CS 211: Defining Classes

Chris Kauffman

Week 3-2
Logistics

P2: Instant Runoff Voting
- Can anyone explain?
- Class decomposition

Topics Today
- Strategies for VotingMachine methods
- Creating classes/objects (project)

Reading
- Building Java Programs Ch 8
- Lab Manual Ch 4 and 5

PracticeIt! BJP 3rd Ed
Exercises
- Ch 8 Exercise 18
- Ch 8 Exercise 20
- Ch 8 Exercise 21
- Ch 8 Exercise 22
Aggregate Data

- Define  Now there’s a type `bleh`, it looks like blah
- Declare Here is a variable, its type is `bleh`
- Assign  Element foo of variable bar gets value blip
- Access  Retrieve element foo of variable bar

Arrays
Create Homogeneous Aggregate Data
- Each constituent element is the same type
- Access via number index: \[ a[5] = \text{something}; \]

Classes
Define Heterogeneous Aggregate Data
- Constituent elements can be of different types
- Access via symbolic field name
  \[
  a.field1 = 1; \\
  a.Xfiled = "init!"; 
  \]
Basic Objects are Just Data

Omelets in SOmelet.java, no static fields

```java
public class SOmelet{
    public int eggs;
    public int ozCheese;
    public String extraIngredients;
    public double totalCookMinutes;
}

main(){
    SOmelet o = new SOmelet();
    o.eggs = 3;
    o.ozCheese = 4;
    o.extraIngredients = "";
    System.out.println("Cooked "+o.totalCookMinutes);
}
```
Exercise: One Class, Many Objects

Draw a Memory Diagram for the `main()` method below at the location indicated

```java
public class Somelet{
    public int eggs;
    public int ozCheese;
    public String extraIngredients;
    public double totalCookMinutes;
}

main()
{
    Somelet small = new Somelet();
    small.eggs = 2;
    small.ozCheese = 3;

    Somelet big = new Somelet();
    big.eggs = 4;
    big.ozCheese = 6;

    Somelet shallow = small;

    Somelet[] oa = new Somelet[5];
    for(int i=0; i<oa.length; i++){
        oa[i] = new Somelet();
        oa[i].eggs = i; oa[i].ozCheese = 2*i;
    }
    // Draw memory diagram HERE
}
```
Typically Want to do stuff with data

- static Methods defined in SomeletMethods.java
- Used in UseSomelet.java (excerpt below)

```java
// Create an omelet
Somelet standard = SomeletMethods.constructSomelet();
// Calculate calories
calories = SomeletMethods.getBaseCalories(standard);
// Cook an omelet
SomeletMethods.cookFor(standard, 4.0);
// Cooked long enough?
safe = !SomeletMethods.foodPoisoningIminent(standard);
```

Notice always invoking static method through SomeletMethods class (irritation)
Defining Static Methods on Objects

Take a reference to the object and do something with it; from SomeletMethods.java

// Cook an omelet for the given amount of time
public static void cookFor(Somelet thisOmelet, double cookMinutes){
    thisOmelet.totalCookMinutes += cookMinutes;
}

// Determine if consumption of the given omelet is risky
public static boolean foodPoisoningIminent(Somelet thisOmelet){
    return
    thisOmelet.totalCookMinutes < 1.0 * thisOmelet.eggs;
}

Notice reference thisOmelet is always required (irritation)
Remember: SOmelet is unconventional

SOmelet.java and SOmeletMethods.java are weird
  ▶ Don’t follow java convention
  ▶ Requires explicit reference thisSOmelet in all methods
  ▶ Precludes *dynamic dispatch* (next week)

However
Static method approach clearly separates
  ▶ Data versus Functions acting on data
Easier to build understanding from there because..

Standard Java
*Let’s mix data and functions together and season with this*
The "Normal" Way

- See OOOmelet.java
- No static methods or fields (except constants)
- Equivalent in most ways to S0melete.java + S0meletMethods.java

```java
public class OOOmelet{
    // No static fields
    public int eggs;
    public int ozCheese;
    public String extraIngredients;
    public double totalCookMinutes;

    // Constructors
    public OOOmelet(int eggs, int ozCheese){ ... }
    public OOOmelet(){...}

    // No static methods
    public void addIngredient(String ingredient){...}
    public void cookFor(double cookMinutes){...}
    ...
}
```
Methods

Discuss this: hidden parameter to method invocation

**Standard: O0Omelet**

```java
public void cookFor(double cookMinutes) {
    this.totalCookMinutes += cookMinutes;
}
```

```java
public boolean isBurned() {
    return this.totalCookMinutes > 2.0 * this.eggs;
}
```

```java
main() {
    O0Omelet oo = new O0Omelet();
    oo.cookFor(4.0);
}
```

**Static: S0melet**

```java
public static void cookFor(S0melet thisOmelet, double cookMinutes) {
    thisOmelet.totalCookMinutes += cookMinutes;
}
```

```java
public static boolean isBurned(S0melet thisOmelet) {
    return thisOmelet.totalCookMinutes > 2.0 * thisOmelet.eggs;
}
```

```java
main() {
    S0melet so = S0meletMethods.constructS0melet();
    S0meletMethods.cookFor(so, 4.0);
}
```
Constructors

Weird methods that build an object but don’t return it. **Compare:**

**Standard**

```java
public class OOOmelet{
    public OOOmelet(int eggs, int ozCheese){
        // No allocation
        this.eggs = eggs;
        this.ozCheese = ozCheese;
        this.extraIngredients = new String(""");
        this.totalCookMinutes = 0.0;
        // No return
    }
}
```

**Static**

```java
public class SSomeletMethods {
    public static SSomelet constructSSomelet(int eggs, int ozCheese){
        SSomelet thisSSomelet = new SSomelet();
        thisSSomelet.eggs = eggs;
        thisSSomelet.ozCheese = ozCheese;
        thisSSomelet.extraIngredients = new String(""");
        thisSSomelet.totalCookMinutes = 0.0;
        return thisSSomelet;
    }
}
```

**Error Checking**

Modify the constructor for OOOmelet to throw a RuntimeException if eggs or ozCheese is a negative number.
Exercise: To String, or Not To String.

That is not a question. 'Tis almost always better to endure writing a $\texttt{toString()}$ method that prints a pretty version of the object.

Write $\texttt{toString()}$ for $\texttt{OOOmelet}$

Welcome to DrJava.

> $\texttt{OOOmelet standard = new OOOmelet();}$
> $\texttt{System.out.println(standard.toString());}$
3 egg 4 oz cheese omelet, cooked for 0.0 minutes

> $\texttt{standard.cookFor(2.3)}$
> $\texttt{System.out.println(standard)}$
3 egg 4 oz cheese omelet, cooked for 2.3 minutes

> $\texttt{OOOmelet coronary = new OOOmelet(5,12);}$
> $\texttt{coronary.addIngredient("bacon");}$
> $\texttt{coronary.cookFor(4.6785)}$
> $\texttt{System.out.println(coronary)}$
5 egg 12 oz cheese omelet, cooked for 4.7 minutes
Don’t touch that

Java enables *Access Control* for insides of classes
- Visibility of fields and methods to other stuff
- public, protected, none, private
- Put them in front of methods and fields
- Play with these in 000melet
Access Modifiers

Access Levels for Fields/Methods by other stuff

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Class</th>
<th>Package</th>
<th>Subclass</th>
<th>World</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>protected</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>no modifier</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>private</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

- Mostly concerned with public and private, read about others on your own
- Most projects will specify required public methods, maybe public fields
- Most of the time you are free to create additional private methods and fields to accomplish your task

Official docs on access modifiers

http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html
Getter, Setter, Class Invariant

Common Java convention is to make all fields private and provide getter and setter methods to change them.

Getter/Setter for Eggs

```java
public class OOOmelet{
    public int eggs;
    public int ozCheese;
    ...
    public double getEggs(){
        return this.eggs;
    }
    public void setEggs(int e){
        if(this.totalCookMinutes > 0){
            throw new RuntimeException("yuck");
        }
        this.eggs = e;
    }
    ...
}
```

Questions

- Does it make sense to change the number of eggs after an omelet is cooked?
- Does it make sense to add `setCookMinutes(double)` to arbitrarily change `totalCookMinutes`?
- Why use getters/setters?
Typically Fields are private

POmelet: Private fields
Provide getters to report fields like eggs and cook time

```java
public class POmelet{
    private int eggs;
    private double totalCookMinutes;
    ...
    public double getEggs(){
        return this.eggs;
    }
    public double getTotalCookMinutes(){
        return this.eggs;
    }
    ...
}
```

Use of Getters v. Private Fields

```java
POmelet x=new POmelet(3,4);
// Correct
int eggs = x.getEggs();
// Error
x.eggs = 5; // No such symbol
x.cookFor(2.5);
// Correct
if(x.getTotalCookMinutes() > 0.0){
    ...
}
// Error
if(x.totalCookMinutes > 0.0){
    ...
}
Why Getters vs. Public Fields

- Simple objects can probably have public fields, direct access
  - Don’t do this as you’ll be penalized on manual inspection
- Slightly more complex objects like O00melet might get away with public fields but would allow ..
  - "Uncooking" of omelets: o.totalCookMinutes = 0.0;
  - Add eggs after being cooked
  - POmelet with private fields prevents this
- Complex objects like Scanner must preserve invariants: different parts must agree with each other.
  - Changing one field might screw up another one
  - Deny direct access via private fields
  - Mutation methods like next() and setX(v) keep all fields synchronized

Abstraction Up and Down

Break a problem into smaller parts. Define public methods between those parts. Think about internal details for one part at a time. Recurse for subparts as needed.
Scope and this

Name resolution rules don’t always require use of keyword this

Using this

```java
class POmelet{
    private int eggs;
    private double totalCookMinutes;

    public int getEggs(){
        return this.eggs;
    }
    public void cookFor(double cookMinutes){
        this.totalCookMinutes += cookMinutes;
    }
}
```

Without this

```java
class POmelet{
    private int eggs;
    private double totalCookMinutes;

    public int getEggs(){
        return eggs;
    }
    public void cookFor(double cookMinutes){
        totalCookMinutes += cookMinutes;
    }
}
```
Exercise static Methods the Best Omelet

- static is stand-alone, independent shared by all objects
- Write code for bestOmelet(arr)

```java
public class OOOmelet{
    // Return the "best" omelet in an array; better omelets have higher calorie counts as reported by the o.getBaseCalories() method. If the array is empty, return null.
    public static OOOmelet bestOmelet(OOOmelet [] arr){...}
}
```

Welcome to DrJava.
> OOOmelet arr[] = {new OOOmelet(3,4), new OOOmelet(2,10),
                      new OOOmelet(8,2), new OOOmelet(3,3)};
> OOOmelet best = OOOmelet.bestOmelet(arr);
> best
2 eggs 10 oz cheese omelet, cooked for 0.0 minutes
> best.getBaseCalories()
1328
> OOOmelet empty[] = {};
> OOOmelet other = OOOmelet.bestOmelet(empty);
> other
null
Recall: Equality and ==

```java
main(){
    int li1=3, li2=3;
    boolean eq1 = (li1 == li2);  // T/F??

    Integer bi1 = new Integer(4);
    Integer bi2 = new Integer(4);
    boolean eq2 = (bi1 == bi2);  // T/F??

    OOOmelet om1 = new OOOmelet(3,4);
    OOOmelet om2 = new OOOmelet(3,4);
    boolean eq3 = (om1 == om2);  // T/F??
}
```

- Draw a memory diagram for the above main method
- Determine the values of eq1, eq2, eq3
x.equals(y) methods

- Provide a **deep** equality check of x to y
- What’s **deep vs shallow**?
- **All** objects have one... why?
- **Most** objects define their own
- Technical note: difference between

  ```java
  public boolean equals(Object other)
  public boolean equals(Omelet other)
  ```
Exercise: Equality of Omelets

public class POmelet{
    private int eggs; // How many eggs in the omelet
    private int ozCheese; // How many ounces of cheese
    private String extraIngredients; // Extra ingredients added
    private double totalCookMinutes; // How long the omelet has cooked

    // Define me
    public boolean equals(POmelet other){...}

    ▶ POmelet x and POmelet y
    ▶ x.equals(y) is true when
        1. x and y have equal eggs (int)
        2. and equal ozCheese (int)
        3. and equal extraIngredients (String)
    ▶ 1 and 2 are easy
    ▶ 3 is slightly trickier
    ▶ Write the equality method
        ▶ Remember that x will be this, y will be other
Note: `equals(..)` is a funky method

- All classes automatically have an `equals(Object o)` method due to inheritance
- Will discuss next week in detail, but all proper `equals(..)` methods following the pattern mentioned in the spec

```java
public class POmelet{
    private int eggs, ozCheese;
    private String extraIngredients;
    private double totalCookMinutes;
    public boolean(Object other){ // Compare to arbitrary object
        if ( ! ( other instanceof POmelet) ) {
            return false; // Not another omelet, can’t b equal
        }
        POmelet that = (POmelet) other; // Caste other to omelet
        return // check relevant fields equal
            this.eggs == that.eggs &&
            this.ozCheese == that.ozCheese &&
            this.extraIngredients.equals(that.extraIngredients);
        // && this.totalCookMinutes == that.totalCookMinutes;
    }
}
```