Advanced Programming Techniques in Java

Class objectives

- Arrays (Chapter 7)

Review: Arrays an

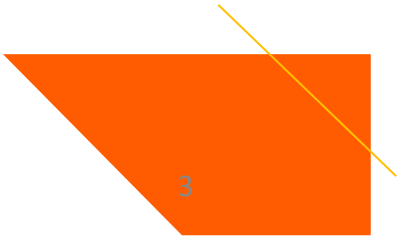
- **Method declaration**

```
public static type methodName
```

- Syntax:

```
public static type[] methodNa
```

- Syntax:


3

Review: Lim

- You cannot resize an existing array

```
int[] A = new int[4]; A.length
= 10;          // error
```

- An array does not know how to print itself

```
int[] A1 = {42, -7, 1, 15};
System.out.println(A1);
```

- You cannot compare arrays with `==` or `.equa`

```
int[] A1 = {42, -7, 1, 15};
int[] A2 = {42, -7, 1, 15};
if (A1 == A2) {  ... }          //
false! if (A1.equals(A2)) {  ... }
// false!
```

Review: Limitation

```
public static void main(String[] args)
        int[] A = {126, 167,
        95}; int[] B = A; int[]
        C = {126, 167, 95};

        System.out.println("A location
        System.out.println("B location
        System.out.println("C location

        System.out.println(Arrays.toStr
        System.out.println(Arrays.toStr
        System.out.println(Arrays.toStr
}
```

Array question

- Write a method **increase** that accepts one arra
  array with all the element values increased by

Array question

- Write a method **increase** that accepts one arra
  array with all the element values increased by

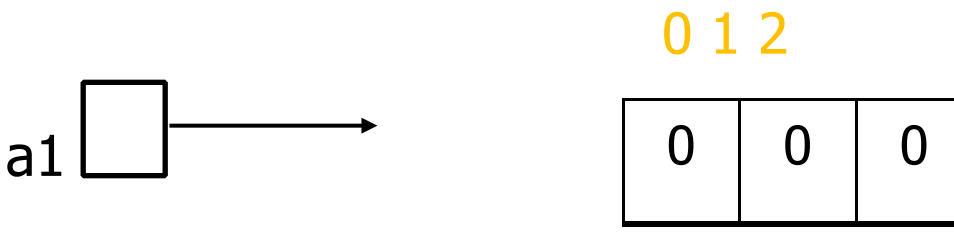
```
public static void increase(int[] a)
      for (int i = 0; i < a.length; i
      { a[i] = a[i] + 2;
      }
}
```

Does this look good?

Arrays and referen

- An array is a type of object
- An array variable is a reference variable
  - It stores a reference to the array

<u>Example</u> `int[] a1 = new int[3];`



Reference and obj

- Arrays and objects use reference semantics

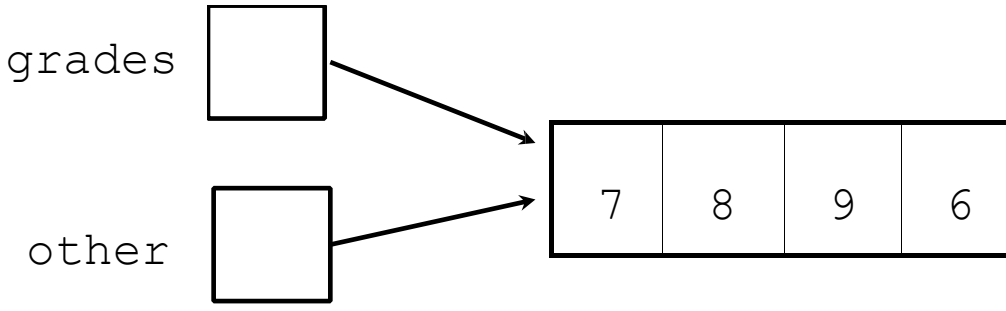
Why?

- Efficiency: copying large objects slows down a
- Sharing: it's useful to share an object's data ar

Copying reference

An example involving an array

```
int[] grades = {7, 8, 9, 6, 10, 7,
int[] other = grades;
```



Arrays passed by r

- Arrays are passed as parameters by reference
  - Changes made in the method are also seen by the

```
public static void main(String
        int[] iq = {126, 167, 95
        increase(iq);
        System.out.println(Array
}

public static void increase(in
        (int i = 0; i < a.length
        a[i] = a[i] * 2;
        }
}
```


Arrays passed by r

- Arrays are passed as parameters by reference
  - Changes made in the method are also seen by the

```
public static void main(String[] args) {
        int[] iq = {126, 167, 95};
        increase(iq);
        System.out.println(Arrays.toString(iq
}

public static void increase(int[] a) { for
        (int i = 0; i < a.length; i++) {
        a[i] = a[i] * 2;
        }
}
```

Reference semant

- Reference semantics (or reference types): beh
address of an object in memory

`Null` reference

- To indicate that a reference variable does
  assign it a special value called `null`

```
int[] grades = null;
String s = null;
```

gr

s

- Attempting to use a `null` reference to ac
  `NullPointerException`
  - Pointer is another name for reference

```
grades[3] = 10; //NullPointerExcepti
ch = s.charAt(5); //NullPointerExcep
```

String vs. Array Ob

```
public class Test{ public static void

   main(String[] args){

      int[] A = {1, 2, 3};
      int[] B = {1, 2, 3};
      int[] C = {1, 2, 3};

      if(A == B){
         System.out.println("true");
      }else{
         System.out.println("false");
      }
      if(A == C){
         System.out.println("true");
      }else{
        System.out.println("false");
      }
   }
}
```

```
publ

   r

}
```

String Objects

- There are two ways to create string objects in
  - `String s1 = "ABC";` //string constant pool
  - `String s3 = new String("ABC");` //heap

- The string constant pool is a separate place in the heap memory where the values of all the strings which are defined in the program are stored

- Duplicates are not allowed in the string constant pool

```
String s2 = "ABC";
```

String immutabilit

```
public class Test{

        public static void main(String[] args){
                String str = "ABC";
                str.concat("DEF");
                System.out.println(str);
        }
}
```

Java Constants

```
public static final int ARRAY_SIZE =

private static final String URL = "ts
```

- Constants in Java have to be initialized wher
- After that, they are read only.

Using an array

- Write a method `mostFrequentDigit` th
occurs most frequently in a number

- <u>Example</u>
  - The number 669260267 contains one 0, tw
  `mostFrequentDigit(669260267)` return
  - If there is a tie, return the digit with the lower val
  returns 3

Using an array

- We could declare 10 counter variables … `int c`

  `counter3, counter4, …, counter9;`

- But a better solution is to use an array of siz

  element at index `i` will store the counter for dig

  Example for 669260267

0	1	2	3	4	5	6
1	0	2	0	0	0	4

- How do we build such an array? And how do

Creating an array c

```
// assume n = 669260267
```

```
int[] counts = new int[10];
int digit = 0;
while (n > 0) {
    // pluck off a digit and add to p
    digit = n % 10;
    counts[digit]++;
    n = n / 10;
}
```

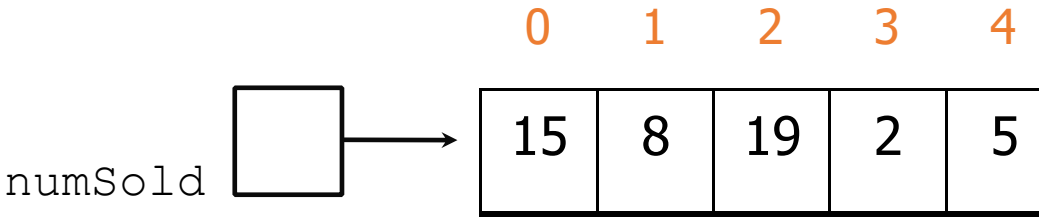
0	1	2	3	4	5	6
1	0	2	0	0	0	4

Creating an array o

```
// Returns the digit value that occurs mo
// Breaks ties by choosing the smaller va
static int mostFrequentDigit(int n) { int
int[10]; int digit = 0; while (n > 0) { d
// pluck off a digit and tally it counts[
/ 10;
    }
    // find the most frequently occurring
    int bestIndex = 0;
    for (int i = 1; i < counts.length; i+
        if (counts[i] > counts[bestIndex]
        bestIndex = i;
        }
    }
    return bestIndex;
}
```

Shifting values in a

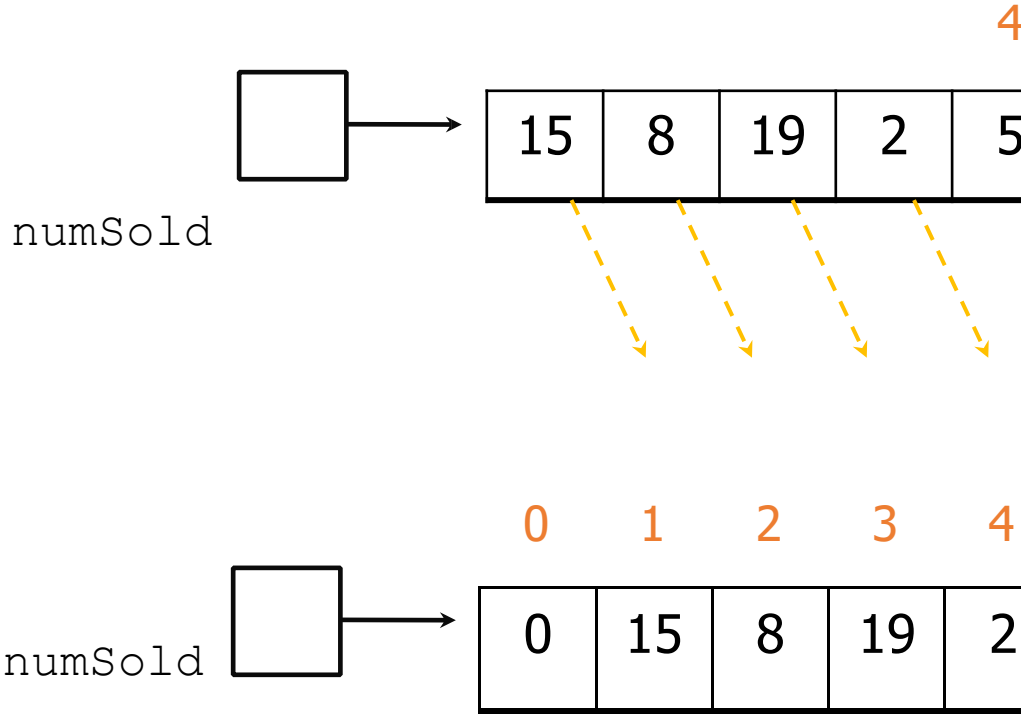
- A small business is using an array to store the n
  period



`numSold[0]` gives the number of
`numSold[1]` gives the number of
`numSold[2]` gives the number of
…
`numSold[9]` gives the number of

Shifting values in a

- At the start of each day, it's necessary to shif
  the new day's sales



- The last value is lost, since it’s now 10

Shifting values in

```
for (int i = 0; i < numSold.length; i
        numSold[i] = numSold[i - 1]
```

- Does this work?

```
for (int i = 0; i < numSold.length; i
        numSold[i] = numSold[i -
}
```

Shifting values in

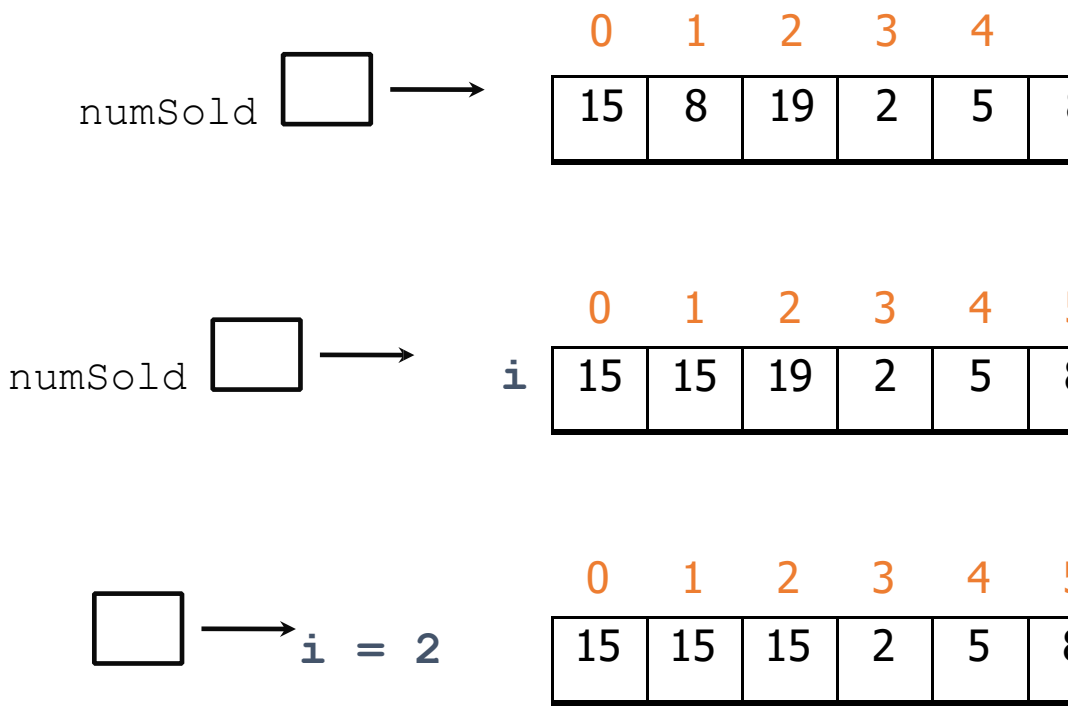
- No, If we run this, we get an `ArrayIndexOu`

```
    for (int i = 1; i < numSold.length; i
            numSold[i] = numSold[i - 1]
```

- Does this work?

```
for (int i = 1; i < numSold.length; i
            numSold[i] = numSold[i -
}
```

Shifting values in



- It doesn't work!

Shifting values in

```
for (int i = 1; i < numSold.length; i
numSold[i] = numSold[i - 1];
}
```

- How can we fix the code below so that it does

```
for (int i = numSold.length - 1; i >=
{ numSold[i] = numSold[i - 1]; }
```

- Are we done?

```
for (int i = 1; i < numSold.length; i
```

Shifting values in

```
        numSold[i] = numSold[i - 1];
}
```

- How can we fix the code below so that it does

```
for (int i = numSold.length - 1; i >=
{ numSold[i] = numSold[i - 1]; }
```

- After performing all the shifts, we would do:

“Growing” an arra

- Once we have created an array, we can’t incr

size

- Instead, we need to do the following:
  - Create a new, larger array
  - Copy the contents of t

    original array into the new array
  - Assign the new arra

    the original array variable

```
int[] a1 = {42, -7, 1, 15};
…
int[] tmp = new int[10];
for (int i = 0; i < a1.length; i+
       tmp[i] = a1[i];
} a1 =
tmp;
```