CS 170
Java Programming 1

Week 3: Objects, Input and Processing

Learning to Create Objects
Learning to Accept Input
Learning to Process Data
What’s the Plan?

- **Topic 1**: Working with Java Objects
  - Learning to create and control *Turtle* objects

- **Topic 2**: Processing and Input
  - Input, creating objects, creating variables
  - Processing, numbers, and calculations
  - Using *Math* functions for complex processing
Topic 1
Creating and Controlling Objects

Learn to Create Turtle Objects
Control Your Turtle with Methods
Creating a Simulation

- Computers let us **simulate** things
  - Buildings, factories, machines (rockets, cars), chips
- Do this by creating **software models** of the things we want to simulate and then writing programs to use them
- We need to define what **kinds** of objects we will want in our simulation and **what they can do**
  - Classes define the types (data and behavior)
  - Instances of the class (objects) act in the simulation
“The computer is the Proteus of machines. Its essence is its universality, its power to simulate. Because it can take on a thousand forms and serve a thousand functions, it can appeal to a thousand tastes.”

-- Seymour Papert in *Mindstorms*
History of Turtles

- Seymour Papert at MIT in the 60s
  - Developed the Logo language
  - Focused on teaching children

- By teaching the computer to do something kids are thinking about thinking
  - Develop problem solving skills
  - Learn by constructing and debugging something
  - Learn by making mistakes and fixing them
Using Turtles

- The **Turtle** class we'll use is part of several classes created at Georgia Tech called the **MediaComp** classes
  - I've already added these to our version of DrJava
  - You don't need the "book-classes" from the text
Creating Objects

- To create objects we ask a class to give us a new instance or object.
- Class is like a cookie cutter.
  - It knows how much space each object needs (shape).
- Many objects can be created from the same class.
Class as Object Factory

- We ask a class to create an object by using the keyword:
  \[
  \text{new ClassName}(\text{value}, \text{value}, ..., \text{...})
  \]

- We can also ask the class to initialize the object
  - May pass arguments (data) to help initialize it
Use Word or OpenOffice to create a new document
  - Save the file as **IC03.doc** (Office 97-2003 compatible)
  - Place on your network U: drive or on a thumb drive
  - Put your name and today's date at the top of the sheet
  - Title it "CS 170 Lab Exercises Week 3"

**Exercise 3.1**: start DrJava
  - In the Interactions Pane type `new World()`
  - Press Enter and shoot a screen-shot of the output
Creating Objects in Java

- **Turtle** objects live inside a **World** object
  - Means we must create a **World** object first
  - Typing `new World()` creates a new **World** object
  - But you don't have any way to refer to it again

- We need a way to refer to the new object so we can send messages to it; we need to **name** or **identify** it!
Naming is Important

- When you get a new pet the first thing you do is **name** it
  - Lets you refer to the new pet without saying: "Please take that dog we got yesterday for a walk."
  - Instead, you can say "Please take Fifi for a walk."

- In programming we name things we want to refer to again
  - Gives us a way to work with them
  - Like the `world` object

- This is called **declaring a variable**

- You must **declare** a variable **before** you use it
Declaring a Variable

- To declare a variable we need to specify what kind or type of thing it is and give it a name
  - Type name; OR
  - Type name = value; OR
  - Type name = new ClassName(v1, v2, ...);

- The equal sign doesn’t mean equal
  - Copies "stuff" on the right into the variable on the left

- Let's look as some examples
Declaring Variables

- **Exercise 3.2**: create these variables and snap a pic.
- Specify the **type** and a **name**
  - `String name;`
  - `World world1;`
  - `int x;`
- Can also give a starting value when you declare it
  - `String fullName = "Stephen Gilbert";`
  - `World world2 = new World();`
  - `int y = 0;`
Variables as "Memory Boxes"

- A variable is like a **box** with a **label** on it. You can:
  - put something **into** a box
  - take something **out of** a box
  - **change** what is in the box
- The size of the box restricts what you can store in it
Object and Primitive Variables

- Primitive variables allocate space based on their size
  - The space is set to the variable’s value

- Object variables allocate space for a reference to an object
  - The variable contains a "serial number" that can be used to locate the actual object in memory.

```java
int a = 3
String s = "Hi";
```
Limits on Declaring Variables

- You can't declare two variables with the same name!
  ```java
  > World earth = new World();
  > World earth = new World();
  Error: Redefinition of 'earth'
  ```

- You can change what an object variable refers to
  ```java
  > World earth = new World();
  > earth = new World();
  ```

- Exercise 3.3: assign values to `world1`, `name` and `x`. 
Create Some Turtles

- **Turtles** must live **inside** a **World** object
  - Supply the **World** when you create the **Turtle**

- **Exercise 3.3**: Create a **Turtle** object named **tim**
  - `tim = new Turtle(world1);`
  - Places `tim` inside the `world1` object; snap a pic

- **Exercise 3.4**: Create a second **Turtle** named **bill**
  - Place `bill` inside the `world2` object. Snap a pic.

- **Exercise 3.5**: Create a third **Turtle** named **sally**
  - Place `sally` inside `world2` along with `bill` then snap
What Can Turtles Do?

- **Turtles** can go forward: `tim.forward();`
  - Tell them how far to move: `sally.forward(50);`
- Can turn left or right 90 degrees: `bill.turnLeft();`
- **Turtles** can also turn like this: `tim.turn(50);`
  - A negative number turns counter-clockwise
- Can put their pen up or down: `sally.penUp();`
- Move to a specific location: `bill.moveTo(25, 50);`
  - First number x, second y, 0,0 in upper-left corner
**Drawing a T**

- **Exercise 3.6**: use a turtle to draw a large letter T

- **Process**
  - Ask the *Turtle* object to go forward 100
  - Ask the *Turtle* object to pick up the pen
  - Ask the *Turtle* object to turn left
  - Ask the *Turtle* object to go forward 25
  - Ask the *Turtle* object to turn 180 degrees
  - Ask the *Turtle* object to put down the pen
  - Ask the *Turtle* object to go forward 50

- Snap a picture when finished.
Topic 2
Input and Interactive Programs

Learn to Create
Interactive Programs
Interactive Programs

- So far, our programs have been **static**, not **dynamic**
  - They work exactly the same each time they run
- We make our programs dynamic or interactive by having them **process input** supplied by the user
  - Input can be commands (mouse, menu or keyboard)
  - Input can also be data, which we'll store in variables
- The **basic pattern** for interactive programs:
  - Ask the user to enter some data (called a prompt)
  - Create a variable to hold the user's input
  - Read the data and store it in the variable
The Greeter Program

- Our first program will be a dynamic variation of last week.
- **Exercise 3.7**: open `Greeter.java` in your `week03` folder.
  - Add your name, date; compile and run.
  - Then, run **Check** and snap a picture

![Image showing a console output of the program](image_url)
For **Greeter**, we need a **variable** to hold the user's name when they supply it.

- What type should it be?
- Go ahead and add the variable

**Scanner** reads numbers and strings from the keyboard

- Have to create a **Scanner**, just like creating a **Turtle**

```java
Scanner cin = new Scanner(System.in);
```

- Go ahead and add this **Scanner** variable as well.
Scanner Import Statements

- Compile your program. What happens?
  - Package containing Scanner class is not automatically loaded when program starts

- Use import statement to add Scanner to program
  ```java
  import java.util.Scanner;
  ```

- Put at very top of your program.
  - In Interactions Pane, anywhere before you use it
Reading the Input

- **Scanner** objects know how to read different types
  - Variable must be the **right type** to hold the result
  - `fullName = cin.nextLine(); // whole line`
  - `word = cin.next(); // one word`
  - `x = cin.nextInt(); // an integer`
  - `a = cin.nextDouble(); // double`

- Place right after the prompt that asks the user for input.
  - Remove "dummy data"; change `println` to `print`

- **Exercise 3.8**: run and shoot a screen-shot
Introducing Processing

- We now have input and output, but no processing
- Processing means converting input to output
  - Can involve arithmetic or other operations
  - In our program we want to change the output so that the user's name becomes part of the greeting
- Remember + concatenates or "pastes" strings together
- Exercise 3.9: complete the output line in Greeter, run the program and then run Check. Fix any errors and paste a screen-shot into your IC document.
CelsiusToFahrenheit

- Close Greeter and open CelsiusToFahrenheit
- As you can see, this is very similar:
  - Have a **prompt** asking the user to enter a temperature
  - Use the Celsius **input** to calculate the Fahrenheit **output**
  - Combine the calculated value with the output text

Enter a temperature in Celsius: 100
The temperature is 212.0 in Fahrenheit
Introducing Numbers

- Six of Java's eight built-in **primitive types** are numbers
- **Two families**: integers and floating-point
  - Integers are whole numbers
  - Real (floating-point) can have a fractional part
- Each family has **different sizes**:
  - Integer has four: `byte`, `short`, `int`, `long`
  - Real has two: `float` and `double`
- We'll usually use the `int` type for integers and the `double` type for real numbers
**Variables and Input**

- **Exercise**: create variables for both input and output
  - Both should be type `double`
  - Name them `celsius` and `fahrenheit`
  - Note that by convention, variables use `camelCase`
- **Exercise**: create a `Scanner` variable to read the input
- **Exercise**: read the input value using your `Scanner`
  - Make sure it appears after the prompt
  - Change the prompt and remove the dummy data
- **Exercise 3.10**: run and shoot a screen-shot
Processing the Data

- How do we convert the Celsius temperature that the user entered into the Fahrenheit temperature?
- Here's the formula for calculating Fahrenheit:
  \[ F = \left( \frac{9}{5} \right) C + 32 \]
- We need to "translate" that into a Java expression.
- An expression is just a calculation that produces a value which can then be used or stored in a variable.
- Let's try this out in the Interactions Pane.
What Are Expressions?

- An arithmetic expression uses operands (values) and operators (+, -, *, /, %) to produce a numeric value
  - (3 + 4) * 5 - 67 / 8
  - 3.141592 * 5.0 * 5.0

- In the Interactions Pane, you can type an expression and when you press Enter, DrJava will evaluate and print it.
Interactive Arithmetic

- In the Interactions Pane evaluate the following values
  - subtract 7 from 9
  - add 7 to 3
  - divide 3 by 2
  - multiply 5 by 10
  - find the remainder when 10 is divided by 3

- **Exercise 3.11**: shoot a screen-shot of the interaction

- Evaluation order: parentheses, multiplicative, additive
Calculating Fahrenheit

Let's try this in the Interactions Pane

- Create two `double`
- variables named `c` and `f`
- Set `c` to `100`
- Use the formula here to assign a value to `f`
- Display the value of `f`

Why don't we get the right answer?

- `(9 / 5)` is `1` because `9` and `5` are integers
Integer Division Operations

- Integer division works like primary-school long division:
  - The quotient is calculated, remainder discarded.
  - The result, an integer, is truncated, not rounded.
  - The remainder operator returns the “remainder” part.
    - 12 % 7 is 5: 7 “goes into” 12 once, with 5 left over.
- Algebra formulas (like you’ll normally work with) assume fractions are real numbers.

Exercise 3.11: fix CelsiusToFahrenheit so that it passes all tests. Paste screenshot into IC03.
Using the Math Class

- Java's **Math** class is used for complex calculations
  - Part of the **java.lang** package so you don't import it

- **Methods for basic numeric operations**
  - **Constants**: `Math.E` and `Math.PI`
  - **Square root**: `Math.sqrt(double)`
  - **Raise a number to a power**
    - `Math.pow(double a, double b) a^b`
Finishing Up

To complete your IC "lab" document:
  - Complete Exercise 3.12 in the Processing Data lesson and the Exercises 3.13 at the end of the Math Functions lesson.
  - Submit to Blackboard when done

Reading for next week:
  - Mediacomp: finish Chapter 3 and start Chapter 4
  - Gaddis: finish Chapter 1 start Chapter 2
Homework and Quiz

- Homework, Quiz and Assignments
  - Deadline is Monday at noon
  - Four homework problems
    - Upload to Assignment Dropbox.
    - Make sure to press Submit

- Proficiency Quiz 2: Basic IPO Problems
  - Practice on IPO assignments (TipCalc, CircleStats)
  - You will have ½ hour, assigned seating
Additional Credits

Some material adapted from instructor materials created by B. Ericson to accompany *Introduction to Computing and Programming with Java: A Multimedia Approach* by M. Guzdial and B. Ericson, released under a Creative Commons 3.0 Attribution License.