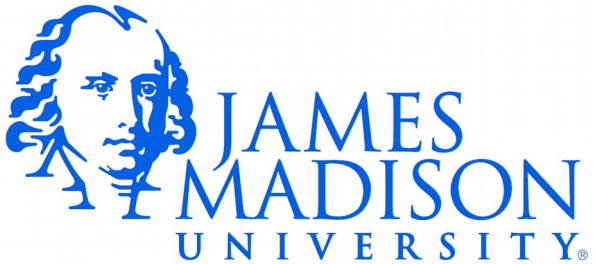
CS139 – static and this



Warm Up Question

- Here is a (functionally correct) solution to our earlier statistics lab.
- Any style problems?

```java
/**
 * Calculate the mean of a collection of doubles.
 *
 * @param numbers - The array of doubles
 * @return The mean, or Double.NaN if the array is empty or null
 */
public static double mean(double[] numbers) {
   double average = 0;
   double result;

   if (numbers == null || numbers.length == 0) {
      result = Double.NaN;
   } else {

      for (int i = 0; i < numbers.length; i++) {
         average += numbers[i];
      }
      result = average / numbers.length;
   }
   return result;
}
```

Warm Up Question

- Here is a (functionally correct) solution to our earlier statistics lab.
- Any style problems?

This variable does not store an average!

```
/**
 * Calculate the mean of a collection of doubles.
 *
 * @param numbers - The array of doubles
 * @return The mean, or Double.NaN if the array is empty or null
 */
public static double mean(double[] numbers) {
   double average = 0;
   double result;

   if (numbers == null || numbers.length == 0) {
      result = Double.NaN;
   } else {

      for (int i = 0; i < numbers.length; i++) {
         average += numbers[i];
      }
      result = average / numbers.length;
   }
   return result;
}
```

Warm Up Question

- Here is a (functionally correct) solution to our earlier statistics lab.
- Any style problems? Better.

```java
/**
 * Calculate the mean of a collection of doubles.
 *
 * @param numbers - The array of doubles
 * @return The mean, or Double.NaN if the array is empty or null
 */
public static double mean(double[] numbers) {
   double sum = 0;
   double result;

   if (numbers == null || numbers.length == 0) {
      result = Double.NaN;
   } else {

      for (int i = 0; i < numbers.length; i++) {
         sum += numbers[i];
      }
      result = sum / numbers.length;
   }
   return result;
}
```

this and static

- Let's look back at Cars.java ...

Why Are Instance Fields Private?

- The challenges of writing correct software:

  http://www.viddler.com/v/42c8494f (38:00)

  - This video was produced in 1992, OO programming became popular in the mid to late 90's

Why Are Instance Fields Private?

- The challenges of writing correct software:

  http://www.viddler.com/v/42c8494f (38:00)

  - This video was produced in 1992, OO programming became popular in the mid to late 90's

- encapsulation – Bundling data with the methods that act on that data
- data hiding – Preventing outside code from directly accessing data
  - (Terms are often used interchangeably.)

- Goal is to create classes that can be treated as black boxes
  - We don't need to know or care how they work to use them
  - We only need to understand the functionality they provide
- Related to the idea of minimizing coupling.
- In response to the video: we want software that *does* have proximity of cause and effect.

```java
import java.util.ArrayList;

public class Person {

   private String name;
   private ArrayList<Person> friends;

   public Person(String name) {
      this.name = name;
      friends = new ArrayList<Person>();
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public ArrayList<Person> getFriends() {
      return friends;
   }

   public void addFriend(Person newFriend) {
      friends.add(newFriend);
   }

   public String toString() {
      return "Person: " + name;
   }

}
```

- Coding a social media application "FriendFaces".
- Where is the flaw in our data hiding?

```java
import java.util.ArrayList;

public class Person {

   private String name;
   private ArrayList<Person> friends;

   public Person(String name) {
      this.name = name;
      friends = new ArrayList<Person>();
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public ArrayList<Person> getFriends() {
      return friends;
   }

   public void addFriend(Person newFriend) {
      friends.add(newFriend);
   }

   public String toString() {
      return "Person: " + name;
   }

}
```

- Coding a social media application “FriendFaces”.
- Where is the flaw in our data hiding?
- Here

```java
import java.util.ArrayList;

public class Person {

   private String name;
   private ArrayList<Person> friends;

   public Person(String name) {
      this.name = name;
      friends = new ArrayList<Person>();
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public ArrayList<Person> getFriends() {
      // Use ArrayList copy constructor!
      return new ArrayList<Person>(friends);
   }

   public void addFriend(Person newFriend) {
      friends.add(newFriend);
   }

   public String toString() {
      return "Person: " + name;
   }

}
```

- Coding a social media application "FriendFaces".
- Where is the flaw in our data hiding?
- Fixed!

Exercise...

- Create a copy constuctor for the Person class (create a new Person that is exactly like an existing Person.)

```
public Person(Person other) {




}
```

Exercise...

- Create a copy constuctor for the Person class (create a new Person that is exactly like an existing Person.)

OK???

```
public Person(Person other) {

    name = other.name;
    friends = other.friends;
}
```

Exercise…

- Create a copy constuctor for the Person class (create a new Person that is exactly like an existing Person.)

OK??? NO! Now we have two people who share a single friend list.

```
public Person(Person other) {

    name = other.name;
    friends = other.friends;
}
```

Exercise...

- Create a copy constuctor for the Person class (create a new Person that is exactly like an existing Person.)

OK???

```
public Person(Person other) {

    name = other.name;
    friends = new ArrayList<Person>(other.friends);
}
```

Exercise…

- Create a copy constuctor for the Person class (create a new Person that is exactly like an existing Person.)

OK??? Yes! Now the new person has a copy of the list of friends.

```
public Person(Person other) {

    name = other.name;
    friends = new ArrayList<Person>(other.friends);
}
```

Why don't we need to copy the name?
String *is* a reference type.

Exercise...

- Create a copy constuctor for the Person class (create a new Person that is exactly like an existing Person.)

OK??? Yes! Now the new person has a copy of the list of friends.

```
public Person(Person other) {

    name = other.name;
    friends = new ArrayList<Person>(other.friends);
}
```

Why don't we need to copy the name? String *is* a reference type. String is immutable. No way for it to be changed without my permission (or at all).

Exercise...

- Create a copy constuctor for the Person class (create a new Person that is exactly like an existing Person.)

OK???

```
public Person(Person other) {

    name = other.getName();
    friends = other.getFriends();

}
```

Exercise...

- Create a copy constuctor for the Person class (create a new Person that is exactly like an existing Person.)

OK??? Yes! This is the best solution. getFriends has already been coded to return a copy.

```
public Person(Person other) {

    name = other.getName();
    friends = other.getFriends();

}
```