.</td>
</tr>
</tbody>
</table>
String type

- The char type only represents one character
- To represent a string of characters, use the String type
- String is a predefined class in the Java library (just like the System class and Scanner class)
  
```java
String message = "Welcome to Java";
```
- The String type is not a primitive type; it is known as a reference type
  - Any Java class can be used as a reference type for a variable
## Simple String methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length()</td>
<td>Returns the number of characters in this string.</td>
</tr>
<tr>
<td>charAt(index)</td>
<td>Returns the character at the specified index from this string.</td>
</tr>
<tr>
<td>concat(s1)</td>
<td>Returns a new string that concatenates this string with string s1.</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>Returns a new string with all letters in uppercase.</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>Returns a new string with all letters in lowercase.</td>
</tr>
<tr>
<td>trim()</td>
<td>Returns a new string with whitespace characters trimmed on both sides.</td>
</tr>
</tbody>
</table>

- These methods can only be invoked from a specific string instance
  - These methods are called instance methods
Instance methods vs static methods

• These methods can only be invoked from a specific string instance
  – These methods are called instance methods
  – The syntax to invoke an instance method is `referenceVariable.methodName(arguments)`

• A non-instance method is called a static method
  – A static method can be invoked without using an object (i.e., they are not tied to a specific object instance)
  – For example, all the methods defined in the Math class are static methods
Getting characters from a string

String message = "Welcome to Java";
System.out.println("The first character in message is " + message.charAt(0));

Indices
message

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
W e l c o m e t o J a v a

message.charAt(0)  message.length() is 15  message.charAt(14)
String concatenation

```java
String s3 = s1.concat(s2); // These two are equivalent
String s3 = s1 + s2;

// 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
```
Reading a string from the console

Scanner input = new Scanner(System.in);
System.out.print("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);
Reading a character from the console

Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
## Comparing strings

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals(s1)</td>
<td>Returns true if this string is equal to string s1.</td>
</tr>
<tr>
<td>equalsIgnoreCase(s1)</td>
<td>Returns true if this string is equal to string s1; it is case insensitive.</td>
</tr>
<tr>
<td>compareTo(s1)</td>
<td>Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.</td>
</tr>
<tr>
<td>compareToIgnoreCase(s1)</td>
<td>Same as compareTo except that the comparison is case insensitive.</td>
</tr>
<tr>
<td>startsWith(prefix)</td>
<td>Returns true if this string starts with the specified prefix.</td>
</tr>
<tr>
<td>endsWith(suffix)</td>
<td>Returns true if this string ends with the specified suffix.</td>
</tr>
</tbody>
</table>
# Substrings

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>substring(beginIndex)</code></td>
<td>Returns this string’s substring that begins with the character at the specified <code>beginIndex</code> and extends to the end of the string, as shown in Figure 4.2.</td>
</tr>
<tr>
<td><code>substring(beginIndex, endIndex)</code></td>
<td>Returns this string’s substring that begins at the specified <code>beginIndex</code> and extends to the character at index <code>endIndex - 1</code>, as shown in Figure 9.6. Note that the character at <code>endIndex</code> is not part of the substring.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14</td>
</tr>
<tr>
<td><code>Welcome to Java</code></td>
</tr>
</tbody>
</table>

```
message.substring(0, 11)  message.substring(11)
```
## Finding a character or a substring in a string

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>indexOf(ch)</code></td>
<td>Returns the index of the first occurrence of <code>ch</code> in the string. Returns −1 if not matched.</td>
</tr>
<tr>
<td><code>indexOf(ch, fromIndex)</code></td>
<td>Returns the index of the first occurrence of <code>ch</code> after <code>fromIndex</code> in the string. Returns −1 if not matched.</td>
</tr>
<tr>
<td><code>indexOf(s)</code></td>
<td>Returns the index of the first occurrence of string <code>s</code> in this string. Returns −1 if not matched.</td>
</tr>
<tr>
<td><code>indexOf(s, fromIndex)</code></td>
<td>Returns the index of the first occurrence of string <code>s</code> in this string after <code>fromIndex</code>. Returns −1 if not matched.</td>
</tr>
<tr>
<td><code>lastIndexOf(ch)</code></td>
<td>Returns the index of the last occurrence of <code>ch</code> in the string. Returns −1 if not matched.</td>
</tr>
<tr>
<td><code>lastIndexOf(ch, fromIndex)</code></td>
<td>Returns the index of the last occurrence of <code>ch</code> before <code>fromIndex</code> in this string. Returns −1 if not matched.</td>
</tr>
<tr>
<td><code>lastIndexOf(s)</code></td>
<td>Returns the index of the last occurrence of string <code>s</code>. Returns −1 if not matched.</td>
</tr>
<tr>
<td><code>lastIndexOf(s, fromIndex)</code></td>
<td>Returns the index of the last occurrence of string <code>s</code> before <code>fromIndex</code>. Returns −1 if not matched.</td>
</tr>
</tbody>
</table>
Finding a character or a substring in a string

```
int k = s.indexOf(' ');  
String firstName = s.substring(0, k);  
String lastName = s.substring(k + 1);
```
Conversion between strings and numbers

```java
int intValue = Integer.parseInt(intString);
double doubleValue =
    Double.parseDouble(doubleString);

String s = number + "";
```
Formatting output

• Use the printf statement
  \texttt{System.out.printf(format, items);}

• Where \texttt{format} is a string that may consist of substrings and format specifiers
  – A format specifier specifies how an item should be displayed
  – Each specifier begins with a percent sign
  – An item may be a numeric value, character, boolean value, or a string
Common specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Output</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%b</td>
<td>a boolean value</td>
<td>true or false</td>
</tr>
<tr>
<td>%c</td>
<td>a character</td>
<td>'a'</td>
</tr>
<tr>
<td>%d</td>
<td>a decimal integer</td>
<td>200</td>
</tr>
<tr>
<td>%f</td>
<td>a floating-point number</td>
<td>45.460000</td>
</tr>
<tr>
<td>%e</td>
<td>a number in standard scientific notation</td>
<td>4.556000e+01</td>
</tr>
<tr>
<td>%s</td>
<td>a string</td>
<td>&quot;Java is cool&quot;</td>
</tr>
</tbody>
</table>

```java
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

display count is 5 and amount is 45.560000
Next Lecture

- Loops
- Methods
- Reading
  - Chapters 5 and 6