Mathematical Functions, Characters, and Strings

Introduction to Programming and Computational Problem Solving - 2

CSE 8B

Lecture 3
Announcements

- Assignment 1 is due Oct 5, 11:59 PM
- Quiz 1 is Oct 7
- Assignment 2 will be released Oct 5
  - Due Oct 12, 11:59 PM
- Educational research study
  - Oct 7, weekly survey
- Reading
  - Liang
    - Chapters 2 and 4
Mathematical functions, characters, and strings

- Numerical data types (e.g., an integer)
- Numeric operations (e.g., addition)
- Mathematical functions (e.g., cosine)
- Reading numbers from the console
- Character data type (i.e., char)
- Comparing and testing characters
- String data type (i.e., String)
- Simple string methods (e.g., number of characters in the string)
- Reading a character and string from the console
# Numerical data types

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>$-2^7$ to $2^7 - 1$ (-128 to 127)</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>$-2^{15}$ to $2^{15} - 1$ (-32768 to 32767)</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>$-2^{31}$ to $2^{31} - 1$ (-2147483648 to 2147483647)</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>$-2^{63}$ to $2^{63} - 1$</td>
<td>64-bit signed</td>
</tr>
<tr>
<td></td>
<td>(i.e., -9223372036854775808 to 9223372036854775807)</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>Negative range:</td>
<td>32-bit IEEE 754</td>
</tr>
<tr>
<td></td>
<td>-3.4028235E+38 to -1.4E-45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Positive range:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.4E-45 to 3.4028235E+38</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>Negative range:</td>
<td>64-bit IEEE 754</td>
</tr>
<tr>
<td></td>
<td>-1.7976931348623157E+308 to -4.9E-324</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Positive range:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.9E-324 to 1.7976931348623157E+308</td>
<td></td>
</tr>
</tbody>
</table>
Number literals

- A literal is a constant value that appears directly in the program

```java
int i = 34;
long x = 1000000;
double d = 5.0 + 1.0;
```

- 34, 1000000, 5.0, and 1.0 are literals
Integer literals

• An integer literal can be assigned to an integer variable as long as it can fit into the variable

• A compilation error will occur if the literal is too large for the variable to hold
  – For example, the statement `byte b = 1000` would cause a compilation error, because 1000 cannot be stored in a variable of the `byte` type

• An integer literal is assumed to be of the `int` type, whose value is between $-2^{31}$ (equals -2147483648) to $2^{31}-1$ (equals 2147483647)

• To denote an integer literal of the `long` type, append it with the letter L or l
  – L is preferred because l (lowercase L) can easily be confused with 1 (the digit one)
Floating-point literals

• Floating-point literals are written with a decimal point
• By default, a floating-point literal is treated as a double type value
  – Example: 5.0 is considered a double value, not a float value
• You can make a number a float by appending the letter f or F, and make a number a double by appending the letter d or D
  – Example: you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number
 Scientific notation

• Floating-point literals can also be specified in scientific notation
  – Example: $1.23456e+2$ (same as $1.23456e2$) is equivalent to $123.456$, and $1.23456e-2$ is equivalent to $0.0123456$

• E or e represents an exponent
# Numeric operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>34.0 - 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>
double vs float

• The double type values are more accurate than the float type values
  – For example,

  ```java
  System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
  ```
  displays `1.0 / 3.0 is 0.3333333333333333`
  
  ```java
  System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
  ```
  displays `1.0F / 3.0F is 0.33333334`
  
  16 digits
  7 digits
Floating-point accuracy

• Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy
• For example,
  System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
  displays 0.5000000000000001, not 0.5, and
  System.out.println(1.0 - 0.9);
  displays 0.0999999999999998, not 0.1
• Integers are stored precisely
  – Calculations with integers yield a precise integer result
Integer division

- Warning: resulting fractional part (i.e., values after the decimal point) are **truncated, not rounded**
  - Example: $5 \div 2$ yields an integer 2
Remainder operator

• Example: an even number \( \% \ 2 \) is always 0 and an odd number \( \% \ 2 \) is always 1
  – You can use this property to determine whether a number is even or odd
• Example: If today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression.

Saturday is the 6\(^{th}\) day in a week

\[
(6 + 10) \ % \ 7 \text{ is } 2
\]

A week has 7 days

After 10 days

The 2\(^{nd}\) day in a week is Tuesday
# Augmented assignment operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>Addition assignment</td>
<td><code>i += 8</code></td>
<td><code>i = i + 8</code></td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assignment</td>
<td><code>i -= 8</code></td>
<td><code>i = i - 8</code></td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assignment</td>
<td><code>i *= 8</code></td>
<td><code>i = i * 8</code></td>
</tr>
<tr>
<td>/=</td>
<td>Division assignment</td>
<td><code>i /= 8</code></td>
<td><code>i = i / 8</code></td>
</tr>
<tr>
<td>%=</td>
<td>Remainder assignment</td>
<td><code>i %= 8</code></td>
<td><code>i = i % 8</code></td>
</tr>
</tbody>
</table>
# Increment and decrement operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
<th>Example (assume $i = 1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>++var</code></td>
<td>preincrement</td>
<td>Increment var by 1, and use the new var value in the statement</td>
<td><code>int j = ++i;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>// j is 2, i is 2</code></td>
</tr>
<tr>
<td><code>var++</code></td>
<td>postincrement</td>
<td>Increment var by 1, but use the original var value in the statement</td>
<td><code>int j = i++;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>// j is 1, i is 2</code></td>
</tr>
<tr>
<td><code>--var</code></td>
<td>predecrement</td>
<td>Decrement var by 1, and use the new var value in the statement</td>
<td><code>int j = --i;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>// j is 0, i is 0</code></td>
</tr>
<tr>
<td><code>var--</code></td>
<td>postdecrement</td>
<td>Decrement var by 1, and use the original var value in the statement</td>
<td><code>int j = i--;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>// j is 1, i is 0</code></td>
</tr>
</tbody>
</table>
Conversion rules

• When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules
  1. If one of the operands is double, the other is converted into double
  2. Otherwise, if one of the operands is float, the other is converted into float
  3. Otherwise, if one of the operands is long, the other is converted into long
  4. Otherwise, both operands are converted into int
Type casting

Implicit casting
  double d = 3; (type widening)

Explicit casting
  int i = (int)3.0; (type narrowing)
  int i = (int)3.9; (fraction part is truncated, not rounded!)

range increases

byte, short, int, long, float, double
Operator precedence

- ( ), var++, var--
- ++var, --var, +, - (unary plus and minus), ! (not)
- (type) casting
- *, /, % (multiplication, division, and remainder)
- +, - (binary addition and subtraction)
- <, <=, >, >= (relational operators)
- ==, != (equality)
- ^ (exclusive or)
- && (and)
- || (or)
- =, +=, -=, *=, /=, %= (assignment operators)

Relational and logical operators will be covered next lecture
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
Reading numbers from the console

1. Create a Scanner object

   Scanner input = new Scanner(System.in);

2. Use the method nextDouble() to obtain to a double value. Example:

   System.out.print("Enter a double value: ");
   Scanner input = new Scanner(System.in);
   double d = input.nextDouble();
Reading numbers from the console

Scanner input = new Scanner(System.in);
int value = input.nextInt();

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextByte()</td>
<td>reads an integer of the byte type.</td>
</tr>
<tr>
<td>nextShort()</td>
<td>reads an integer of the short type.</td>
</tr>
<tr>
<td>nextInt()</td>
<td>reads an integer of the int type.</td>
</tr>
<tr>
<td>nextLong()</td>
<td>reads an integer of the long type.</td>
</tr>
<tr>
<td>nextFloat()</td>
<td>reads a number of the float type.</td>
</tr>
<tr>
<td>nextDouble()</td>
<td>reads a number of the double type.</td>
</tr>
</tbody>
</table>
Explicit import and implicit Import

• At top of source file

import java.util.Scanner; // Explicit Import

import java.util.*; // Implicit import
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) \quad e^a
Math.log(a) \quad \log_e(a)
Math.log10(a) \quad \log_{10}(a)
Math.pow(a, b) \quad a^b
Math.sqrt(a) \quad \sqrt{a}
Rounding methods

Math.ceil(x)
Math.floor(x)
Math.rint(x)

• If you want to return an integer type, then

\[
\text{int Math.round(float x)}
\]
  • Returns (int)Math.floor(x + 0.5f)

\[
\text{long Math.round(double x)}
\]
  • Returns (long)Math.floor(x + 0.5)

nearest integer not less than x
nearest integer not greater than x
x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned (i.e., round to nearest, round half to even)
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 65536 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>\ \</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>
Comparing and testing characters

```java
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><code>isDigit(ch)</code></td>
<td>Returns true if the specified character is a digit.</td>
</tr>
<tr>
<td><code>isLetter(ch)</code></td>
<td>Returns true if the specified character is a letter.</td>
</tr>
<tr>
<td><code>isLetterOrDigit(ch)</code></td>
<td>Returns true if the specified character is a letter or digit.</td>
</tr>
<tr>
<td><code>isLowerCase(ch)</code></td>
<td>Returns true if the specified character is a lowercase letter.</td>
</tr>
<tr>
<td><code>isUpperCase(ch)</code></td>
<td>Returns true if the specified character is an uppercase letter.</td>
</tr>
<tr>
<td><code>toLowerCase(ch)</code></td>
<td>Returns the lowercase of the specified character.</td>
</tr>
<tr>
<td><code>toUpperCase(ch)</code></td>
<td>Returns the uppercase of the specified character.</td>
</tr>
</tbody>
</table>

Relational and logical operators will be covered next lecture
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;
```
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)
  – The syntax to invoke a static method is
    `ClassName.methodName(arguments)`
  – For example, all the methods defined in the Math class are
    static methods

Methods will be covered next week
Getting characters from a string

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

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
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><code>equals(s1)</code></td>
<td>Returns true if this string is equal to string s1.</td>
</tr>
<tr>
<td><code>equalsIgnoreCase(s1)</code></td>
<td>Returns true if this string is equal to string s1; it is case insensitive.</td>
</tr>
<tr>
<td><code>compareTo(s1)</code></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><code>compareToIgnoreCase(s1)</code></td>
<td>Same as <code>compareTo</code> except that the comparison is case insensitive.</td>
</tr>
<tr>
<td><code>startsWith(prefix)</code></td>
<td>Returns true if this string starts with the specified prefix.</td>
</tr>
<tr>
<td><code>endsWith(suffix)</code></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.</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 extend to the character at index <code>endIndex - 1</code>. Note that the character at <code>endIndex</code> is not part of the substring.</td>
</tr>
</tbody>
</table>

Indices:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>e</td>
<td>l</td>
<td>c</td>
<td>o</td>
<td>m</td>
<td>e</td>
<td>t</td>
<td>o</td>
<td></td>
<td>J</td>
<td>a</td>
<td>v</td>
<td>a</td>
<td></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 <code>-1</code> 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 <code>-1</code> 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 <code>-1</code> 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 <code>-1</code> 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 <code>-1</code> 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 <code>-1</code> 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 <code>-1</code> 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 <code>-1</code> if not matched.</td>
</tr>
</tbody>
</table>
Finding a character or a substring in a string

```java
int k = s.indexOf(' ');  
String firstName = s.substring(0, k);  
String lastName = s.substring(k + 1);  
```

![Diagram](https://via.placeholder.com/150)
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
  
  System.out.printf(format, items);

• Where 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

• Selections
• Loops
• Reading
  – Liang
    • Chapters 3 and 5