Advanced Programming Techniques in Java

Class obj

- Wrapper Classes (Last subsection of 10.1)
- More on Arrays (Chapter 7)
- Object Oriented Design (Section 8.1)

Review: L
# of arrays

- 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 ==

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

Review: String Obj

- 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

s

- Duplicates are not allowed in the string constant pool

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

Wrapper Classes for Primit

- Primitive numeric types are not objects, but so processed like objects
  - When?
- Java provides *wrapper classes* whose objects
  - `Float, Double, Integer, Boolean,`
  - They provide constructor methods to create nev

- Also provide methods to “unwrap”

Wrapper classes

Primitive Type	Wrapp
`int`	`Intege`
`double`	`Doubl`
`char`	`Chara`
`float`	`Float`

boolean	Boole

- A wrapper is an object whose sole purpose is to hold

Boxing/Unboxing

- Java automatically converts between the two using unboxing
- **Boxing**: automatic conversion from primitive data t type
- **Unboxing**: automatic conversion from a wrapper o

Examples

```
Integer i1=35;

Integer i2=1234; Integer i3=i1+i2

int i2Val=i2++; int

i3Val=Integer.parseInt(“-357”);

Integer i4= Integer.valueOf(753);

System.out.println(i1);
```

More Examples

- `System.out.println(i1+i2);`
- `System.out.println(i1.toStrin`
- Operations are the same as primitive ty

  autoboxes before the operator is applied.
- What happens for the == operator?

Review: S

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

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

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

- After performing all the shifts, we would do:

“Growing

- Once we have created an array, we can’t incr
- Instead, we need to do the following:

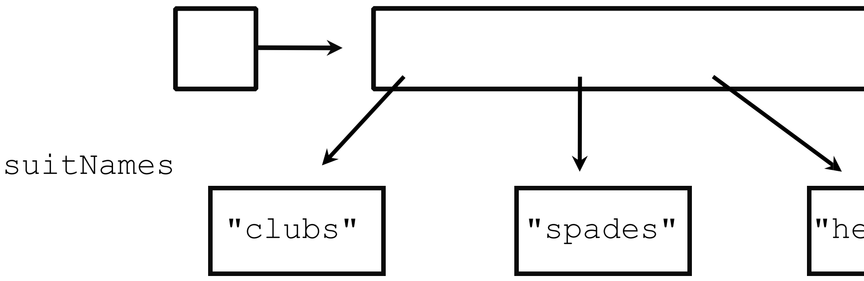
○ Create a new, larger array ○ Copy the contents of t
original array into the new array ○ Assign the new arra
the original array variable

```
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;
```

Array of o

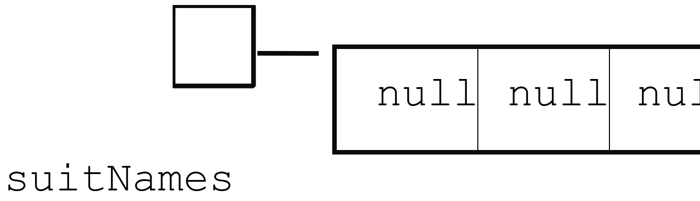
- All the arrays we have looked so far have stor
- But … you can have arrays of any Java type
- We can use an array to represent a collection

```
String[] suitNames = {"clubs", "spades'
```



Array of o

```
String[] suitNames = new String[4];
```



- Because we didn’t use an initialization list in contain all `null` values

Multidime

- You can form arrays of arbitrarily many dimen
- The most common type is a two-dimensional (
- We can visualize it as a matrix consisting of ro

```
int[]             //one-dimensional array 
int[][]           //two-dimensional array 
int[][][]         //three-dimensional arra
```

Row indices

	0	1	2	3	4	5
0	15	8	3	16	12	7
	6	11	9	4	1	5
3	17	3	5	18	10	6
4	8	14	13	6	13	12
	1	9	5	16	20	2

2D Arrays

- **Declaring** and **creating** a 2D array:

```
<type> [][]arrayName = new <type> [<r
```

```
int[][] score = new int[5][8];
```

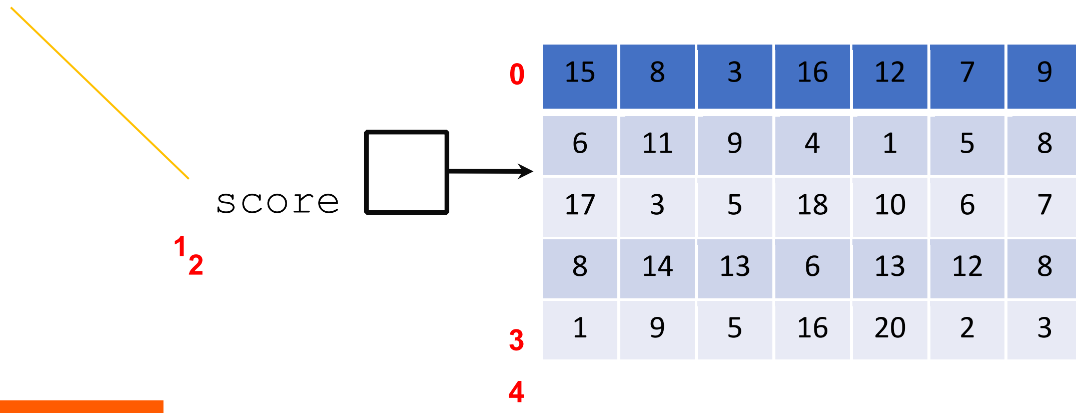
Number of rows

Nu

- To **access** an element: `arrayName[<row>][<c`

`score[3][4]` will give you the value at row 3, colun

0 1 2 3 4 5 6



15	8	3	16	12	7	9
6	11	9	4	1	5	8
17	3	5	18	10	6	7
8	14	13	6	13	12	8
1	9	5	16	20	2	3

2D Arrays

score$^{1}_{2}$

- `score[0]` represents the entire first row
- `score[1]` represents the entire second row,

- A 2D array is really an array of arrays score

- score.length gives the number of rov
- score[0].length gives the number of col

Printing a 2D Arrays

```
public static void main (String[] args)
arr = {{1, 2, 3}, {3, 4, 5}, {2, 2, 2}}
print(arr); }
```

```
public static void print(int[][] arr) {
          r < arr.length; r++) { for (int c
          arr[r].length; c++) { System.out.p
          " ");
                     }
                     System.out.println();
          }
}
```

Sorting an array

- Sorting is a common programming task in sorting:
  - List containing exam scores sorted from lowest
  - List of student records and sorted by student n name

Why do we

- Searching for an element in an array will b
  information like phone numbers)
- It’s always nice to see data in sorted displa

Bubble Sort

- Oldest and simplest sorting algorithm
- Relatively slow algorithm
- Idea:
  - Large values “bubble” to the end of the list whil
    beginning of the list
- Algorithm in words:
  - Compare every pair of adjacent items, swappin
  - Repeat this process until a pass is made all the
    swapping any items (i.e. the items are in the co

Bubble Sort

Bubble Sort

```
public static void bubbleSort(int[]
    arr) int didswap = 1, tmp = 0 while
    (didswap == 1) { didswap = 0;
           for (int i = 1; i < arr.length; i++)
                      if (arr[i - 1] > arr[i]) {
                                 tmp = arr[i - 1];
                                 arr[i - 1] =
                                 arr[i]; arr[i] =
                                 tmp; didswap = 1;
                      }
           }
    }
}
```

Bubble Sort

```
public static void bubbleSort(int[]
   arr) int didswap = 1, tmp = 0 while
   (didswap == 1) { didswap = 0;
         for (int i = 1; i < arr.length; i++) {
                  if (arr[i - 1] > arr[i]) {
                           tmp = arr[i - 1];
                           arr[i - 1] =
                           arr[i]; arr[i] =
                           tmp; didswap = 1;
                  }
         }
   }
}
```

Bubble Sort

```
public static void bubbleSort(int[]
   arr) int didswap = 1, tmp = 0 while
   (didswap == 1) { didswap = 0;
         for (int i = 1; i < arr.length; i++) {
                  if (arr[i - 1] > arr[i]) {
                           tmp = arr[i - 1];
                           arr[i - 1] =
                           arr[i]; arr[i] =
                           tmp; didswap = 1;
                  }
         }
   }
}
```

Are we done

```
public static void bubbleSort(int[]
   arr) int didswap = 1, tmp = 0 while
   (didswap == 1) { didswap = 0;
        for (int i = 1; i < arr.length; i++) {
                if (arr[i - 1] > arr[i]) {
                        tmp = arr[i - 1];
                        arr[i - 1] =
                        arr[i]; arr[i] =
                        tmp; didswap = 1;
                }
        }
   }
}
```

Bubble Sort

Classes and Ob

Object-Oriented Programmin

- **Procedural programming**
- Oldest style of programming
- **Object-oriented programming**

Procedural vs. OO Programming

- Command-line interface
- e.g., to delete a file you type `rm data.txt`
- “verb noun”
- GUI interface
- e.g., to delete a file you locate the icon for the file an options, including the delete option
- “noun verb”

- Object-oriented programming (OOP) :
- Reasoning about a program as a set of objects rathe

So far …

- We have seen:
- **variables**, which represent data (categorized by **ty**
- **methods**, which represent behavior

- It is possible to create new types that are com
- Such types are called **object types** or **referen**

What is an Object?

- An object groups together:
- One or more data values (the object's fields)
- A set of operations that the object can perform (the
- An **object** is a programming entity that has **state** (d

State and Behavior

- **Definition**

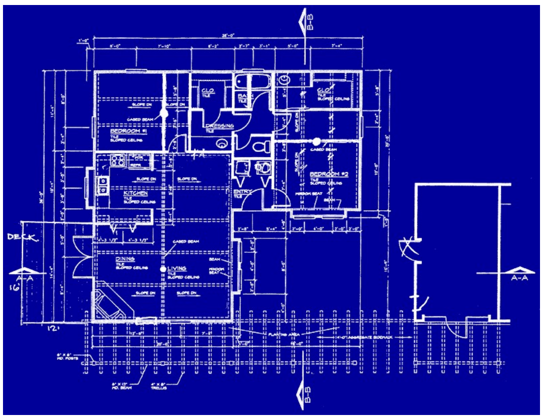
- A **state** is a set of values (internal data) stored in ar

- **Definition**

- A **behavior** is a set of actions an object can perforn state

What is a Class?

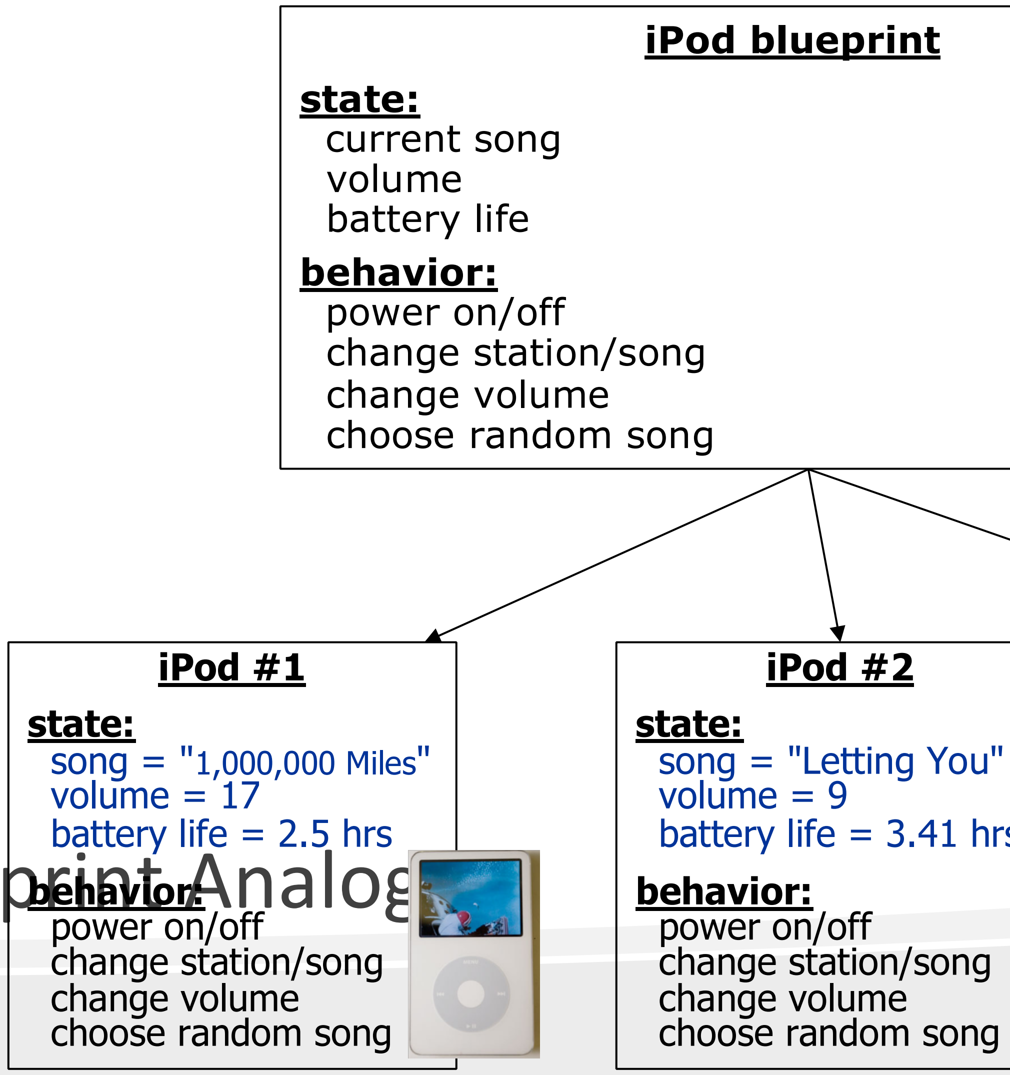
- It is a definition of a new type of objects



- The objects of a given class are built according to it
- Objects of a class are referred to as instance of the

- **Definition**
- A **class** is like a blueprint (defined by the user) for w

Blueprint Analog



iPod blueprint

state:
current song
volume
battery life

behavior:
power on/off
change station/song
change volume
choose random song

iPod #1

state:
song = "1,000,000 Miles"
volume = 17
battery life = 2.5 hrs

behavior:
power on/off
change station/song
change volume
choose random song

iPod #2

state:
song = "Letting You"
volume = 9
battery life = 3.41 hrs

behavior:
power on/off
change station/song
change volume
choose random song

Classes and Objects

- To create a new type of object in Java we must specify:
- The state stored in each object
- The behavior each object can perform
- How to construct objects of that type

Creating your own Classes

- Let’s implement a `Point` class
- We will define a type of object named `Point`
- Each `Point` object will contain **fields** (states)
- Each `Point` object will contain **methods** (behaviors

Point objects

- Java has a class of objects named `Point`
- To use `Point`, you must write: `import java.awt.*;`

- Constructing a `Point` object, general syntax:

```
Point <name> = new Point(<x>, <y>);
Point <name> = new Point();   // the orig
```

- Example:

```
Point p1 = new Point(5, -2);
Point p2 = new Point();
```

- Available methods: `translate(dx,dy), setLoc`

  `PointExample1.java`

- Available public fields: `x, y`

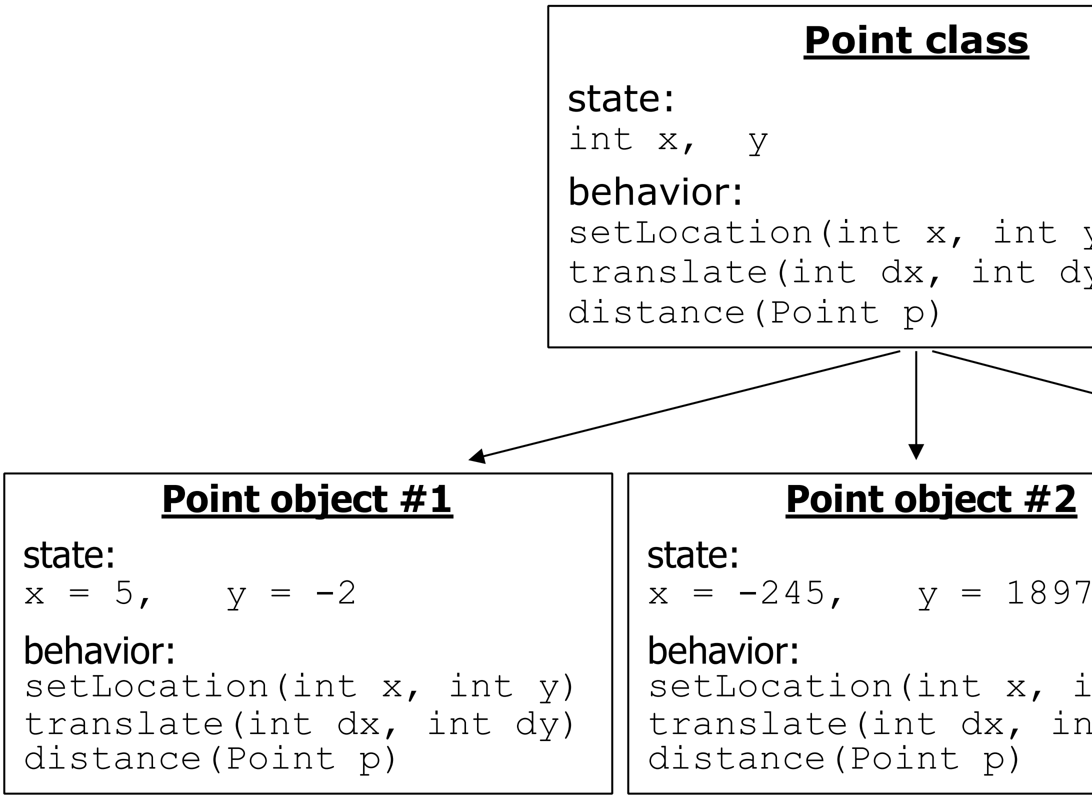
```
import java.awt.*; public

class PointExample1{

   public static void main(String[] args){
         Point p = new Point(3, 8);
         System.out.println("initially p =
         p.translate(-1, -2);
         System.out.println("after translat
   }

}
```

`Point` Class as Blueprint



Point class

state:

```
int x, y
```

behavior:

```
setLocation(int x, int y
translate(int dx, int dy
distance(Point p)
```

Point object #1

state:

```
x = 5,    y = -2
```

behavior:

```
setLocation(int x, int y)
translate(int dx, int dy)
distance(Point p)
```

Point object #2

state:

```
x = -245,    y = 1897
```

behavior:

```
setLocation(int x, i
translate(int dx, in
distance(Point p)
```

Object State: Fields

- **Definition**
- A field is a variable inside an object that makes up part o

- **Syntax**

```
 <type> <name>;
```

- **Example** `public class Student{` Each `Student`

```
      double gpa;
 }
```

gpa field

String name;

Point Class (ver. 1)

```
public class
Point{ int x; int
y; }
```

- This code creates a new type named Point
- Point objects do not contain any behavior **yet**

- Each Point object contains two fields: an `int` name
- Each object has its own copy of each field
- If we create 100 Point objects, we'll have 100 pairs of `x` a

Different type of varia

- Static variables
- Instance variables
- Local variables
- Constants

Constructing objects

- **Construct**: To create a new object
- Objects are constructed with the **new** keyword
- Most objects must be constructed before they can be us
- **Syntax**

```
  <type> <name> = new <type> ( <parame
```

- Strings are also objects, but can be constructed wi

```
String name = "Amanda Ann Camp";
```

- **Example**:

```
Point p = new Point();
```

Access/Modify Fields

- **Syntax**
- Access: **object.field**
- Modify: **object.field = value;**

- **Example**

```
Point p1 = new Point();
```

- The keyword `new` creates (constructs) `Point` object
- When a `Point` object is constructed, its fields are gi

```
Point p2 = new Point();
System.out.println("the x-coord is " + p1
p2.y = 13;
```

Point Class (ver. 1)

```
public class
Point{ int x; int
y; }
```

- The `Point` class isn't itself an executable program

  themselves are not complete programs
- They can only be used as part of larger programs to solve
- The program that creates and uses objects is known