CSCI 1103: Simple Objects

Chris Kauffman

Last Updated:
Mon Oct 30 14:35:25 CDT 2017

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

Homogeneous Data

- All same data type
- Single name, multiple `ints`, multiple `doubles`, etc.
- Usually indexed by element number (4th elem, 9th elem)
- Example: arrays, collection of the same thing (*homogeneous*)
- Elements accessed via `array[index]`

Heterogeneous Data

- Data types different
- Single name, multiple values in an combination
- Example: need 1 int, 1 double, 2 booleans
- Usually indexed by field name as in

  ```
  myStudent.gpa  = 3.91;
  myStudent.name = "Sam";
  ```
- Example: classes/objects in Java, grouped data

Before and Now

- Discussed arrays and `Strings` previously (homogeneous)
- Now dealing with classes/objects (heterogeneous)

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

```java
// 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
```

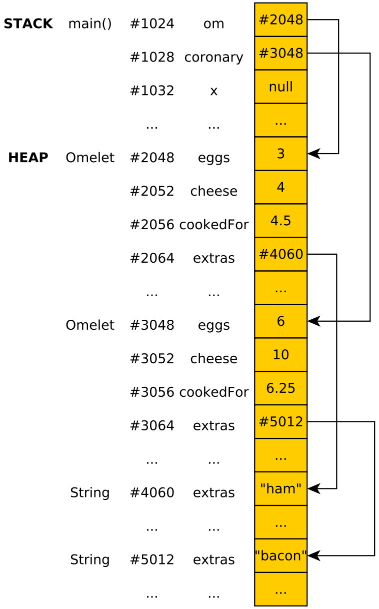
Multiple Objects

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

```
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";

  }
}
```



STACK	main()	#1024	om	#2048
		#1028	coronary	#3048
		#1032	x	null
		...	...	...
HEAP	Omelet	#2048	eggs	3
		#2052	cheese	4
		#2056	cookedFor	4.5
		#2064	extras	#4060
		...	...	...
	Omelet	#3048	eggs	6
		#3052	cheese	10
		#3056	cookedFor	6.25
		#3064	extras	#5012
		...	...	...
	String	#4060	extras	"ham"
		...	...	...
	String	#5012	extras	"bacon"
		...	...	...

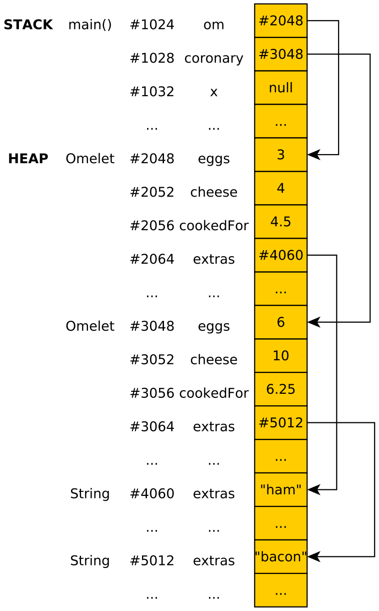
Exercise: Changes to memory

Draw changes to memory for commented lines 1-7

```
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 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

```
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?

```
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)
```

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)
```

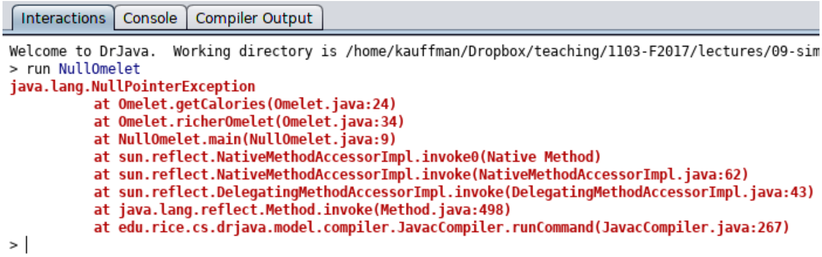
- Unless set, all fields of `om` have default values
- Default value for `String extras` is `null`
- Cannot invoke a `length()` method on a `null` pointer: exception
- Fix with something like

  ```
  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

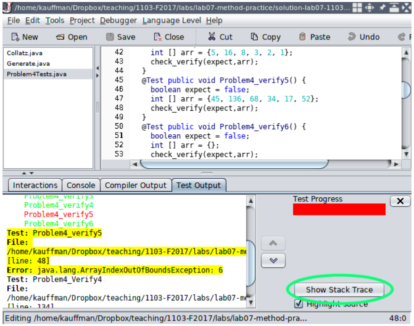
```
> java NullOmelet
Exception in thread "main"
 java.lang.NullPointerException           Exception causing failure
   at Omelet.getCalories(Omelet.java:24)  Active method, line num of exception
   at Omelet.richerOmelet(Omelet.java:34) Next oldest active method
   at NullOmelet.main(NullOmelet.java:9)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



```
Welcome to DrJava.  Working directory is /home/kauffman/Dropbox/teaching/1103-F2017/lectures/09-sim
> run NullOmelet
java.lang.NullPointerException
        at Omelet.getCalories(Omelet.java:24)
        at Omelet.richerOmelet(Omelet.java:34)
        at NullOmelet.main(NullOmelet.java:9)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at edu.rice.cs.drjava.model.compiler.JavacCompiler.runCommand(JavacCompiler.java:267)
> |
```

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

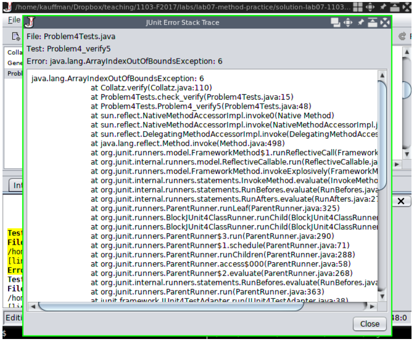


/home/kauffman/Dropbox/teaching/1103-F2017/labs/lab07-method-practice/solution-lab07-1103...
File Edit Tools Project Debugger Language Level Help
New Open Save Close Cut Copy Paste Undo
Collatz.java
Generate.java
Problem4Tests.java
Interactions Console Compiler Output Test Output
Test Progress
Show Stack Trace
Highlight source

```
    int [] arr = {5, 16, 8, 3, 2, 1};
    check_verify(expect,arr);
  }
  @Test public void Problem4_verify5() {
    boolean expect = false;
    int [] arr = {45, 136, 68, 34, 17, 52};
    check_verify(expect,arr);
  }
  @Test public void Problem4_verify6() {
    boolean expect = false;
    int [] arr = {};
    check_verify(expect,arr);
```

```
    Problem4_verify4
    Problem4_verify5
    Problem4_verify6
Test: Problem4_verify5
File:
/home/kauffman/Dropbox/teaching/1103-F2017/labs/lab07-me
[line: 48]
Error: java.lang.ArrayIndexOutOfBoundsException: 6
Test: Problem4_Verify4
File:
/home/kauffman/Dropbox/teaching/1103-F2017/labs/lab07-me
[line: 134]
```

```
Editing /home/kauffman/Dropbox/teaching/1103-F2017/labs/lab07-method-pra...    48:0
```



/home/kauffman/Dropbox/teaching/1103-F2017/labs/lab07-method-practice/solution-lab07-1103...
JUnit Error Stack Trace
Close

```
File: Problem4Tests.java
Test: Problem4_verify5
Error: java.lang.ArrayIndexOutOfBoundsException: 6

java.lang.ArrayIndexOutOfBoundsException: 6
        at Collatz.verify(Collatz.java:110)
        at Problem4Tests.check_verify(Problem4Tests.java:15)
        at Problem4Tests.Problem4_verify5(Problem4Tests.java:48)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.j
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(Framework
        at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.ja
        at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkM
        at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMetho
        at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java
        at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:2
        at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner
        at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
        at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
        at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
        at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
        at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
        at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java
        at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
        at junit.framework.JUnit4TestAdapter.run(JUnit4TestAdapter.java:38)
```

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"
- Loop sets `omArray[3]` and `omArray[4]` to omelets
- `omArray[2]` is still `null` and attempting to access its fields in print loop will cause an exception

```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);
    }
  }
}
```

Things that should bug smart, lazy programmers

Construction

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

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

Methods

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

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

Safety and Consistency

I can break omelets super easily!

```
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.