Static Methods, Java Arrays
Chapters 20 & 21
Announcements

• HW6: Java Programming (Pennstagram)
  – Due: Tuesday the 20\textsuperscript{th} at 11:59pm
  – Note: Run JUnit tests (under Tools menu)

• Reminder: please complete mid-semester survey
  – See post on Piazza

• Upcoming: Midterm 2
  – Friday, March 23\textsuperscript{rd} in class
  – Coverage: mutable state, queues, deques, GUI, Java
A Little More on Interfaces

Working with objects abstractly
Review: Interfaces

• Give a type for an object based on how it can be used, not on how it was constructed
• Describe a contract that objects must satisfy
• Example: Interface for objects that have a position and can be moved

```java
public interface Displaceable {
    public int getX();
    public int getY();
    public void move(int dx, int dy);
}
```

No fields, no constructors, no method bodies!
A class that *implements* an interface provides appropriate definitions for the methods specified in the interface.

The class fulfills the contract implicit in the interface.

```java
public class Point implements Displaceable {
    private int x, y;
    public Point(int x0, int y0) {
        x = x0;
        y = y0;
    }

    public int getX() { return x; }
    public int getY() { return y; }
    public void move(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }
}
```
Using interface types

• Variables of an interface type can refer to objects of any class implementing the interface.
• Point, Circle, and ColoredPoint are all subtypes of Displaceable

```java
Displaceable d0, d1, d2;
d0 = new Point(1, 2);
d1 = new Circle(new Point(2,3), 1);
d2 = new ColoredPoint(-1,1, red);
d0.move(-2,0);
d1.move(-2,0);
d2.move(-2,0);
...
... d0.getX() ... ⇒ -1
... d1.getX() ... ⇒ 0
... d2.getX() ... ⇒ -3
The class that created the object value determines which move code is executed: dynamic dispatch
```
Abstraction

• The interface gives us a single name for all the possible kinds of “moveable things.” This allows us to write code that manipulates arbitrary Displaceable objects, without caring whether it’s dealing with points or circles.

```java
class DoStuff {
    public void moveItALot (Displaceable s) {
        s.move(3,3);
        s.move(100,1000);
        s.move(1000,234651);
    }

    public void dostuff () {
        Displaceable s1 = new Point(5,5);
        Displaceable s2 = new Circle(new Point(0,0),100);
        moveItALot(s1);
        moveItALot(s2);
    }
}
```
Colored interface

• Contract for objects that have a color
  – Circles and Points don’t implement Colored
  – ColoredPoints do

```java
public interface Colored {
    public Color getColor();
}
```
public class ColoredPoint implements Displaceable, Colored {

    Point center;
    private Color color;
    public Color getColor() {
        return color;
    }

    ...
}

“Datatypes” in Java

OCaml

```ocaml
type shape =
  | Point of ...
  | Circle of ...

let draw_shape (s:shape) =
  begin match s with
  | Point ... -> ...
  | Circle ... -> ...
  end
```

Java

```java
interface Shape {
    public void draw();
}

class Point implements Shape {
    ...
    public void draw() {
        ...
    }
}

class Circle implements Shape {
    ...
    public void draw() {
        ...
    }
}
```
Static Methods and Fields

functions and global state
Java Main Entry Point

```java
class MainClass {
    public static void main (String[] args) {
        ...
    }
}
```

- Program starts running at `main`
  - `args` is an array of `Strings` (passed in from the command line)
  - must be public
  - returns `void` (i.e. is a command)

- What does `static` mean?
How familiar are you with the idea of "static" methods and fields?

1. I haven't heard of the idea of "static".
2. I've used "static" before without really understanding what it means.
3. I have some familiarity with the difference between "static" and "dynamic".
4. I totally get it.
Static == Decided at *Compile Time*
Dynamic == Decided at *Run Time*

**Compile Time:**
- `javalc Foo.java`
- checks for *syntax* errors – is it possibly legal code
- typechecking – do the types make sense?
- compilation – produce `Foo.class`
- in Codio: "Build Project"

**Run Time:**
- `java Foo arg1 arg2 arg3`
- instantiates the abstract stack machine
- executes the code in `Foo.class` (assuming main exists)
public class Max {

    public static int max (int x, int y) {
        if (x > y) {
            return x;
        } else {
            return y;
        }
    }

    public static int max3(int x, int y, int z) {
        return max(max(x, y), z);
    }
}

public class Main {

    public static void main (String[] args) {
        System.out.println(Max.max(3, 4));
        return;
    }
}

Internally (within the same class), call with just the method name.

closest analogue of top-level functions in OCaml, but must be a member of some class.

main method must be static; it is invoked to start the program running.

Externally, prefix with name of the class.
Static vs. Dynamic Methods

• Static Methods are *independent* of object values
  – Similar to OCaml functions
  – Cannot refer to the local state of objects (fields or normal methods)

• Use static methods for:
  – Non-OO programming
  – Programming with primitive types: Math.sin(60), Integer.toString(3), Boolean.valueOf("true")
  – “public static void main”

• “Normal” methods are *dynamic*
  – Need access to the local state of the particular object on which they are invoked
  – We only know at runtime which method will get called

```java
void moveTwice (Displaceable o) {
    o.move (1,1); o.move(1,1);
}
```
Method call examples

- Calling a (dynamic) method of an object \( o \) that returns a number:
  \[
  x = o.m() + 5;
  \]

- Calling a static method of a class \( C \) that returns a number:
  \[
  x = C.m() + 5;
  \]

- Calling a method of that returns void:
  \[
  \begin{align*}
  \text{Static} & : & C.m(); \\
  \text{Dynamic} & : & o.m();
  \end{align*}
  \]

- Calling a static or dynamic method in a method of the same class:
  \[
  \begin{align*}
  \text{Either} & : & m(); \\
  \text{Static} & : & C.m(); \\
  \text{Dynamic} & : & this.m();
  \end{align*}
  \]

- Calling (dynamic) methods that return objects:
  \[
  \begin{align*}
  x &= o.m().n(); \\
  x &= o.m().n().x().y().z().a().b().c().d().e();
  \end{align*}
  \]
Which **static** method can we add to this class?

```java
public class Counter {
    private int r;

    public Counter () {
        r = 0;
    }

    public int inc () {
        r = r + 1;
        return r;
    }

    // 1,2, or 3 here?
}
```

1. **public static int dec () {**
   
   ```java
   r = r - 1;
   return r;
   ```

2. **public static int inc2 () {**
   
   ```java
   inc();
   return inc();
   ```

3. **public static int getInitialVal () {**
   
   ```java
   return 0;
   ```

Answer: 3
Recap: Static vs. Dynamic Class Members

```java
public class FancyCounter {
    private int c = 0;
    private static int total = 0;

    public int inc () {
        c += 1;
        total += 1;
        return c;
    }

    public static int getTotal () {
        return total;
    }
}
```

```java
FancyCounter c1 = new FancyCounter();
FancyCounter c2 = new FancyCounter();
int v1 = c1.inc();
int v2 = c2.inc();
int v3 = c1.getTotal();
System.out.println(v1 + " " + v2 + " " + v3);
```
Static Class Members

• Static methods can depend *only* on other static things
  – Static fields and methods, from the same or other classes

• Static methods *can* create *new* objects and use them
  – This is typically how `main` works

• `public static` fields are the "global" state of the program
  – Mutable global state should generally be avoided
  – Immutable global fields are useful: for constants like pi

```java
public static final double PI = 3.141592653589793238462643383279;
```
Style: naming conventions

<table>
<thead>
<tr>
<th>Kind</th>
<th>Part-of-speech</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>noun</td>
<td>RacingCar</td>
</tr>
<tr>
<td>field / variable</td>
<td>noun</td>
<td>initialSpeed</td>
</tr>
<tr>
<td>static final field</td>
<td>noun</td>
<td>MILES_PER_GALLON</td>
</tr>
<tr>
<td>(constants)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>method</td>
<td>verb</td>
<td>shiftGear</td>
</tr>
</tbody>
</table>

- Identifiers consist of alphanumerics and _ and cannot start with a digit
- The larger the scope, the more informative the name should be
- Conventions are important: variables, methods and classes can have the same name
Why naming conventions matter

```java
public class Turtle {
    private Turtle Turtle;
    public Turtle() {
    }

    public Turtle Turtle (Turtle Turtle) {
        return Turtle;
    }
}

Many more details on good Java style here:
http://www.seas.upenn.edu/~cis120/current/java_style.shtml
Java arrays

Working with static methods
Java Arrays: Indexing

• An array is a sequentially ordered collection of values that can be indexed in constant time.
• Index elements from 0

![Array diagram]

• Basic array expression forms
  
  \[ a[i] \] access element of array \( a \) at index \( i \)
  
  \[ a[i] = e \] assign \( e \) to element of array \( a \) at index \( i \)
  
  \( a\.length \) get the number of elements in \( a \)
Java Arrays: Dynamic Creation

- Create an array `a` of size `n` with elements of type `C`
  ```java
  C[] a = new C[n];
  ```
- Create an array of four integers, initialized as given:
  ```java
  int[] x = {1; 2; 3; 4};
  ```
- Arrays live in the heap; values with array type are mutable references:
  ```java
  int[] a = new int[4];
  a[2] = 7;
  ```

**Diagram:**

- **Stack**
  - `a`

- **Heap**
  - `int[]`:
    - `length`: 4
    - `0`: 0
    - `7`: 0
  - Array entries are mutable
  - Length is a `final` (immutable) field
Java Arrays: Aliasing

Variables of array type are references and can be aliases

```java
int[] a = new int[4];
int[] b = a;
a[2] = 7;
int ans = b[2];
```
What is the value of ans at the end of this program?

```java
int[] a = {1, 2, 3, 4};
int ans = a[a.length];
```

1. 1
2. 2
3. 3
4. 4
5. `NullPointerException`
6. `ArrayIndexOutOfBoundsException`

Answer: `ArrayIndexOutOfBoundsException`