

# Using Objects from the Java API

Deborah A. Trytten  
CS 1324

# Classes and Objects: What We Know So Far

- ▶ Classes with a single static method: main
  - All the programs we wrote before Midterm 1
- ▶ Classes with a main method and some other static methods
  - Most of the programs we wrote after Midterm 1
- ▶ These are not typical classes
  - They can't be used to build objects
  - Most Java API classes can be used to build objects, although there are exceptions (Math, Arrays)

# A More Typical Class: String

- ▶ We've used lots of String objects
- ▶ Most were anonymous (not assigned to a variable):  
`System.out.println("I'm anonymous");`
- ▶ Some were not:  
`String name = new String ("Raven");`  
`String otherName = "Jazz"; // Hidden constructor`
- ▶ Reference variables: name and otherName
  - Variable holds the address of the object
  - Another example: `int[] array = new int[10];`

# Memory Allocation: Primitive Data vs. Objects

## Primitive Data

- ▶ Variable declaration:
  - `int length;`
  - Memory allocated in stack frame
- ▶ Value assigned:
  - `length = 3;`
  - No additional memory allocated

## String Objects

- ▶ Reference variable declaration:
  - `String name;`
  - Memory allocated in stack frame
- ▶ Object constructed and address assigned:
  - `name = new String("Daisy");`
  - Memory allocated on the heap
  - Address stored in stack frame

# Object Data

- ▶ Most objects conceal some data
  - Can be primitive data types or other objects
  - Data represents the properties of the object
  - Usually cannot access data directly
- ▶ Example: Each String object has its own sequence of characters
  - Perfect size `char[]`
- ▶ Sometimes we can guess the data and other times we can't
  - Example: `StringBuilder` (oversize `char[]`)
  - Example: `Scanner` (who knows?)

# Accessor Methods

- ▶ We can access the data in objects indirectly by using methods that are not labeled static
  - Example: `charAt()`, `length()`, `equals()`, `equalsIgnoreCase()`
- ▶ Syntax: `objectName.methodName(arguments)`
- ▶ The `objectName` tells Java which object's data to access
  - Object name becomes an *implicit* argument to the method

# Example Accessor: charAt

- ▶ Change a String to uppercase
  - Are we really changing the original String?
  - Write code to figure out
- ▶ Print out the letters in a String, one to a line

# iClicker Question

The String class has a method with the following signature:

```
String replace(char oldChar, char newChar)
```

This method creates a new String that replaces all occurrences of oldChar in a given String with newChar. If we have this declaration


```
String upper = new String ("Abcde");
```

How do we make upper refer to a String with all small letters?

- ▶ a) `replace('A', 'a');`
- ▶ b) `upper.replace('A', 'a');`
- ▶ c) `upper.replace("A", "a");`
- ▶ d) `upper = upper.replace('A', 'a');`
- ▶ e) `upper = upper.replace("A", "a");`




# Mutator Methods

- ▶ Some classes have methods that change the data in the object
  - ▶ String does not
    - Immutable objects
  - ▶ StringBuilder is similar to String, but is mutable
    - Stores an oversize array of characters
  - ▶ Use the API to find some mutators in the StringBuilder class
    - Notice unexpected return types
  - ▶ Reverse a String using a StringBuilder object
- 

# String vs. StringBuilder

- ▶ Java has two classes that store sequences of characters
  - String (immutable)
    - Perfect size char[]
  - StringBuilder (mutable)
    - Oversize char[]
- ▶ We were able to use String objects before we knew about arrays
  - Encapsulation

# Design Decisions

- ▶ By making String immutable, Java can be efficient in how it stores String objects
    - This matters because there are so many String objects
  - ▶ Example:
    - StringBuilder objects must have extra space for characters that can be added later
    - String objects do not need any extra space
  - ▶ Having multiple classes allows Java programmers to make good design decisions
- 

# Example: Design Decision

- ▶ Suppose we are storing a person's physical location (like GPS coordinates) as characters (not integers)
- ▶ Should we use a String or StringBuilder?

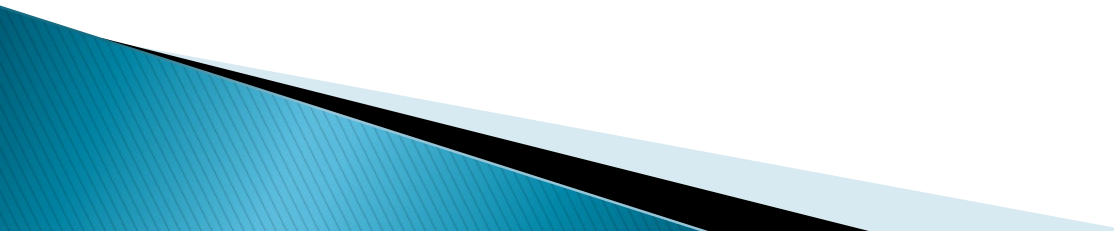
# iClicker Question

Suppose we are storing millions of people's names in a program that we expect will run for decades.

Which class should we use?

- a) String
- b) StringBuilder
- c) char[]
- d) char


# Wrapper Classes

- ▶ Java has a class related to each primitive data type
    - Boolean, Character, Double, Integer
  - ▶ Are objects in these classes are immutable
    - How can we tell when looking at the documentation?
  - ▶ Why do these classes exist?
    - We can do some cool programming with objects that can't be done with primitive data types
    - We'll see this next week!
- 

# Example: Double Class

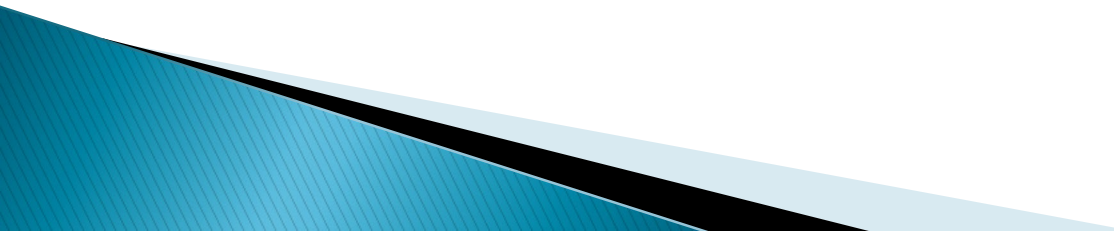
- ▶ Read a String containing a floating point number and create a double without using Scanner
  - Check if value is infinite or not a number
  - Convert it to a double if it is a legitimate, finite number

# What is a Class?

- ▶ Classes in the Java API are like bigger, more complex data types
  - ▶ Primitive data type example: `int`
    - Stores an integral number in binary
    - Has `+`, `-`, `*`, `/` provided by compiler
    - One data type, lots of integer values
  - ▶ Class as data type example: `String`
    - Stores a sequence of characters
    - Has methods provided by Java API
    - One `String` class, lots of `String` objects
- 



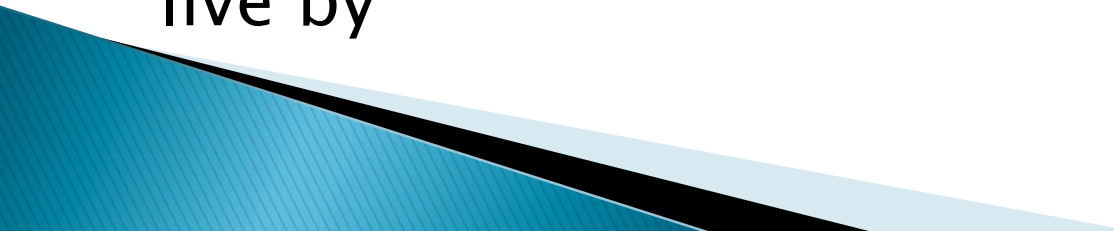
# What Does a Class Do?

- ▶ A class defines the type of data that is stored in objects of the class
  - ▶ A class defines methods that can access and change object data
- 

# Example: Point Class

- ▶ Used to store ordered pairs: (x, y)
- ▶ Data: two integers
  - Every Point object stores two int variables x and y
  - The API documentation tells us the name of the variables (fields), but this is unusual
- ▶ Methods: getX, getY, setLocation, etc.
  - Call these methods on any Point object to access or change its data

# What is an Object?

- ▶ Each object has a value for every data element in the class
  - ▶ Multiple objects from the same class will have the same data elements and type, but may have different data values
  - ▶ Classes are like cookie cutters
    - Objects are like cookies
    - Each cookie can have its own decorations
  - ▶ Classes establish the rules that the objects must live by
- 

# One Class / Multiple Objects

- ▶ Most classes allow many objects to be constructed
- ▶ Create three Point objects:

```
Point point1 = new Point(3, 5);  
Point point2 = new Point(2, 4);  
Point point3 = new Point(point1); // check API
```

  - Do they each have the same type(s) of data?
  - Do they each have the same data values?
- ▶ Draw memory diagram

# iClicker Question

How many objects are constructed in the following code fragment?

```
Point point1, point2, point3;  
point1 = new Point(1, 3);  
point2 = point1;  
point3 = new Point();
```

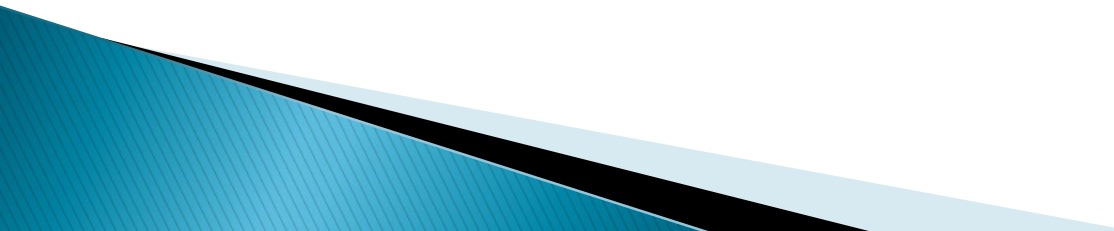
- a) 1      b) 2      c) 3      d) 4
- 

# Example: Accessors & Mutators

- ▶ Accessor methods allow programs to **obtain** data values stored in an object
- ▶ Mutator methods allow programs to **change** data values stored in an object
- ▶ Point example continued:  

```
point3.setLocation(9, 2);  
point2.translate(1, 5);  
point2.toString();
```
- ▶ Adjust memory diagram

# Example: Random Class

- ▶ Read API
  - ▶ Write a code fragment that prints 100 random numbers between 1 and 100 using this class.
  - ▶ We can use the class even though we have no idea how the numbers are created (encapsulation).
  - ▶ We don't even really know whether the methods are accessors or mutators. (They are both!)
- 

# Classes: Another Perspective

## ▶ Classes are a contract

- Each object from a class will have a legal state
  - Values for each data type defined by the class
- Mutators will result in another legal state
  - ALWAYS ALWAYS ALWAYS ALWAYS

## ▶ Example:

- State of a `StringBuilder` object is 0 or more characters in sequence (oversize array) with no gaps
- Methods enforce this
  - `deleteCharAt(int index)` doesn't leave gaps
  - `setLength(int newLength)`
    - What happens if an argument of `-1` passed to `newLength`?