Programming Languages and Techniques (CIS1200)

Lecture 23

Static Methods, Java Arrays

Chapters 20, 21

Announcements

- HW06: Pennstagram
  - Java array programming
  - Available on course website
  - Due *Tuesday, March 25th*

Midterm 2 Logistics

- **Friday, March 28th, 2025**
  - ***During lecture:*** **1:45-2:45PM**
- **Location**: Meyerson B1 (MEYH)
- **Coverage:** Chapters 1-24
- **Format:** 60 minutes; one handwritten, letter sized, single sided sheet of notes allowed.
- **Review Session:**
  Wednesday, March 26 from 7-9pm in Towne 100

Static Methods

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

Internally (within the same class), 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; it is invoked to start the program running

Externally, prefix with name of the class

Static Fields

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

```
public static final double PI = 3.141592653589793238462643383279;
```

Style: naming conventions

Kind	Part-of-speech	Example
interface	adjective	`Runnable`
class	noun	`RacingCar`
field / variable	noun	`initialSpeed`
static final field (constants)	noun	`MILES_PER_GALLON`
method	verb	`shiftGear`

- 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

Why naming conventions matter

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

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

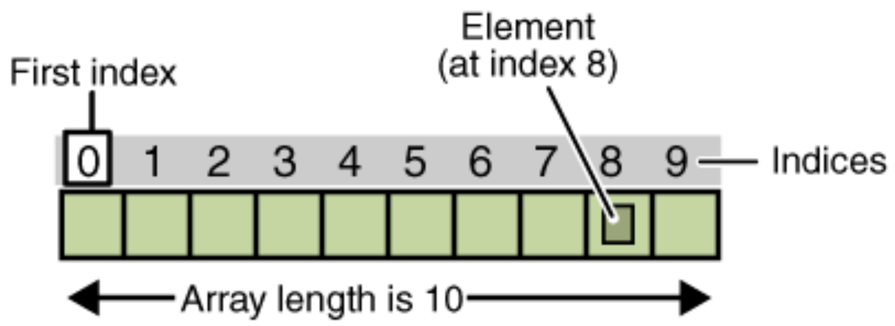
Many more details on good Java style here:
http://www.seas.upenn.edu/~cis1200/current/java_style

Java Arrays

Working with static methods

Java Arrays: Indexing

- Arrays are sequentially ordered collections of values that can be indexed directly (it takes the same time to access any position in the array)
- The first index is 0



Index must be in range:
a[20] or a[-1] triggers
ArrayIndexOutOfBoundsException

Array must be defined:
If a is null, then a[i] triggers
NullPointerException

- 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: Creation

- Create an array a of size n with elements of type C, initialized with default values (null for references, 0 for int, etc.)

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

- Create an array with given initial values

```
C[] a = new C[] { new C(1), new C(2) };
```

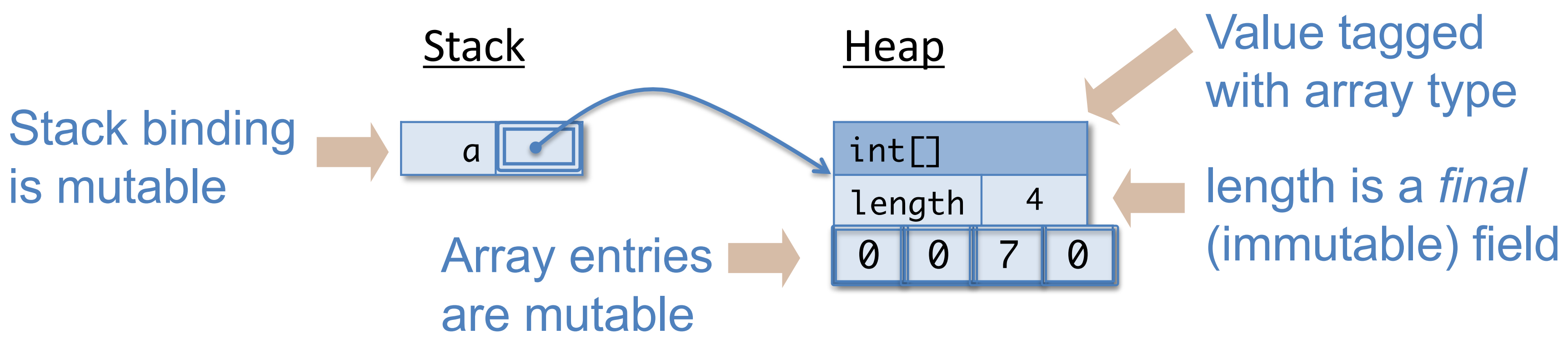
- When initializing a variable can omit `new` keyword and type

```
C[] a = { new C(1), new C(2) };
```

Arrays and the Java ASM

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

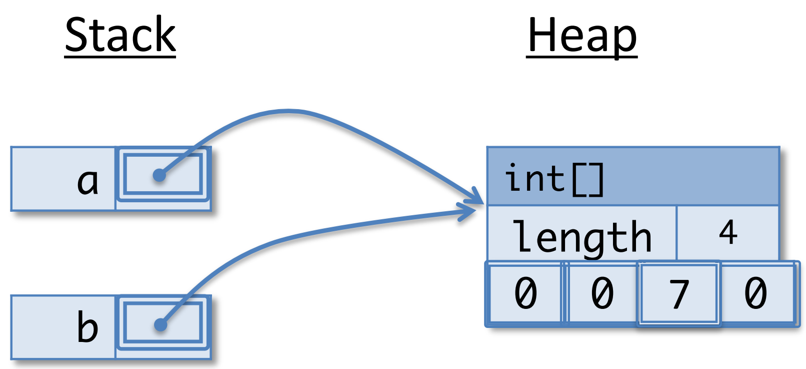
```
int[] a = new int[4];
a[2] = 7;
```



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];
```



23: What is the value of *ans* at the end of this program?

0

1 — 0%

2 — 0%

3 — 0%

4 — 0%

NullPointerException — 0%

ArrayIndexOutOfBoundsException — 0%

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

23: What is the value of *ans* at the end of this program?

✓ 0

1 0%

2 0%

3 0%

4 0%

NullPointerException 0%

ArrayIndexOutOfBoundsException 0%

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

Answer: NullPointerException

23: What is the value of *ans* at the end of this program?

0

1	0%
2	0%
3	0%
0	0%
NullPointerException	0%
ArrayIndexOutOfBoundsException	0%

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

23: What is the value of *ans* at the end of this program?

0

1	0%
2	0%
3	0%
0	0%
NullPointerException	0%
ArrayIndexOutOfBoundsException	0%

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

Answer: 0

Array Iteration

For loops

initialization loop condition update

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

loop body

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

General pattern for computing info about an array

For-each loops

element declaration

array

```
for (int x : a) {
    total += x;
}
```

loop body

Note that this is "just" iteration – no access to the array index!

```
static int sum(int[] a) {
  int total = 0;
  for (int x : a) {
    total += x;
  }
  return total;
}
```

Access all array elements in sequence

Array Copy and Equality

- Use `System.arraycopy` to copy arrays
- Use `Arrays.equals` to compare arrays structurally

Copy data from array a to array c, starting at position 0 in a and at position 0 in c. Copy a.length elements.

```
int[] a = { 1, 2, 3 };
int[] b = a;
int[] c = new int[a.length];
System.arraycopy(a,0,c,0,a.length);

System.out.println(a == b);                   // true
System.out.println(a == c);                   // false
System.out.println(a.equals(b));              // true
System.out.println(a.equals(c));              // false
System.out.println(Arrays.equals(a,b));       // true
System.out.println(Arrays.equals(a,c));       // true
```

Multidimensional Arrays

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

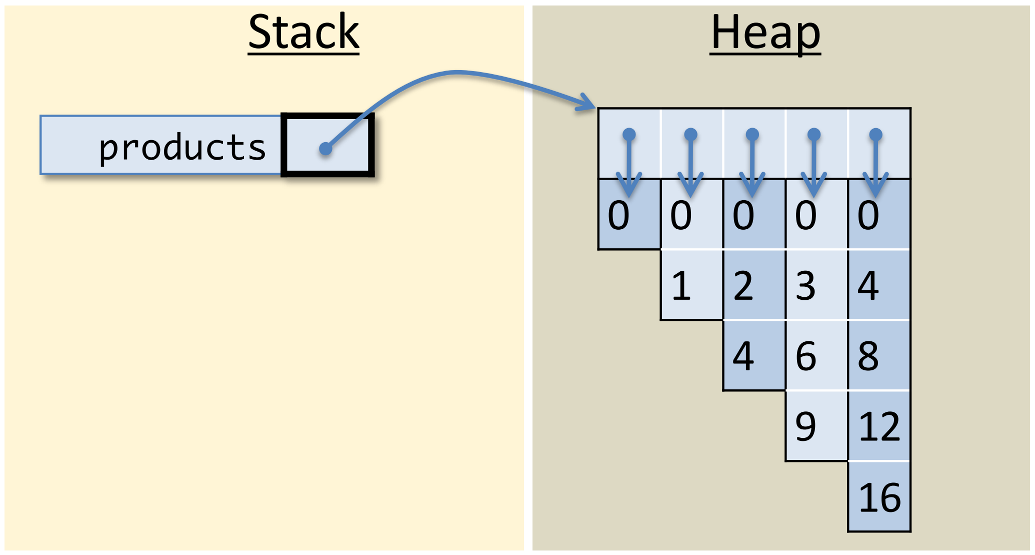
Multi-Dimensional Arrays

```
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;
    }
}
```

What would a Java ASM stack and heap look like after running this program?

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 on the heap.

Demo

ArrayDemo.java

ArrayExamples.java

Design Exercise: Resizable Arrays

Arrays that grow without bound.

Please see Chapter 33 in the Lecture Notes for more practice with arrays

Object encapsulation

- ***All modification to the state of the object must be done using the object's own methods.***

- Use encapsulation to preserve invariants about the state of the object.

- Enforce encapsulation by not returning aliases from methods.