Programming Languages and Techniques (CIS120)

Lecture 23
October 26, 2018

Static Methods, Java Arrays
Chapters 20 & 21
Announcements

• HW6: Java Programming (Pennstagram)
  – Due: Tuesday, October 30 at 11:59pm

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

• Upcoming: Midterm 2
  – Friday, November 9th in class
  – Coverage: mutable state, queues, deques, GUI, Java
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:**
- `javac 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)
### Static method example

```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);
    }
}
```

```java
public class Main {
    public static void main (String[] args) {
        System.out.println(Max.max(3,4));
        return;
    }
}
```
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:
  
  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. ```java
   public static int dec () {
       r = r - 1;
       return r;
   }
```

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

3. ```java
   public static int getInitialVal () {
       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 (constants)</td>
<td>noun</td>
<td>MILES_PER_GALLON</td>
</tr>
<tr>
<td>method</td>
<td>verb</td>
<td>shiftGear</td>
</tr>
</tbody>
</table>

- 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
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

![Image of array indexing](image)

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

**Stack**

**Heap**

- 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;
it int ans = b[2];
```

<table>
<thead>
<tr>
<th>Stack</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>int[]</td>
</tr>
<tr>
<td>b</td>
<td>length</td>
</tr>
<tr>
<td></td>
<td>0 0 7 0</td>
</tr>
</tbody>
</table>
What is the value of `ans` at the end of this program?

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

Options:
- `NullPointerException`
- `ArrayIndexOutOfBoundsException`

Correct option:
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
What is the value of `ans` at the end of this program?

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

Answer: NullPointerException
What is the value of `ans` at the end of this program?

```java
int[] a = {}; // 1
int ans = a.length; // 2

// Possible exceptions:
NullPointerException // 3
ArrayIndexOutOfBoundsException // 4
```
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

Answer: 0
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];
```

Options:
- `NullPointerException`
- `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

Answer: 0
Array Iteration
For loops

**Initialization**

```
for (int i = 0; i < a.length; i++) {
```

**Loop body**

```
total += a[i];
```

**Loop condition**

```
i++
```

**Update**

```
}
```

---

**General pattern for computing info about an array**

```java
static double sum(double[] a) {
    double total = 0;
    for (int i = 0; i < a.length; i++) {
        total += a[i];
    }
    return total;
}
```
Multidimensional Arrays
Multi-Dimensional Arrays

A 2-d array is just an array of arrays...

```java
String[][][] names = {{{"Mr. ", "Mrs. ", "Ms. 
"},
{"Smith", "Jones"}}};

System.out.println(names[0][0] + names[1][0]);
    // --> Mr. Smith
System.out.println(names[0][2] + names[1][1]);
    // --> Ms. Jones
```

String[][][] just means (String[][])[]
names[1][1] just means (names[1])[][1]

More brackets → more dimensions
int[][] products = new int[5][];
for(int col = 0; col < 5; col++) {
    products[col] = new int[col+1];
    for(int row = 0; row <= col; row++) {
        products[col][row] = col * row;
    }
}

What would a “Java ASM” stack and heap look like after running this program?
Multi-Dimensional Arrays

```java
int[][] products = new int[5][];
for(int col = 0; col < 5; col++) {
    products[col] = new int[col+1];
    for(int row = 0; row <= col; row++) {
        products[col][row] = col * row;
    }
}
```

Note: This heap picture is simplified – it omits the class identifiers and length fields for all 6 of the arrays depicted. (Contrast with the array shown earlier.)

Note also that orientation doesn’t matter on the heap.
Demo

ArrayDemo.java
Design Exercise: Resizable Arrays

Arrays that grow without bound.
public class ResArray {

/** Constructor, takes no arguments. */
public ResArray() { … }

/** Access the array at position i. If position i has not yet
 * been initialized, return 0.
 */
public int get(int i) { … }

/** Modify the array at position i to contain the value v. */
public void set(int i, int v) { … }

/** Return the extent of the array. */
public int getExtent() { … }

}

Object Invariant: extent is 1 past the last nonzero value in data (can be 0 if the array is all zeros)