BBM 102 – Introduction to Programming II

Spring 2018

Introduction to Java &
Introduction to Object Orientation
Today

- **Introduction to Java**
  - Java as a Platform
  - Your First Java Program
  - Basic Programming Elements

- **Object Oriented Paradigm**
  - Principles of Object Orientation
  - Classes and Objects
  - Sample Object Designs
What is Java?

- An island of Indonesia lying between the Indian Ocean and the Java Sea.
What is Java?

- Informal. Brewed coffee.
What is Java?

- A technology which is both a programming language and a platform.
- Developed by Sun Microsystems.
- First public version was released in 1995.
Software Development with Java

- All source code is first written in plain text files ending with the “.java” extension.
- Those source files are then compiled into “.class” files by the javac compiler.
- A “.class” file does not contain code that is native to your processor; it instead contains bytecodes — the machine language of the Java Virtual Machine (Java VM).
- The java launcher tool then runs your application with an instance of the Java Virtual Machine, i.e. your code is run by JVM.

http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html
Platform Independence: Write Once Run Anywhere

- Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS.

http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html
The Java Platform

- A platform is the hardware or software environment in which a program runs.

- The Java platform has two components:
  - The **Java Virtual Machine**: It's the base for the Java platform and is ported onto various hardware-based platforms
  - The **Java Application Programming Interface (API)**: It is a large collection of ready-made software components that provide many useful capabilities.

- As a platform-independent environment, the Java platform can be a bit slower than native code.
  - However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.

http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello world!");
    }

}
Basic Programming Elements

- Variables, Types and Expressions
- Flow of Control
  ✦ Branching
  ✦ Loops
Variables

- **Variables** in a program are used to store data such as numbers and letters. They can be thought of as containers of a sort.
- You should choose variable names that are helpful. Every variable in a Java program must be declared before it is used for the first time.
- A variable declaration consists of a type name, followed by a list of variable names separated by commas. The declaration ends with a semicolon.

**Syntax:**
```
data_type  variable_name  [  =  initial_value  ];
```

```java
int  styleNumber,  numberOfChecks,  numberOfDeposits;
double  amount,  interestRate;
char  answer;
```
# Primitive Data Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Kind of Value</th>
<th>Memory Used</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Integer</td>
<td>1 byte</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>Integer</td>
<td>2 bytes</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>4 bytes</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>Integer</td>
<td>8 bytes</td>
<td>-9,223,372,036,854,75,808 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>float</td>
<td>Floating-point</td>
<td>4 bytes</td>
<td>±3.40282347 × 10+38 to ±1.40239846 × 10−45</td>
</tr>
<tr>
<td>double</td>
<td>Floating-point</td>
<td>8 bytes</td>
<td>±1.79769313486231570 × 10+308 to ±4.94065645841246544 × 10−324</td>
</tr>
<tr>
<td>char</td>
<td>Single character (Unicode)</td>
<td>2 bytes</td>
<td>All Unicode values from 0 to 65,535</td>
</tr>
<tr>
<td>boolean</td>
<td></td>
<td>1 bit</td>
<td>True or false</td>
</tr>
</tbody>
</table>

There are also Class Data Types which we will cover later.
Identifiers

- The technical term for a name in a programming language, such as the name of a variable, is an **identifier**.
- An identifier can contain only letters, digits 0 through 9, and the underscore character “_”.
- The first character in an identifier cannot be a digit.
- There is no limit to the length of an identifier.
- Java is **case sensitive** (e.g., `personName` and `personname` are two different variables).

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Valid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputStream</td>
<td>Yes</td>
</tr>
<tr>
<td>4you</td>
<td>No</td>
</tr>
<tr>
<td>my.work</td>
<td>No</td>
</tr>
<tr>
<td>FirstName</td>
<td>Yes</td>
</tr>
<tr>
<td>_tmp</td>
<td>Yes</td>
</tr>
<tr>
<td>Public</td>
<td>No</td>
</tr>
</tbody>
</table>

Public is a reserved word.
# Java Reserved Words

<table>
<thead>
<tr>
<th>abstract</th>
<th>assert</th>
<th>boolean</th>
<th>break</th>
<th>byte</th>
<th>case</th>
</tr>
</thead>
<tbody>
<tr>
<td>catch</td>
<td>char</td>
<td>class</td>
<td>const</td>
<td>continue</td>
<td>default</td>
</tr>
<tr>
<td>double</td>
<td>do</td>
<td>else</td>
<td>enum</td>
<td>extends</td>
<td>FALSE</td>
</tr>
<tr>
<td>final</td>
<td>finally</td>
<td>float</td>
<td>for</td>
<td>goto</td>
<td>if</td>
</tr>
<tr>
<td>implements</td>
<td>import</td>
<td>instanceof</td>
<td>int</td>
<td>interface</td>
<td>long</td>
</tr>
<tr>
<td>native</td>
<td>new</td>
<td>null</td>
<td>package</td>
<td>private</td>
<td>protected</td>
</tr>
<tr>
<td>public</td>
<td>return</td>
<td>short</td>
<td>static</td>
<td>strictfp</td>
<td>super</td>
</tr>
<tr>
<td>switch</td>
<td>synchronized</td>
<td>this</td>
<td>throw</td>
<td>throws</td>
<td>transient</td>
</tr>
<tr>
<td>TRUE</td>
<td>try</td>
<td>void</td>
<td>volatile</td>
<td>while</td>
<td></td>
</tr>
</tbody>
</table>
Naming Conventions

- Class types begin with an uppercase letter (e.g. `String`).
- Primitive types begin with a lowercase letter (e.g. `float`).
- Variables of both class and primitive types begin with a lowercase letters (e.g. `firstName`, `classAverage`).
- Multiword names are "punctuated" using uppercase letters.
Assignment Statements

- An assignment statement is used to assign a value to a variable.
- The "equal sign" is called the assignment operator.
- Syntax:

  ```
  variable_name = expression;
  ```

  where `expression` can be another variable, a literal or constant, or something to be evaluated by using `operators`.

  ```
  amount = 100;
  interestRate = 0.12;
  answer = 'Y';
  fullName = firstName + ' ' + lastName;
  ```
Initializing Variables

- A variable that has been declared, but not yet given a value is said to be *uninitialized*.
- Uninitialized class variables have the value **null**.
- Uninitialized primitive variables may have a default value.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>0</td>
</tr>
<tr>
<td>short</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>long</td>
<td>0L</td>
</tr>
<tr>
<td>float</td>
<td>0.0f</td>
</tr>
<tr>
<td>double</td>
<td>0.0d</td>
</tr>
<tr>
<td>char</td>
<td>&quot;\u0000'</td>
</tr>
<tr>
<td>String (or any object)</td>
<td>null</td>
</tr>
<tr>
<td>boolean</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

- It's good practice not to rely on a default value.
Constants

- Literal expressions such as 2, 3.7, or 'y' are called constants.

- Integer constants can be preceded by a + or - sign, but cannot contain commas.

- Floating-point constants can be written with digits after a decimal point or using e notation.
  - 765000000.0 can be written as 7.65e8
  - 0.000483 can be written as 4.83e-4
Imprecision in Floating Point Numbers

- Floating-point numbers often are only approximations since they are stored with a finite number of bits.

- Hence $1.0/3.0$ is slightly less than $1/3$.

- $1.0/3.0 + 1.0/3.0 + 1.0/3.0$ is less than $1$. 
Named Constants

- Java provides a mechanism that allows you to define a variable, initialise it, and moreover fix the variable’s value so that it cannot be changed.

```java
public static final Type Variable = Constant;
```

- The convention for naming constants is to use all uppercase letters, with an underscore symbol “_” between words.

```java
public static final double PI = 3.14159;
public static final int DAYS_PER_WEEK = 7;
...
float area = PI * r * r ;
int daysInYear = 52 * DAYS_PER_WEEK ;
```
Assignment Compatibility

- Java is *strongly typed*.

- A value of one type can be assigned to a variable of any type further to the right (not to the left):

  \[ \text{byte} \rightarrow \text{short} \rightarrow \text{int} \rightarrow \text{long} \rightarrow \text{float} \rightarrow \text{double} \]

- You can assign a value of type `char` to a variable of type `int`. 
Type Conversion (Casting)

- Implicit conversion

```java
double doubleVariable = 5;    // 5.0
int intVariable = 5;          // 5
doubleVariable = intVariable; // 5.0
```

- Explicit conversion

```java
double doubleVariable = 5.0;
int intVariable = doubleVariable;   // Illegal
int intVariable = (int) doubleVariable; // Legal, 5
```
Operators and Precedence

- **Precedence**
  - **First**: The unary operators: plus (+), minus (-), not (!), increment (++) and decrement (--)  
  - **Second**: The binary arithmetic operators: multiplication (*), integer division (/) and modulus (%)  
  - **Third**: The binary arithmetic operators: addition (+) and subtraction (-)

When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.

When unary operators have equal precedence, the operator on the right acts before the operation(s) on the left.

Parenthesis can change the precedence.
### Operators and Precedence - Example

<table>
<thead>
<tr>
<th>Ordinary Math</th>
<th>Java (Preferred Form)</th>
<th>Java (Parenthesized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(rate^2 + \text{delta})</td>
<td>rate * rate + delta</td>
<td>(rate * rate) + delta</td>
</tr>
<tr>
<td>2(salary + bonus)</td>
<td>2 * (salary + bonus)</td>
<td>2 * (salary + bonus)</td>
</tr>
<tr>
<td>(\frac{1}{time + 3\text{mass}})</td>
<td>1 / (time + 3 * mass)</td>
<td>1 / (time + (3 * mass))</td>
</tr>
<tr>
<td>(\frac{a - 7}{t + 9v})</td>
<td>(a - 7) / (t + 9 * v)</td>
<td>(a - 7) / (t + (9 * v))</td>
</tr>
</tbody>
</table>

Figure from “Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson, 2012”
Arrays

- Array is a sequence of values.
- Array indices begin at zero.
- Defining Arrays

```csharp
Base_Type[] Array_Name = new Base_Type[Length];

int[] numbers = new int[100]; // or,
int[] numbers;
numbers = new int[100];
```

- Initialising Arrays

```csharp
double[] reading = {3.3, 15.8, 9.7}; // or,
double[] reading = new double[3];
reading[0] = 3.3;
reading[1] = 15.8;
reading[2] = 9.7;
```
Strings

- A value of type **String** is a
  - Sequence (Array) of characters treated as a single item
  - Character positions start with 0

![Indices](Indices.png)

*Note that the blanks and the period count as characters in the string.*

- Can be declared in three ways:

```java
String greeting;
greeting = "Hello World!";

String greeting = "Hello World!";

String greeting = new String("Hello World!");
```

Figure from “Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson, 2012”
Concatenating Strings

You can connect—or join or paste—two strings together to obtain a larger string. This operation is called concatenation and is performed by using the “+” operator.

```java
String greeting, sentence;
greeting = "Hello";

sentence = greeting + " my friend!";
System.out.println(sentence); // Hello my friend!

String solution = "The answer is " + 42;
System.out.println(solution); // The answer is 42

// Java converts the number constant 42 to the
// string constant "42" and then concatenates the
// two strings
```
**String Methods**

- **Homework**: Investigate the methods given below. You will be responsible in the exams.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>charAt (Index)</code></td>
<td>Get character at the specified index.</td>
</tr>
<tr>
<td><code>length()</code></td>
<td>Return the length of the string.</td>
</tr>
<tr>
<td><code>compareTo(A_String)</code></td>
<td>Compare two strings and return -1, 0, or 1.</td>
</tr>
<tr>
<td><code>replace(OldChar, NewChar)</code></td>
<td>Replace all occurrences of <code>OldChar</code> with <code>NewChar</code>.</td>
</tr>
<tr>
<td><code>concat(A_String)</code></td>
<td>Concatenate two strings.</td>
</tr>
<tr>
<td><code>substring(Start)</code></td>
<td>Return a substring starting from the given index.</td>
</tr>
<tr>
<td><code>equals(Other_String)</code></td>
<td>Check if two strings are equal.</td>
</tr>
<tr>
<td><code>equalsIgnoreCase(Other_String)</code></td>
<td>Case-insensitive string equality check.</td>
</tr>
<tr>
<td><code>toLowerCase()</code></td>
<td>Convert to lowercase.</td>
</tr>
<tr>
<td><code>indexOf(A_String)</code></td>
<td>Find the index of the first occurrence of <code>A_String</code> in the string.</td>
</tr>
<tr>
<td><code>toUpperCase()</code></td>
<td>Convert to uppercase.</td>
</tr>
<tr>
<td><code>lastIndexOf(A_String)</code></td>
<td>Find the index of the last occurrence of <code>A_String</code> in the string.</td>
</tr>
<tr>
<td><code>trim()</code></td>
<td>Remove leading and trailing whitespace.</td>
</tr>
</tbody>
</table>
Boolean Type

- Java has the logical type boolean
- Type boolean has two literal constants
  - true
  - false

```java
int number = -5;
boolean isPositive = (number > 0);  // False
```
### Java Comparison Operators

<table>
<thead>
<tr>
<th>Math Notation</th>
<th>Name</th>
<th>Java Notation</th>
<th>Java Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
<td>==</td>
<td>balance == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>answer == 'y'</td>
</tr>
<tr>
<td>≠</td>
<td>Not equal to</td>
<td>!=</td>
<td>income != tax</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>answer != 'y'</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>&gt;</td>
<td>expenses &gt; income</td>
</tr>
<tr>
<td>≥</td>
<td>Greater than or equal to</td>
<td>&gt;=</td>
<td>points &gt;= 60</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>&lt;</td>
<td>pressure &lt; max</td>
</tr>
<tr>
<td>≤</td>
<td>Less than or equal to</td>
<td>&lt;=</td>
<td>expenses &lt;= income</td>
</tr>
</tbody>
</table>
# Java Logical Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Java Notation</th>
<th>Java Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical <em>and</em></td>
<td><code>&amp;&amp;</code></td>
<td><code>(sum &gt; min) &amp;&amp; (sum &lt; max)</code></td>
</tr>
<tr>
<td>Logical <em>or</em></td>
<td>`</td>
<td></td>
</tr>
<tr>
<td>Logical <em>not</em></td>
<td><code>!</code></td>
<td>!(number &lt; 0)</td>
</tr>
</tbody>
</table>
Flow of Control

- *Flow of control* is the order in which a program performs actions.

- A *branching statement* chooses between two or more possible actions.
  ✤ If-else, switch statements

- A *loop statement* repeats an action until a stopping condition occurs.
  ✤ For, while, do-while loops
Basic if Statement

- Syntax
  
  \[
  \text{if (Expression)} \\
  \text{Action}
  \]

- If the *Expression* is true then execute *Action*

- *Action* is either a single statement or a group of statements within braces

```java
if (value2 < value1) {
    // Rearrange numbers so
    int tmp = value1; // value2 variable should
    value1 = value2; // hold the bigger value
    value2 = tmp;
}
```
if-else Statement

- Syntax
  
  ```
  if (Expression)
      Action1
  else
      Action2
  ```

- If `Expression` is true then execute `Action1` otherwise execute `Action2`

- The actions are either a single statement or a list of statements within braces

```c
int maximum;
if (value1 < value2) { // is value2 larger?
    maximum = value2; // yes: value2 is larger
}
else {                // (value1 >= value2)
    maximum = value1; // no: value2 is not larger
}
```
if-else-if Statement

- If statements can be nested (also called as multi-way, multi-branch if statement)

```java
if (a == '0')
    System.out.println ("zero");
else if (a == '1')
    System.out.println ("one");
else if (a == '2')
    System.out.println ("two");
else if (a == '3')
    System.out.println ("three");
else if (a == '4')
    System.out.println ("four");
else
    System.out.println ("five+");
```
Switch Statement

- Switch statement can be used instead of multi-way if statement.
- Syntax

  ```java
  switch(controlling_expression) {
    case expression1:
      action1;
      break;
    case expression2:
      action2;
      break;
    ...
    default:
      actionN;
  }
  ```

- Every case ends with `break` statement.
Switch Statement

Switch statements are more readable than nested if statements

```java
switch (a) {
    case '0':
        System.out.println ("zero"); break;
    case '1':
        System.out.println ("one"); break;
    case '2':
        System.out.println ("two"); break;
    case '3':
        System.out.println ("three"); break;
    case '4':
        System.out.println ("four"); break;
    default:
        System.out.println ("five+"); break;
}
```
The Conditional (Ternary) Operator

- The ? and : together are called the *conditional operator* or *ternary operator*.

```plaintext
if (n1 > n2)
    max = n1;
else
    max = n2;
```

can be written as:

```plaintext
max = (n1 > n2) ? n1 : n2;
```
for Loops

- The for loop is a pretest loop statement. It has the following form.

```java
for (initialisation; boolean-expression; increment) {
    nested-statements
}
```

- initialisation is evaluated first.
- boolean-expression is tested before each iteration of the loop.
- increment is evaluated at the end of each iteration.
- nested-statements is a sequence of statements. If there is only one statement then the braces may be omitted.
Varying Control Variable

- for ( int i = 1; i <= 100; i++ )
  ✪ from 1 to 100 in increments of 1

- for ( int i = 100; i >= 1; i-- )
  ✪ from 100 to 1 in increments of -1

- for ( int i = 7; i <= 77; i += 7 )
  ✪ from 7 to 77 in increments of 7

- for ( int i = 20; i >= 2; i -= 2 )
  ✪ from 20 to 2 in decrements of 2
For Loop Example

String[] classList = {"Jean", "Claude", "Van", "Damme"};

for (int i=0; i<classList.length; i++) {
    System.out.println(classList[i]);
}

for (String name : classList) {
    System.out.println(name);
}
While Loop

- The while loop is a pretest loop statement. It has the following form.

  ```
  while (boolean-expression) {
    nested-statements
  }
  ```

- `boolean-expression` is an expression that can be true or false.

- `nested-statements` is a sequence of statements. If there is only one statement then the braces can be omitted.

- The boolean expression is tested before each iteration of the loop. The loop terminates when it is false.
While Loop Example

```java
int[] numbers = { 1, 5, 3, 4, 2 };
int i=0, key = 33;  // Let's look for something that does not exist.
boolean found = false;

while (!found){  // Is there a problem here?
    if (numbers[i++] == key)
        found=true;
}

if (found)
    System.out.println("Key is found in the array");
else
    System.out.println("Key is NOT found");
```
While Loop Example

```java
int[] numbers = { 1, 5, 3, 4, 2 }; int i=0, key = 33;

boolean found = false;

while (!found && i<numbers.length){
    if (numbers[i++] == key)
        found=true;
}

if (found)
    System.out.println("Key is found in the array");
else
    System.out.println("Key is NOT found!");
```

Make sure that the loop ends somehow.
Do-While Loop

- The do-while loop is a post-test loop statement. It has the following form.

  ```
  do {
    nested-statements
  } while (boolean-expression);
  ```

- `nested-statements` is a sequence of statements. If there is only one statement then the braces may be omitted.

- `boolean-expression` is an expression that can be true or false.

- The boolean expression is tested after each iteration of the loop. The loop terminates when it is false.
Do-While Example

Scanner scan = new Scanner(System.in);
int myNumber;

do {
    System.out.println(
        "Enter a number between 0 and 100: ");

    myNumber = scan.nextInt();
} while (!(myNumber >= 0 && myNumber <= 100));

System.out.println("You entered a valid number");
The break statement is used in loop (for, while, and do-while) statements and switch statements to terminate execution of the statement. A break statement has the following form.

```java
break;
```

After a break statement is executed, execution proceeds to the statement that follows the enclosing loop or switch statement.

Use `break` statements sparingly (if ever).
Continue Statement

- A `continue` statement
  - Ends current loop iteration
  - Begins the next one

- Use of continue statement is not recommended
  - Introduce unneeded complications
Breaking a Loop

```java
int[] numbers = { 1, 5, 3, 4, 2 };
int i = 0, key = 3;

while (i < numbers.length) {
    if (numbers[i] == key)
        break;
    i++;
}

if (i < numbers.length)
    System.out.println("Key is found in the array");
else
    System.out.println("Key is NOT!");
```
Object-Oriented Paradigm

- Centered on the concept of the object
- Object

- Is data with methods
  - Data (attributes) can be simple things like number or character strings, or they can be other objects.
  - Defines things that are responsible for themselves
    - Data to know what state the object is in.
  - Method (code) to function properly.
What is an Object?

- Informally, an object represents an entity which is either physical, conceptual or software.

- Physical entity

- Conceptual entity

- Software entity
Basic Principles of Object Orientation

Object Orientation

Abstraction
Encapsulation
Modularity
Hierarchy
What is Abstraction?

Abstraction is one of the fundamental ways that we as humans cope with complexity.

Dahl, Dijkstra, and Hoare suggest that “abstraction arises from a recognition of similarities between certain objects, situations, or processes in the real world, and the decision to concentrate upon these similarities and to ignore for the time being the differences”.

Salesperson

Not saying

which salesperson

just a salesperson in general!

Customer

Product
What is Abstraction?

Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.
What is Encapsulation?

- Hide implementation from clients
  - Clients depend on interface

Information Hiding:
How does an object encapsulate? What does it encapsulate?

Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.
What is Encapsulation?

Encapsulation hides the details of the implementation of an object.
What is Modularity?

- The breaking up of something complex into manageable pieces.
What is Hierarchy?

Elements at the same level of the hierarchy should be at the same level of abstraction.
What is Really an Object?

- Formally, an object is a concept, abstraction, or thing with sharp boundaries and meaning for an application.

- An object is something that has:
  - **State** (property, attribute)
  - **Behavior** (operation, method)
  - **Identity**
Representing Objects

- An object is represented as rectangles with underlined names.

- **Class Name Only**: 
  - : Professor

- **Object Name Only**: 
  - ProfessorClark

- **Class and Object Name**: 
  - ProfessorClark : Professor

- **Equation**: 
  - $a + b = 10$
What is a Class?

- A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics.
  - An object is an instance of a class.

- A class is an abstraction in that it:
  - Emphasizes relevant characteristics
  - Suppresses other characteristics.
Example Class

Class
Course

Properties
Name
Location
Days offered
Credit hours
Start time
End time

Behavior
Add a student
Delete a student
Get course roster
Determine if it is full

\[ a + b = 10 \]
Representing Classes

- A class is represented using a compartmented rectangle

Professor Clark

Professor

\[ a + b = 10 \]
Class Compartments

- A class is comprised of three sections
  - The first section contains the **class name**
  - The second section shows the **structure** (attributes)
  - The third section shows the **behavior** (operations)
How Many Classes do you See?
Relationship between Classes and Objects

- A class is an abstract definition of an object
  - It defines the structure and behavior of each object in the class
  - It serves as a template for creating objects
- Objects are grouped into classes
The state of an object encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties.

State of an Object (property or attribute)
Behavior of an Object (operation or method)

- Behavior is how an object acts and reacts, in terms of its state changes and message passing.

![Diagram showing Class: CourseOffering with Operations: addStudent, deleteStudent, getStartTime, getEndTime]
Identity of an Object

- Each object has a **unique identity**, even if the state is identical to that of another object.

Professor “J Clark” teaches Biology

Professor “J Clark” teaches Biology
Sample Class: Automobile

- Attributes
  - manufacturer’s name
  - model name
  - year made
  - color
  - number of doors
  - size of engine

- Methods
  - Define attributes (specify manufacturer’s name, model, year, etc.)
  - Change a data item (color, engine, etc.)
  - Display data items
Sample Class: Circle

- Attributes
  - Radius
  - Center Coordinates
    - X and Y values

- Methods
  - Define attributes (radius and center coordinates)
  - Find area of the circle
  - Find circumference of the circle
Sample Class: Baby

- **Attributes**
  - Name
  - Gender
  - Weight
  - Decibel
  - # poops so far

- **Methods**
  - Get or Set specified attribute value
  - Poop
Sample Class: Person
Summary

- So far, we covered basics of objects and object oriented paradigm.
  - We tried to think in terms of objects.

- From now on, we should be seeing objects everywhere 😊
  - Or, we should be realizing that we were seeing objects everywhere already.
  - This is actually something you do naturally. Why not do programming that way?

- We will continue next week with actually creating objects by using Java.
Acknowledgments

- The course material used to prepare this presentation is mostly taken/adopted from the list below:
  - Ku-Yaw Chang, Da-Yeh University.