Programming Languages and Techniques (CIS120)

Lecture 23
March 16th, 2016

Java: Static Methods & Arrays
Chapters 20 & 21
Announcements

• HW6: Java Programming (Pennstagram)
  – Available on the web
  – Due: Tuesday, March 29\textsuperscript{th} at 11:59pm

• Special guest lecture in class, Monday, March 28\textsuperscript{th}
  – Effective OCaml
  – Yaron Minsky, Jane Street Capital CTO

• Midterm 2
  – March 22\textsuperscript{nd}, 6:15-8:15PM, location TBA
  – Make-up exam, Wed March 23\textsuperscript{rd}, 9-11AM
  – Sign up for make-up exam on course website by March 20\textsuperscript{th}
Static Methods and Fields

functions and global state
Java Main Entry Point

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.
```java
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 module), call with just the method name

main method must be static, invoked to start the program running

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

Externally, call with name of the class
mantra

Static == Decided at Compile Time
Dynamic == Decided at Run Time
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 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:

Static \[ C.m(); \]
Dynamic \[ o.m(); \]

• Calling a static or dynamic method in a method of the same class:

Either \[ m(); \]
Static \[ C.m(); \]
Dynamic \[ this.m(); \]

• Calling (dynamic) methods that return objects:

\[
x = o.m().n();
\]
\[
x = o.m().n().x().y().z().a().b().c().d().e();
\]
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 getInitialValue ()** {
   
   ```java
   return 0;
   ```

Answer: 3
Static Field and Methods

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

```
public static final int PI = 3.14159265359793238462643383279
```
Identifiers consist of alphanumeric characters 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.

<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 (constants)</td>
<td>noun</td>
<td>MILES_PER_GALLON</td>
</tr>
<tr>
<td>method</td>
<td>verb</td>
<td>shiftGear</td>
</tr>
</tbody>
</table>
public class Turtle {
    private Turtle Turtle;
    public Turtle() { }

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

Many more details 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 \textit{constant} time.
- Index elements from 0

![Diagram of an array with indices and elements]

- Basic array expression forms
  \begin{align*}
  a[i] & \quad \text{access element of array } a \text{ at index } i \\
  a[i] = e & \quad \text{assign } e \text{ to element of array } a \text{ at index } i \\
  a.length & \quad \text{get the number of elements in } a
  \end{align*}
Java Arrays: Dynamic Creation

- Create an array `a` of size `n` with elements of type `C`:
  ```java
  C[] a = new C[n];
  ```
- Arrays live in the heap; values with array type are mutable references.

```java
int[] a = new int[4];
a[2] = 7;
```
Java Arrays: Initialization

```java
int[] myArray = { 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 };

String[] yourArray = { "foo", "bar", "baz" };

Point[] herArray = { new Point(1,3),
                   new Point(5,4) };

herArray = new Point[] { new Point(2,3),
                        new Point(6,5) };
```
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[0];
```

1. 1
2. 2
3. 3
4. 4
5. NullPointerException
6. ArrayIndexOutOfBoundsException
What is the value of ans at the end of this program?

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

1. 1
2. 2
3. 3
4. 4
5. NullPointerException
6. ArrayIndexOutOfBoundsException
What is the value of ans at the end of this program?

```java
int[] a = null;
int ans = a.length;
```

1. 1
2. 2
3. 3
4. 0
5. NullPointerException
6. ArrayIndexOutOfBoundsException
What is the value of ans at the end of this program?

```java
int[] a = {};
int ans = a.length;
```

1. 1
2. 2
3. 3
4. 0
5. NullPointerException
6. ArrayIndexOutOfBoundsException
What is the value of ans at the end of this program?

```java
int[] a = {1, 2, 3, 4};
int[] b = a;
b[0] = 0;
int ans = a[0];
```

1. 1
2. 2
3. 3
4. 0
5. NullPointerException
6. ArrayIndexOutOfBoundsException
What is the value of ans at the end of this program?

Counter[] a = { new Counter(), new Counter() };  
Counter[] b = a;  
a[0].inc();  
b[0].inc();  
int ans = a[0].inc();

1. 1  
2. 2  
3. 3  
4. 0  
5. NullPointerException  
6. ArrayIndexOutOfBoundsException
What is the value of ans at the end of this program?

```java
Counter[] a = { new Counter(), new Counter() };
Counter[] b = { new Counter(), new Counter() };
a[0].inc();
b[0].inc();
int ans = a[0].inc();
```

1. 1
2. 2
3. 3
4. 0
5. NullPointerException
6. ArrayIndexOutOfBoundsException

```
public class Counter {
    private int r;

    public Counter () {
        r = 0;
    }

    public int inc () {
        r = r + 1;
        return r;
    }
}
```
What is the value of ans at the end of this program?

```java
public class Counter {
    private int r;
    public Counter () {
        r = 0;
    }
    public int inc () {
        r = r + 1;
        return r;
    }
}
```

1. 1
2. 2
3. 3
4. 0
5. NullPointerException
6. ArrayIndexOutOfBoundsException