Advanced Programming Techniques in Java

COSI 12B

Class objectives

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

Review: Limitations 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 `==` or `.equals` for Strings)

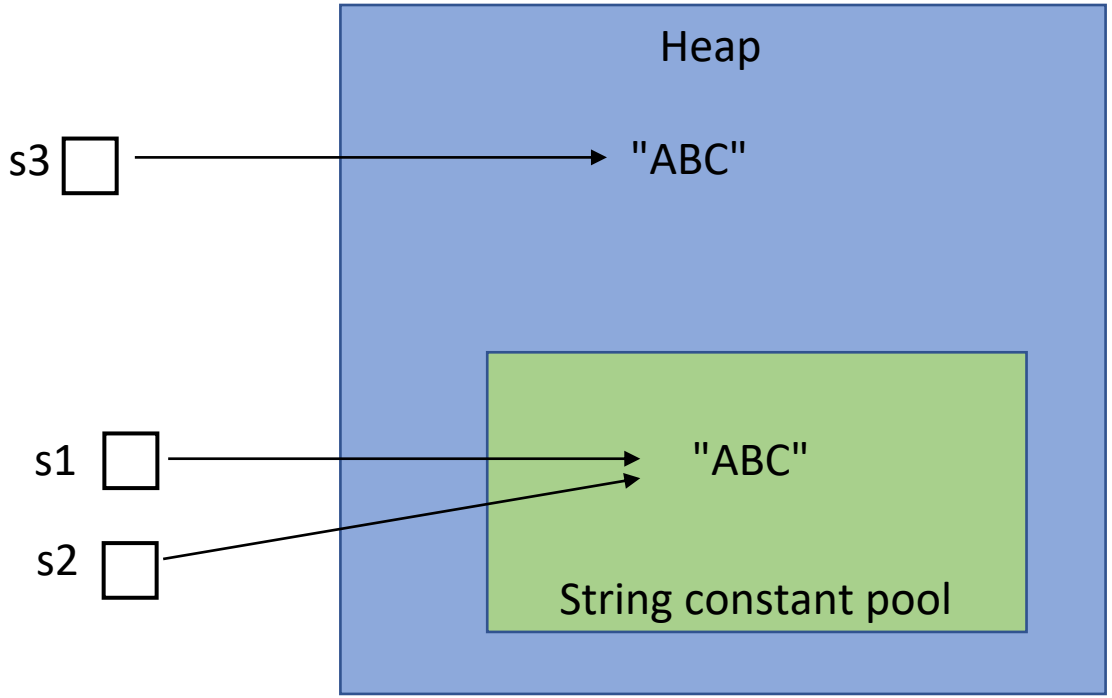
```
int[] A1 = {42, -7, 1, 15};
int[] A2 = {42, -7, 1, 15};
if (A1 == A2) {  ... }          // false!
if (A1.equals(A2)) {  ... }     // false!
```

Review: String Objects

- There are two ways to create string objects in Java
  - `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";
```



Wrapper Classes for Primitive Types

- Primitive numeric types are not objects, but sometimes they need to be processed like objects
  - When?
- Java provides *wrapper classes* whose objects contain primitive-type values
  - `Float, Double, Integer, Boolean, Character`
  - They provide constructor methods to create new objects that “wrap” a specified value
  - Also provide methods to “unwrap”

Wrapper classes

Primitive Type	Wrapper Type
`int`	`Integer`
`double`	`Double`
`char`	`Character`
`float`	`Float`
`boolean`	`Boolean`

- A wrapper is an object whose sole purpose is to hold a primitive value

Boxing/Unboxing

- Java automatically converts between the two using techniques known as **boxing** and **unboxing**
- **Boxing**: automatic conversion from primitive data to a wrapper object of the appropriate type
- **Unboxing**: automatic conversion from a wrapper object to its corresponding primitive data

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.toString()+i2.toString());`
- Operations are the same as primitive types in current version of java
  - autoboxes before the operator is applied.
- What happens for the == operator?

Review: Shifting values in an array

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

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

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

- After performing all the shifts, we would do: `numSold[0] = 0;`

“Growing” an array

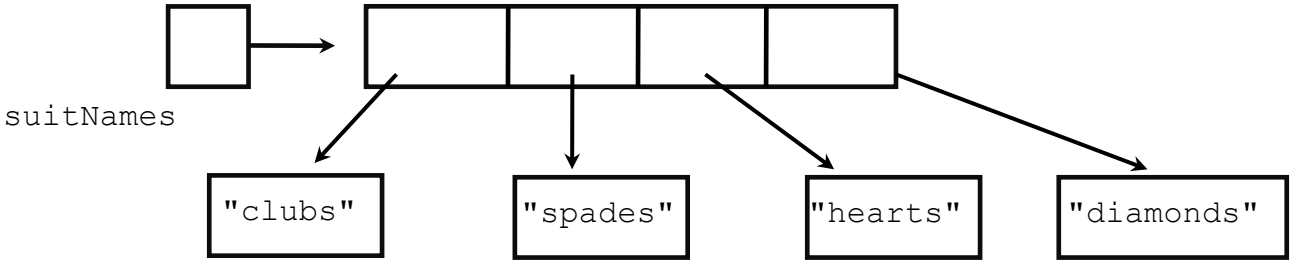
- Once we have created an array, we can’t increase its size
- Instead, we need to do the following:
  - Create a new, larger array
  - Copy the contents of the original array into the new array
  - Assign the new array to the original array variable

```
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 objects

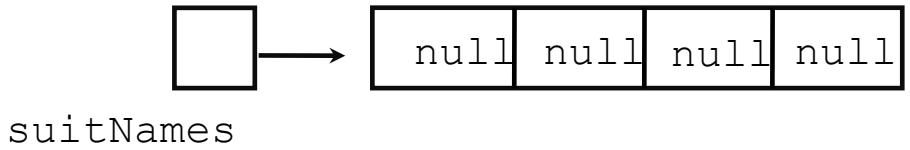
- All the arrays we have looked so far have stored primitive values
- But … you can have arrays of any Java type
- We can use an array to represent a collection of objects

```
String[] suitNames = {"clubs", "spades", "hearts", "diamonds"};
```



Array of objects

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



- Because we didn't use an initialization list in this case, the array will initially contain all `null` values

Multidimensional arrays

- You can form arrays of arbitrarily many dimensions (multi-dimensional arrays)
- The most common type is a two-dimensional (2D) array
- We can visualize it as a matrix consisting of rows and columns

```
int[]          //one-dimensional array of integers
int[][]        //two-dimensional array of integers
int[][][]      //three-dimensional array of integers
```

Column indices

	0	1	2	3	4	5	6	7
0	15	8	3	16	12	7	9	5
1	6	11	9	4	1	5	8	13
2	17	3	5	18	10	6	7	21
3	8	14	13	6	13	12	8	4
4	1	9	5	16	20	2	3	9

Row indices

2D Arrays

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

```
<type> [][]arrayName = new <type> [<row>][<column>]
```

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

Number of rows

Number of columns

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

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

score

	0	1	2	3	4	5	6	7
0	15	8	3	16	12	7	9	5
1	6	11	9	4	1	5	8	13
2	17	3	5	18	10	6	7	21
3	8	14	13	6	13	12	8	4
4	1	9	5	16	20	2	3	9

2D Arrays

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

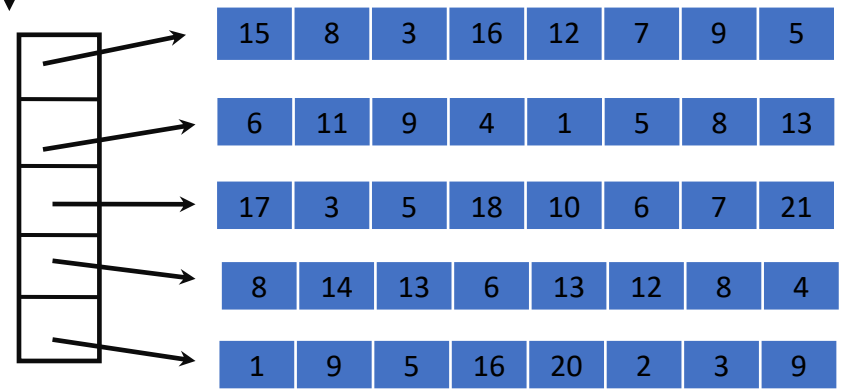
score

	0	1	2	3	4	5	6	7
0	15	8	3	16	12	7	9	5
1	6	11	9	4	1	5	8	13
2	17	3	5	18	10	6	7	21
3	8	14	13	6	13	12	8	4
4	1	9	5	16	20	2	3	9

- A 2D array is really an array of arrays

- `score.length` gives the number of rows
- `score[0].length` gives the number of columns

score



Printing a 2D Arrays

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

public static void print(int[][] arr) {
     for (int r = 0; r < arr.length; r++) {
                for (int c = 0; c < arr[r].length; c++) {
                     System.out.print(arr[r][c] + " ");
                }
                System.out.println();
     }
}
```

Sorting an array

- Sorting is a common programming task in computer science
- Example of sorting:
  - List containing exam scores sorted from lowest to highest or vice versa
  - List of student records and sorted by student number or alphabetically by the first or last name

Why do we sort?

- Searching for an element in an array will be more efficient (e.g., looking up for information like phone numbers)
- It’s always nice to see data in sorted display

Bubble Sort

- Oldest and simplest sorting algorithm
- Relatively slow algorithm

- Idea:
  - Large values “bubble” to the end of the list while smaller values “sink” towards the beginning of the list

- Algorithm in words:
  - Compare every pair of adjacent items, swapping if necessary
  - Repeat this process until a pass is made all the way through the array without swapping any items (i.e. the items are in the correct order)

Bubble Sort

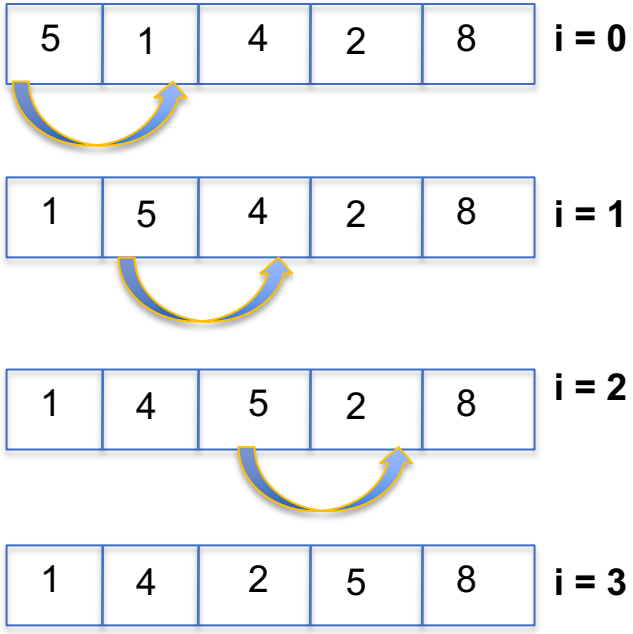
```
public static void bubbleSort(int[] arr)
   int didswap = 1, tmp = 0
   while (didswap == 1) {
         didswap = 0;
         for (int i = 1; i < arr.length; i++) {
                 if (arr[i - 1] > arr[i]) {
                         tmp = arr[i - 1];
                         arr[i - 1] = arr[i];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

Bubble Sort

5	1	4	2	8

```
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;
                  }
          }
    }
}
```

Pass 1



Bubble Sort

5	1	4	2	8

```
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;
                  }
          }
    }
}
```

Pass 2

1	4	2	5	8	**i = 0**
1	4	2	5	8	**i = 1**
1	2	4	5	8	**i = 2**
1	2	4	5	8	**i = 3**

Are we done?

Bubble Sort

5	1	4	2	8

```
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;
                  }
          }
    }
}
```

Pass 3

1	2	4	5	8	**i = 0**
1	2	4	5	8	**i = 1**
1	2	4	5	8	**i = 2**
1	2	4	5	8	**i = 3**

Classes and Objects

Object-Oriented Programming

- **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 and click on it. You have at this point several options, including the delete option
  - "noun verb"
- Object-oriented programming (OOP) :
  - Reasoning about a program as a set of objects rather than a set of actions

So far …

- We have seen:
  - **variables**, which represent data (categorized by **types**)
  - **methods**, which represent behavior
- It is possible to create new types that are combinations of the existing types
- Such types are called **object types** or **reference types**

What is an Object?

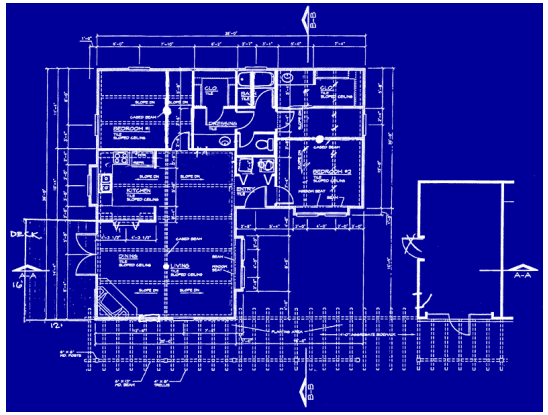
- An object groups together:
  - One or more data values (the object's fields)
  - A set of operations that the object can perform (the object's methods)
- **Definition**
  - An **object** is a programming entity that has **state** (data) and **behavior** (methods)

State and Behavior

- **Definition**
  - A **state** is a set of values (internal data) stored in an object
- **Definition**
  - A **behavior** is a set of actions an object can perform, often reporting or modifying its internal state

What is a Class?

- **Definition**
  - A **class** is like a blueprint (defined by the user) for which objects are created
    - It is a definition of a new type of objects



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

Blueprint Analogy

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

iPod #3

state:
song = "Discipline"
volume = 24
battery life = 1.8 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 create a class and add code to it that 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 origin, (0, 0)
```

- <u>Example</u>:

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

- Available methods: `translate(dx,dy)`, `setLocation(x,y)`, `distance(p2)`, ...
- Available public fields: `x`, `y`

PointExample1.java

```
import java.awt.*;

public class PointExample1{

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

}
```

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

Point object #3

state:
`x = 18, y = 42`

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

Object State: Fields

- **Definition**
  - A field is a variable inside an object that makes up part of its state

- **Syntax**

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

- **Example**

```
public class Student{
      String name;
      double gpa;
}
```

Each `Student` object has a `name` and `gpa` field

`Point` Class (ver. 1)

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

Save this code into a file named **`Point.java`**

- This code creates a new type named `Point`
- Each Point object contains two fields: an `int` named `x` and an `int` named `y`
- Each object has its own copy of each field
  - If we create 100 Point objects, we'll have 100 pairs of `x` and `y` fields, one for each object
- `Point` objects do not contain any behavior **yet**

Different type of variables

- 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 used

- **Syntax**

```
<type> <name> = new <type> ( <parameters> );
```

- **Example**:

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

- Strings are also objects, but can be constructed without new

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

Access/Modify Fields

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

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

- The keyword `new` creates (constructs) `Point` objects
- When a `Point` object is constructed, its fields are given default initial value of 0

Point Class (ver. 1)

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

Point.java

- The `Point` class isn't itself an executable program
- Objects themselves are not complete programs
  - They can only be used as part of larger programs to solve problems
- The program that creates and uses objects is known as **client code**