Logistics

Reading from Eck

- Ch 5 on Objects/Classes
- Ch 5.1-2 specifically

Goals

- Simple Objects
- Arrays of objects

Lab08: Simple Object/Methods

- Stock object
- Methods in same java file

Project 4

- In the works
- Deadline will be adjusted for later release
Latin for: All the same VS Different

<table>
<thead>
<tr>
<th>Homogeneous Data</th>
<th>Heterogeneous Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ All same data type</td>
<td>▶ Data types different</td>
</tr>
<tr>
<td>▶ Single name, multiple <strong>ints</strong>, multiple <strong>doubles</strong>, etc.</td>
<td>▶ Single name, multiple values in an combination</td>
</tr>
<tr>
<td>▶ Usually indexed by element number (4th elem, 9th elem)</td>
<td>▶ Example: need 1 int, 1 double, 2 booleans</td>
</tr>
<tr>
<td>▶ Example: <strong>arrays</strong>, collection of the same thing (<strong>homogeneous</strong>)</td>
<td>▶ Usually indexed by <strong>field name</strong> as in</td>
</tr>
<tr>
<td>▶ Elements accessed via array[index]</td>
<td><code>myStudent.gpa = 3.91;</code>    <code>myStudent.name = &quot;Sam&quot;;</code></td>
</tr>
<tr>
<td><strong>Before and Now</strong></td>
<td>▶ Example: <strong>classes/objects</strong> in Java, grouped data</td>
</tr>
<tr>
<td>▶ Discussed arrays and <strong>Strings</strong> previously (<strong>homogeneous</strong>)</td>
<td></td>
</tr>
<tr>
<td>▶ Now dealing with <strong>classes/objects</strong> (<strong>heterogeneous</strong>)</td>
<td></td>
</tr>
</tbody>
</table>
Java Classes

- .java files define classes
- Classes serve 2 purposes
  1. Group related code together (binary/decimal conversions, Collatz methods, etc.)
  2. Define new reference types
- Mixture of purposes can cause confusion but is somewhat intentional
- Most of the time Methods and Data Types are tied together
Simple Objects

- Java classes define new reference types by specifying **fields**
  
  // Represent data associated with an omelet.
  public class Omelet{
      public int eggs; // How many eggs in the omelet
      public int cheese; // How many ounces of cheese
      public double cookedFor; // How long the omelet has cooked
      public String extras; // Extra ingredients added
  }

- Fields are data data associated with the an **object** which as instance of the class. Every instance has its own fields

- **Instances** of objects are created with **new** and a constructor
  
  Omelet om = new Omelet();

- Field values can be accessed/changed using **dot notation** (if they are public)
  
  om.eggs = 5;
  int e = om.eggs;
Simple Objects Demo

// Represent data associated with an omelet.
public class Omelet{
    public int eggs;  // How many eggs in the omelet
    public int cheese;  // How many ounces of cheese
    public double cookedFor;  // How long the omelet has cooked
    public String extras;  // Extra ingredients added
}

public class OmeletMain{
    public static void main(String[] args){
        Omelet om = new Omelet();
        om.eggs = 3;
        om.cheese = 4;
        om.cookedFor = 4.5;
        om.extras = "ham";
        System.out.printf("Omelet has %d eggs\n",om.eggs);
        System.out.printf("Omelet has %d oz cheese\n",om.cheese);
        System.out.printf("Omelet cooked for %.1f minutes\n",om.cookedFor);
        System.out.printf("Omelet has these extras: %s\n",om.extras);
    }
}

> javac OmeletMain.java
> java OmeletMain
Omelet has 3 eggs
Omelet has 4 oz cheese
Omelet has these extras: ham
Omelet cooked for 4.5 minutes
Multiple Objects

- Each instance of class is its own object, has its own memory
- Can create as many omelets as you have memory for

```java
public class TwoOmelets {
    public static void main(String args[]) {
        Omelet om = new Omelet();
        om.eggs = 3;
        om.cheese = 4;
        om.cookedFor = 4.5;
        om.extras = "ham";

        Omelet coronary = new Omelet();
        coronary.eggs = 6;
        coronary.cheese = 10;
        coronary.cookedFor = 6.25;
        coronary.extras = "bacon";
    }
}
```
Exercise: Changes to memory

Draw changes to memory for commented lines 1-7

```java
public class TwoOmeletsExercise{
    public static void main(String args[]){
        Omelet om = new Omelet();
        om.eggs = 3;
        om.cheese = 4;
        om.cookedFor = 4.5;
        om.extras = "ham";

        Omelet coronary = new Omelet();
        coronary.eggs = 6;
        coronary.cheese = 10;
        coronary.cookedFor = 6.25;
        coronary.extras = "bacon";

        om.cookedFor += 0.75; // 1
        coronary.eggs++; // 2
        om.extras = "turkey"; // 3
        coronary.extras = // 4
            coronary.extras + " sausage";
        Omelet x = coronary; // 5
        x.cookedFor += 1.0; // 6
        om = new Omelet(); // 7
    }
}
```
Static Methods for Objects

- Classes/Objects typically have methods associated with them
- static methods follow the same conventions for objects as previously established for other things like arrays and ints
- Can change the object, return answers, call other methods

Exercise: OmeletOps

- OmMethods.java contains code which operates on Omelets
- OmeletOps.java has a main() method which uses those methods
- Show the output of running the main()
OmMeth: Omelet Methods

// Static methods to work on Omelet objects
public class OmMeth{
    // Cook the omelet for given time
    public static void cookFor(Omelet om, double time){
        om.cookedFor += time;
    }
    // Add an egg to the omelet
    public static void addEgg(Omelet om){
        om.eggs++;
    }
    // Calculate the calorie count of the basic omelet ingredients.
    public static int getCalories(Omelet om){
        int EGG_CALORIES = 94;
        int CHEESE_OZ_CALORIES = 114;
        return om.eggs * EGG_CALORIES +
               om.cheese * CHEESE_OZ_CALORIES;
    }
    // Determine if consumption of the given omelet is risky
    public static boolean foodPoisoningIminent(Omelet om){
        return om.cookedFor < (1.0 * om.eggs);
    }
    // Return a reference to the omelet x or y that has more calories
    public static Omelet richerOmelet(Omelet x, Omelet y){
        int xCal = getCalories(x);
        int yCal = getCalories(y);
        if(xCal > yCal){
            return x;
        } else{
            return y;
        }
    }
}
OmeletOps: main() method

// Demonstrate use of OmMeth methods for omelets
// Show output at the commented lines
public class OmeletOps{
    public static void main(String args[]){
        Omelet standard = new Omelet();
        standard.eggs = 2;
        standard.cheese = 4;
        standard.cookedFor = 0.0;
        standard.extras = "ham";

        OmMeth.addEgg(standard);
        OmMeth.cookFor(standard, 2.5);

        int calories = OmMeth.getCalories(standard);
        boolean safe = OmMeth.foodPoisoningIminent(standard);

        System.out.printf("Has %d eggs\n", standard.eggs); // 1
        System.out.printf("Has %d calories\n", calories); // 2
        System.out.printf("Safe to eat? %b\n", safe); // 3

        Omelet coronary = new Omelet();
        coronary.eggs = 6;
        coronary.cheese = 10;

        Omelet richer = OmMeth.richerOmelet(standard, coronary);
        System.out.printf("Richer has %d eggs\n", richer.eggs); // 4
    }
}

Compile and Run

> javac OmeletOps.java
> java OmeletOps

Has 3 eggs
Has 738 calories
Safe to eat? true
Richer has 6 eggs
Distributing Code Across Multiple Files

- Complex problems involve many parts
- Code to solve them is often distributed into many .java files
  - Each class in its own .java file
  - Extra helper methods somewhere in another file
  - `main()` method in its own class and .java file
- Organization and Navigation of files will start to become more important
Class Definition and Methods in One File

- Commonly the case that the definition of a class and the methods that operate on it are stored in the same .java file
- For Omelets, file would be something like the following
- This is the structure we will often use, labs and projects

```java
public class Omelet{
    // FIELDS
    public int eggs; // How many eggs in the omelet
    public int cheese; // How many ounces of cheese
    public double cookedFor; // How long the omelet has cooked
    public String extras; // Extra ingredients added

    // STATIC METHODS

    // Cook the omelet for given time
    public static void cookFor(Omelet om, double time){
        om.cookedFor += time;
    }
    // Add an egg to the omelet
    public static void addEgg(Omelet om){
        om.eggs++;
    }
    ...
}
```
NullPointerExceptions

Mistakes with initializing object references often lead to Exceptions

Code

// Demonstrates a NullPointerException with Omelets
public class NullOmelet{
    public static void main(String args[]){
        Omelet anOm = new Omelet();
        anOm.eggs = 2;
        anOm.cheese = 4;
        Omelet noOm = null;

        Omelet richer = Omelet.richerOmelet(anOm, noOm);
        System.out.printf("Richer has %d eggs\n", richer.eggs);
    }
}

Result: Exception and Stack Trace

> javac NullOmelet.java
> java NullOmelet
Exception in thread "main" java.lang.NullPointerException
    at Omelet.getCalories(Omelet.java:24)
    at Omelet.richerOmelet(Omelet.java:34)
    at NullOmelet.main(NullOmelet.java:9)
Quick Exercise: Identify errors

What seems to be the trouble below?

```java
1 public class OmeletException{
2    public static void main(String args[]){
3        Omelet om = new Omelet();
4        int len = omextras.length();
5        System.out.printf("Extra ingredients as \%d chars\n",len);
6    }
7 }
```

> javac OmeletException.java
> java OmeletException
Exception in thread "main" java.lang.NullPointerException
at OmeletException.main(OmeletException.java:4)
Answer: Identify errors

```
1 public class OmeletException{
2   public static void main(String args[]){
3       Omelet om = new Omelet();
4       int len = om.extras.length();
5       System.out.printf("Extra ingredients as %d chars\n",len);
6   }
7 }

> javac OmeletException.java
> java OmeletException
Exception in thread "main" java.lang.NullPointerException
at OmeletException.main(OmeletException.java:4)

- Unless set, all fields of om have default values
- Default value for String extras is null
- Cannot invoke a length() method on a null pointer: exception
- Fix with something like
  Omelet om = new Omelet();
  om.extras = "butter";
  int len = om.extras.length();
```
Reading Stack Traces

- Basic debugging skills will become essential
- Stack traces in Java provide information on where an error finally causes a problem
- Frequently this is not where the error originates but reading a stack trace gives someplace to begin the search

```java
> java NullOmelet
Exception in thread "main"
java.lang.NullPointerException  Exception causing failure
  at Omelet.getCalories(Omelet.java:24)  Active method, line num of exception
  at Omelet.richerOmelet(Omelet.java:34)  Next oldest active method
  at NullOmelet.main(NullOmelet.java:9)  Oldest active method:line num
```
Stack Traces During Testing in DrJava

- When test fails, DrJava can report stack traces
- Look for the "Stack Trace" button in the lower right corner

![Stack Trace Example](image)
Exercise: Arrays of Objects

- Creating arrays of objects has the same feel as primitives
- One key difference: initial contents of new arrays are the default value for objects: null

Array setup bug

This code creates an array of Omelet objects but has an error so that the final printing loop will cause an exception. Find the bug and draw some pictures to support your diagnoses.

```java
// Demonstrate assigning objects (Omelets) to an array of objects. This program has a bug.
public class OmeletArray{
    public static void main(String args[]){
        Omelet omArray[] = new Omelet[5];
        omArray[0] = new Omelet();
        omArray[0].eggs = 2;
        omArray[0].cheese = 4;
        omArray[0].extras = "ham mushroom";

        omArray[1] = new Omelet();
        for(int i=0; i<4; i++){
            Omelet.addEgg(omArray[1]);
        }
        omArray[1].cheese = 3;
        omArray[1].cookedFor = 1.0;

        for(int i=3; i<omArray.length; i++){
            omArray[i] = new Omelet();
            omArray[i].eggs = i;
            omArray[i].cheese = i/2;
            omArray[i].extras = "";
        }

        for(int i=0; i<omArray.length; i++){
            System.out.printf("Om: %d eggs, %d cheese,",
                                omArray[i].eggs, omArray[i].cheese);
            System.out.printf("
                                %.1f cooked, %s extras\n",
                               .omArray[i].cookedFor, omArray[i].extras);
        }
    }
}```
Answer: Array setup bug

- omArray elements are initially null, point at no objects/Omelets
- First two blocks set omArray[0] and omArray[1] to omelets
- Does not matter in this case that omArray[1].extras is null: will print as "null"
- omArray[2] is still null and attempting to access its fields in print loop will cause an exception

```
   1 // Demonstrate assigning objects (Omelets) to an
   2 // array of objects. This program has a bug.
   3 public class OmeletArray{
   4     public static void main(String args[]){
   5         Omelet omArray[] = new Omelet[5];
   6         omArray[0] = new Omelet();
   7         omArray[0].eggs = 2;
   8         omArray[0].cheese = 4;
   9         omArray[0].extras = "ham mushroom";
  10         omArray[1] = new Omelet();
  11         for(int i=0; i<4; i++){
  12             Omelet.addEgg(omArray[1]);
  13         }
  14         omArray[1].cheese = 3;
  15         omArray[1].cookedFor = 1.0;
  16         for(int i=3; i<omArray.length; i++){
  17             omArray[i] = new Omelet();
  18             omArray[i].eggs = i;
  19             omArray[i].cheese = i/2;
  20             omArray[i].extras = "";
  21         }
  22         for(int i=0; i<omArray.length; i++){
  23             System.out.printf("Om: %d eggs, %d cheese," ,
  24                 omArray[i].eggs, omArray[i].cheese);
  25             System.out.printf("%.1f cooked, %s extras\n" ,
  26                 omArray[i].cookedFor, omArray[i].extras);
  27         }
  28     }
  29 }
```
Things that should bug smart, lazy programmers

Construction

*Isn't there a shorter way to create an omelet initially than this?*

```java
Omelet om = new Omelet();
om.eggs = 3;
om.cheese = 4;
...
```

Methods

*Do I really have to write* 
*Omelet.doSomething()* *for every method?*

```java
Omelet.addEgg(om);
Omelet.cookFor(om);
```

Safety and Consistency

*I can break omelets super easily!*

```java
om.eggs = -2;
Omelet.cookFor(om, 3.0);
om.cookedFor = 0;
```

*public and static*

*What’s the deal with these words we keep using with methods?*

*public static void addEgg(*...*)*

Answers coming soon

All of these are related. The fun just doesn’t stop.