Programming Languages and Techniques (CIS120)

Lecture 22

October 23rd, 2015

Java: Static Methods & Arrays

Announcements

- HW6: Java Programming (Pennstagram)
  - Available on the web
  - Due: Thursday, October 29th at 11:59pm

- Midterm 2
  - Friday, November 6th
  - In class
  - Details to follow

Java Main Entry Point

```
class GUI {

    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 an 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 Methods and Fields

functions and global state

Static method example

```
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);
   }
}
```

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

Internally (within the module), call with just the method name

```
public class Main {

 public static void main (String[] args) {

       System.out.println(Max.max(3,4));
       return;
   }
}
```

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

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

```
void moveTwice (Displaceable o) {
   o.move (1,1); o.move(1,1);
}
```

Method call examples

- Calling a (dynamic) method of another object that returns a number:

```
x = o.m() + 5;
```

- Calling a static method of another class that returns a number:

```
x = C.m() + 5;
```

- Calling a method of another class that returns void:

Static

```
C.m();
```

Dynamic

```
o.m();
```

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

```
m();   x = m() + 5;
```

- 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?

```
public class Counter {

  private int r;

  public Counter () {
    r = 0;
  }

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

}
```

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

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

3.
```
public static int getInitialValue () {
    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

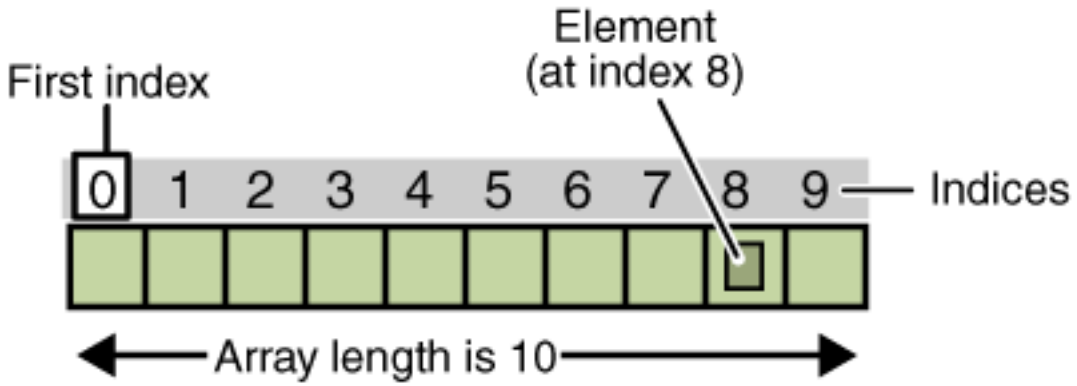
- Static methods *can* create *new* objects and use them
  - This is typically how `main` works

- `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

Java arrays

Java Arrays: Indexing

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



- 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`

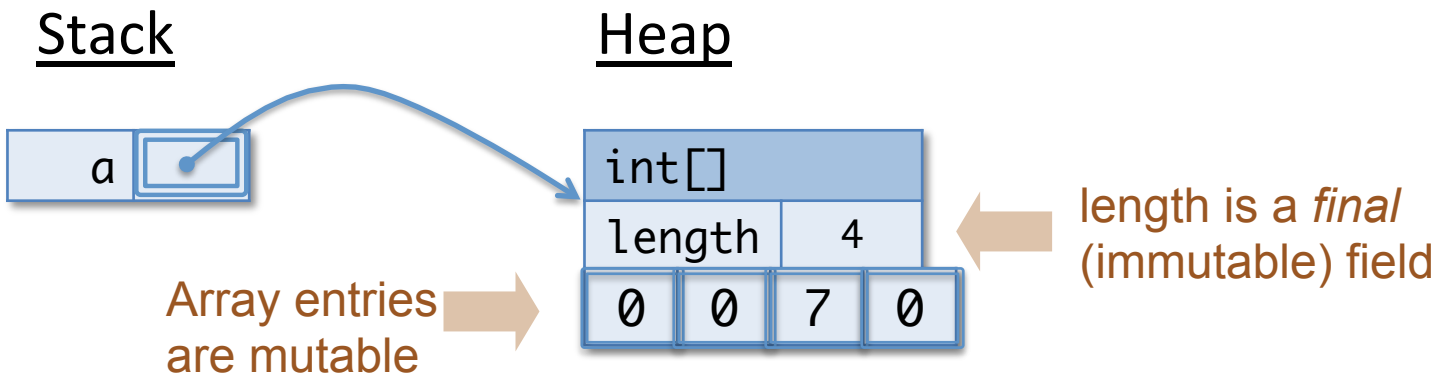
Java Arrays: Dynamic Creation

- Create an array a of size n with elements of type C

```
C[] a = new C[n];
```

- Arrays live in the heap; values with array type are mutable references

```
int[] a = new int[4];
a[2] = 7;
```



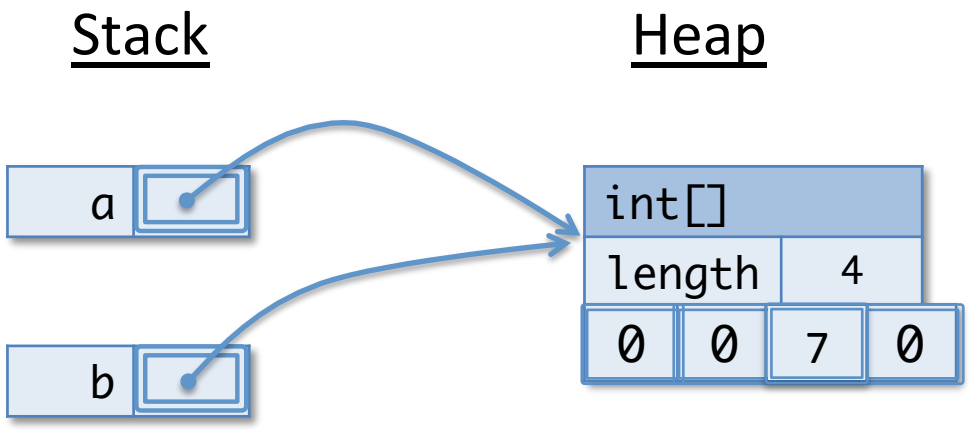
Java Arrays: Initialization

```
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

```
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?

```
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?

```
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?

```
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?

```
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?

```
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

```
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?

```
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?

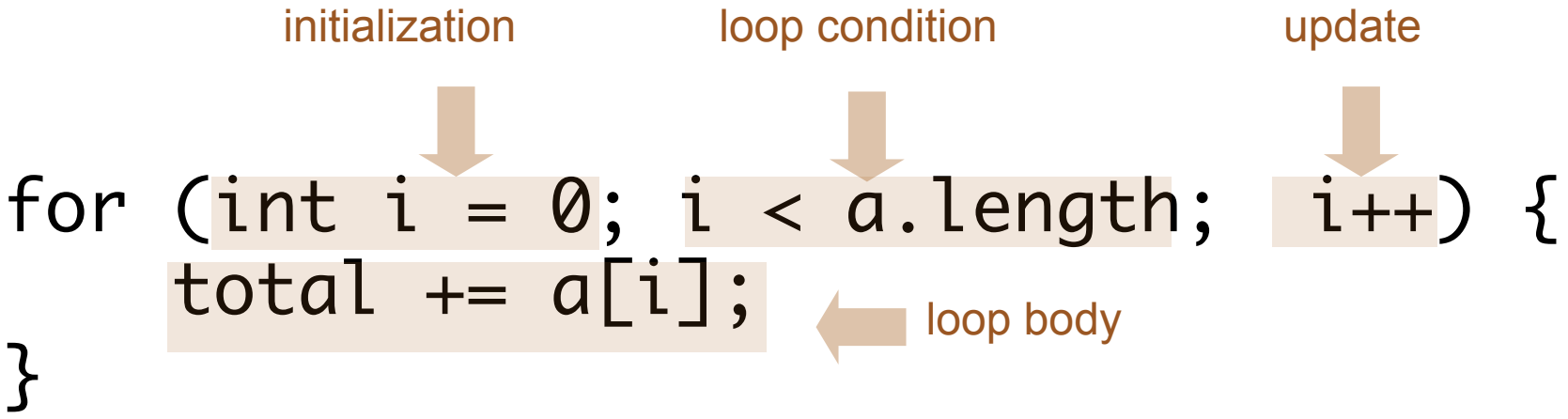
```
Counter[] a = { new Counter(), new Counter() };
Counter[] b = { a[0], a[1] };
a[0].inc();
b[0].inc();
int ans = a[0].inc();
```

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;
  }

}
```

Array Iteration

For loops




```
static double sum(double[] a) {
  double total = 0;
  for (int i = 0; i < a.length; i++) {
    total += a[i];
  }
  return total;
}
```

General pattern for computing info about an array

Multi-Dimensional Arrays

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

```
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]`

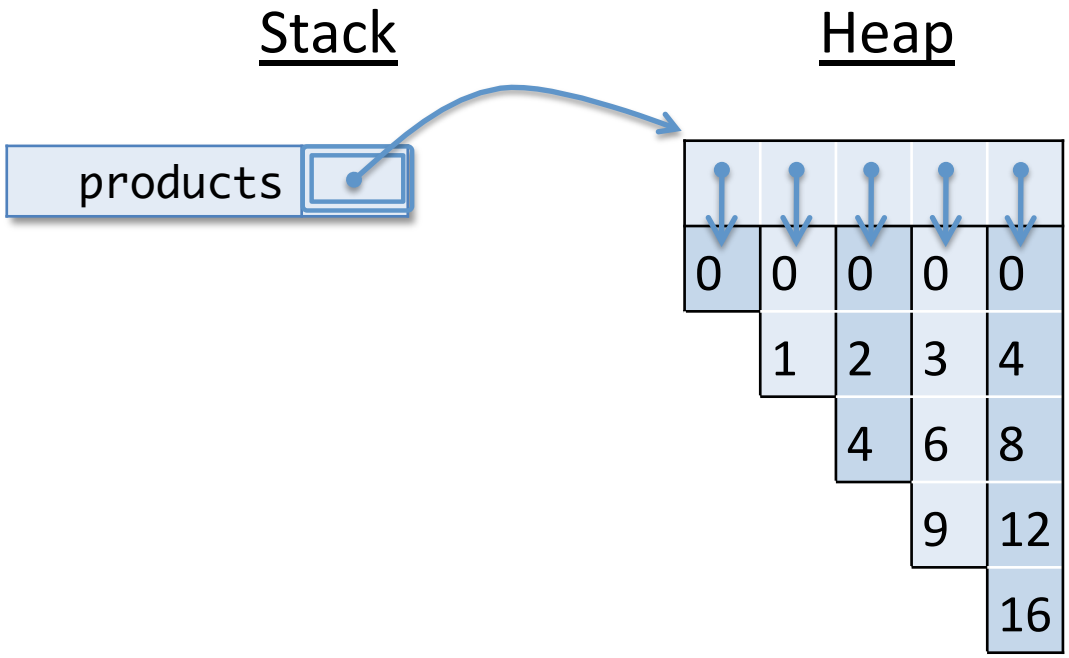
More brackets → more dimensions

Multi-Dimensional Arrays

```
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;
   }
}
```

Multi-Dimensional Arrays

```
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.

Demo

ArrayDemo.java