THE UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

COMP 110 Introduction to Programming

Thursday November 13, 2014

Jay Aikat
Fall 2014
TR 9:30 – 10:45, GS-G100

Previous Class

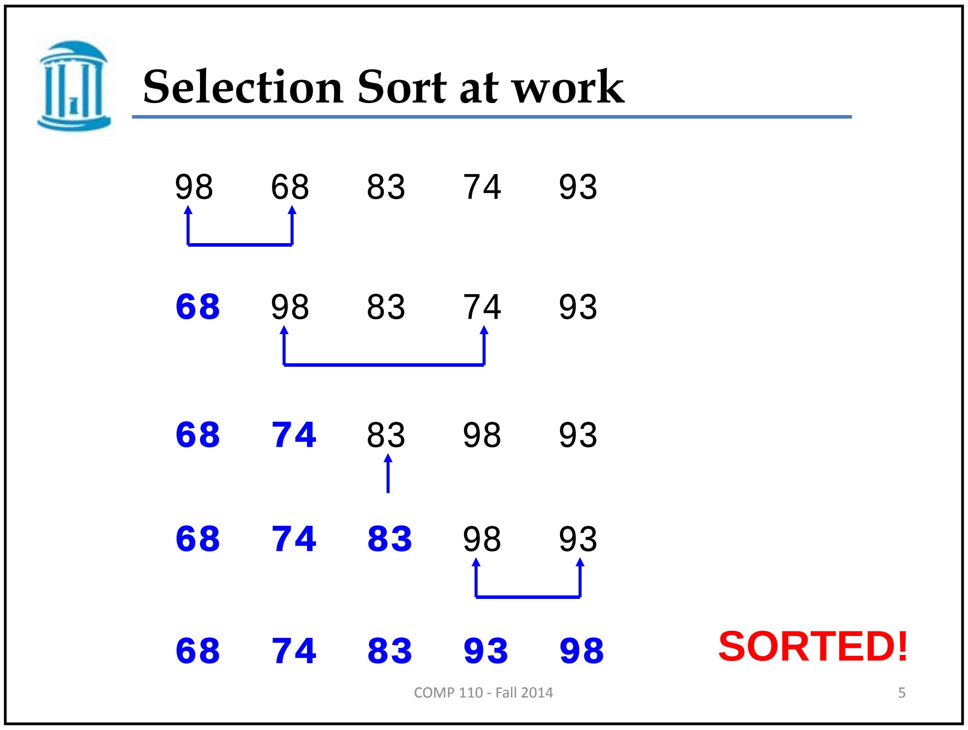
- What did we discuss?

Announcements

- **Assignment 5: Due Fri, Nov 21**

Selection Sort

- Scan the list to find the smallest value
- Exchange (swap) that value with the value in the first position in the list
- Scan rest of list for the next smallest value
- Exchange that value with the value in the second position in the list
- And so on, until you get to the end of the list



Selection Sort at work

98 68 83 74 93

68 98 83 74 93

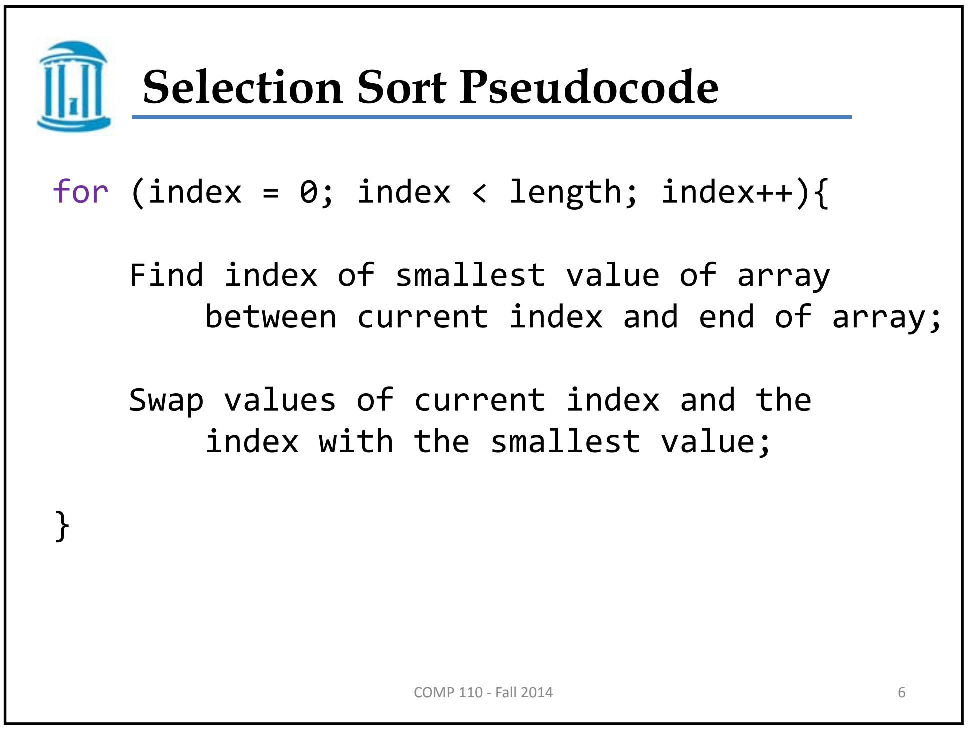
68 **74** 83 98 93

68 **74** **83** 98 93

68 **74** **83** **93** **98**

SORTED!

COMP 110 - Fall 2014 5



Selection Sort Pseudocode

```
for (index = 0; index < length; index++){

    Find index of smallest value of array
        between current index and end of array;

    Swap values of current index and the
        index with the smallest value;

}
```

COMP 110 - Fall 2014 6

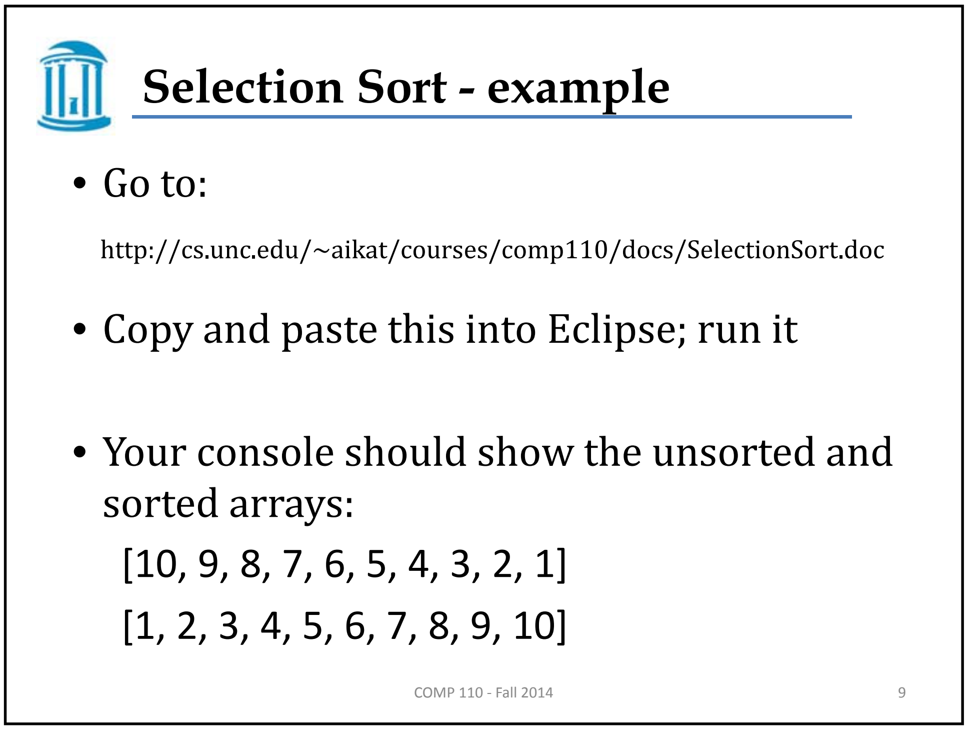
Swap

```
private static void swap(int i, int j, int[] a) {
      int temp = a[i];
      a[i] = a[j];
      a[j] = temp;
}
```

- This method will swap the value of a[i] and a[j]

Selection Sort - example

- Open up Eclipse
- Create a new Java Project – call it **Sorting**
- Create a new class – call it **SelectionSortExample**



Selection Sort - example

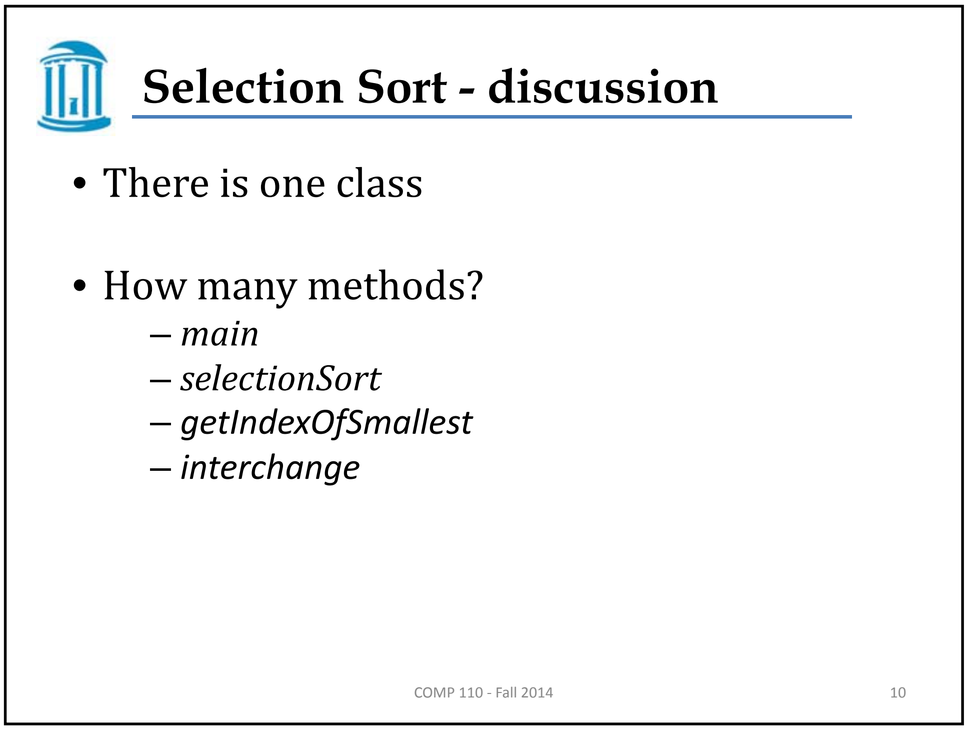
- Go to:

  http://cs.unc.edu/~aikat/courses/comp110/docs/SelectionSort.doc

- Copy and paste this into Eclipse; run it

- Your console should show the unsorted and sorted arrays:

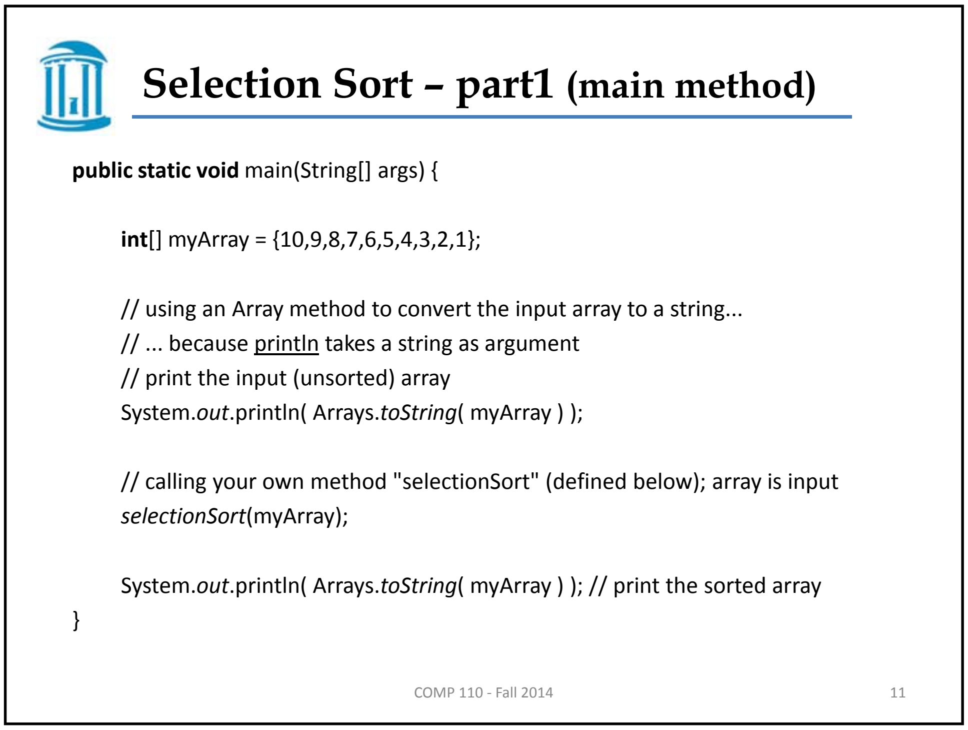
```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```



Selection Sort - discussion

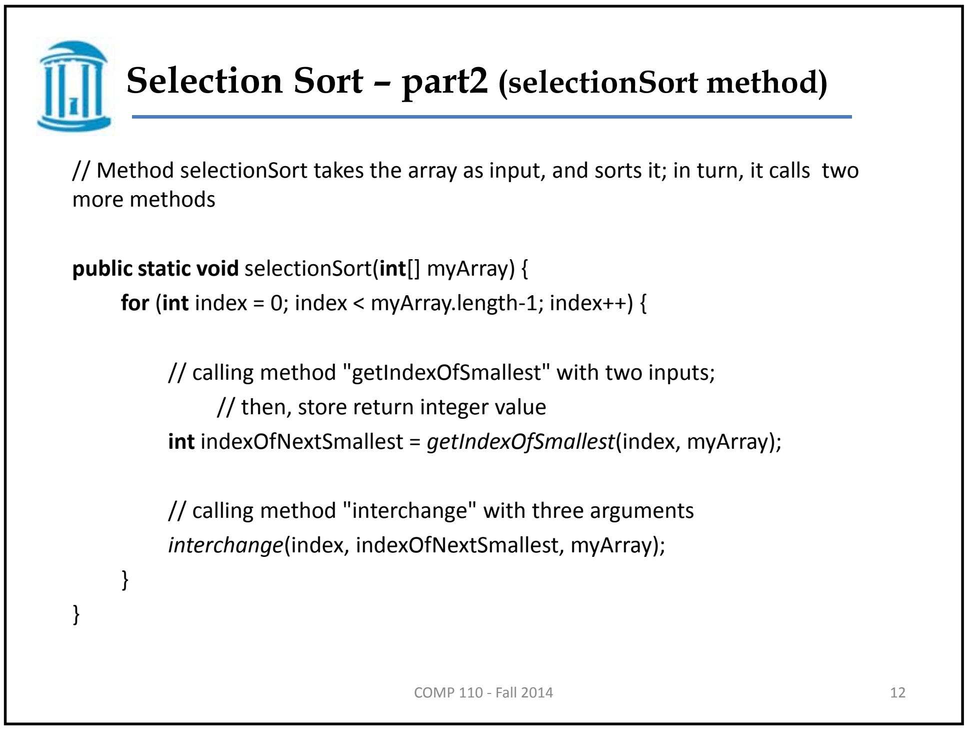
- There is one class

- How many methods?
  - *main*
  - *selectionSort*
  - *getIndexOfSmallest*
  - *interchange*



Selection Sort – part1 (main method)

```
public static void main(String[] args) {

        int[] myArray = {10,9,8,7,6,5,4,3,2,1};

        // using an Array method to convert the input array to a string...
        // ... because println takes a string as argument
        // print the input (unsorted) array
        System.out.println( Arrays.toString( myArray ) );

        // calling your own method "selectionSort" (defined below); array is input
        selectionSort(myArray);

        System.out.println( Arrays.toString( myArray ) ); // print the sorted array
}
```



Selection Sort – part2 (selectionSort method)

// Method selectionSort takes the array as input, and sorts it; in turn, it calls two more methods

```
public static void selectionSort(int[] myArray) {
        for (int index = 0; index < myArray.length-1; index++) {

                // calling method "getIndexOfSmallest" with two inputs;
                        // then, store return integer value
                int indexOfNextSmallest = getIndexOfSmallest(index, myArray);

                // calling method "interchange" with three arguments
                interchange(index, indexOfNextSmallest, myArray);
        }
}
```

Selection Sort – part3 (getIndexOfSmallest)

```
private static int getIndexOfSmallest(int startIndex, int[] a) {

    int min = a[startIndex];
    int indexOfMin = startIndex;

    for (int index = startIndex + 1; index < a.length; index++) {

        if (a[index] < min) {
                min = a[index];
                indexOfMin = index;
        }
    }
    return indexOfMin;

}
```

Selection Sort – part4 (interchange)

```
// Method interchange used to swap the two array elements

private static void interchange(int i, int j, int[] a) {

    int temp = a[i];
    a[i] = a[j];
    a[j] = temp; //original value of a[i]
}
```

Overloading

- We’ve seen that a class can have multiple constructors. Notice that they have the same name

```
public class Pet {
    public Pet()
    {…}
    public Pet(String initName, int initAge, double initWeight)
    {…}
    public Pet(String initName)
    {…}

    public static void main(String[] args) {
        Pet p = new Pet(); // First constructor will be called
        Pet q = new Pet(“Garfield”, 3, 10); // Second constructor
        Pet w = new Pet(“Odie”); // Third constructor
        Pet u = new Pet(“Nermal”, 2); // Wrong – no matching method
    }
```

Overloading

- Using the same method name for two or more methods *within the same class*
  – It’s not only for constructors
- Parameter lists must be different
  – `public double average(int n1, int n2)`
  – `public double average(double n1, double n2)`
  – `public double average(double n1, double n2, double n3)`
- Java knows what to use based on the number and types of the arguments

Overloading

- Java knows what to use based on the number and types of the arguments
  - You’ve used overloading before
    - `System.out.println(“The result is”);`
      `// String type parameter`
    - `System.out.println(20);`
      `// int type parameter`
- Java makes the decision based on a method’s **signature**

Method Signature

- The signature includes a **method’s name** and the **number and types** of its **parameters**
  - `Pet q = new Pet(“Garfield”, 3, 10);`
  - `Pet w = new Pet(“Odie”);`
- Signature does NOT include return type
  - Cannot have two methods with the same signature in the same class
  - `public double average(int n1, int n2)`
  - `public int average(int n1, int n2) // Wrong overloading`
  - Java won’t know what method to call if average(1,2) is invoked

Overloading and Type Conversion

- Java always tries to find an exactly matching method. If it fails, it tries type conversion
  - If a class has the following two methods:
  - `public double average(int n1, int n2)`
  - `public double average(double n1, double n2)`
    - If the method call is average(3,3), the first method will be called
  - However, if a class only has this method:
  - `public double average(double n1, double n2)`
    - If the method call is average(3,3), it will be converted to average(3.0,3.0) and call the (only) method
  - Recall: byte->short->int->long->float->double

How to Use Overloading

- Use it only if two or more methods are performing exactly the same function
  - `public void setPet(String newName)`
  - `public void setPet(String newName, int newAge, double newWeight)`
- It is a very bad idea to create methods that have the same name but do different things
  - `public void setPet(int newAge)`
  - `public void setPet(double newWeight)`
  - What happens if we call setPet(3)? What about setPet(3.0)?
    - Use setAge() and setWeight() instead
    - Usually we do not overload methods if parameters can be converted

Visibility Modifiers

- *public* visibility
  - can be accessed from anywhere
- *private* visibility
  - can only be accessed from inside the class (inside the same Java source file)
- default visibility
  - members declared without a visibility modifier
  - can be accessed by any class in the same package

```
public class Rectangle
{
    private int length;
    private int width;

    public Rectangle ()
    {
             length = 0;
             width = 0;
    }

    ...
}
```

Using a Class

```
public class Student {
    public String name;
    public int classYear;
    public double GPA;
    public String major;

// ...

    public String getMajor() {
         return major;
    }

    public void increaseYear() {
         classYear++;
    }
}
```

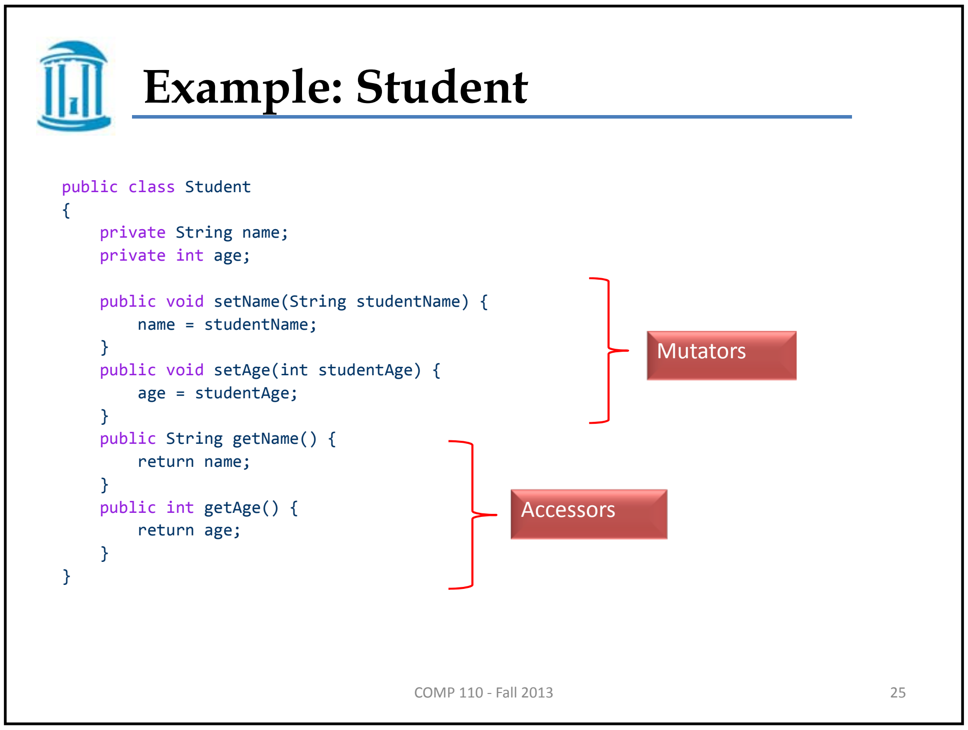
```
public class StudentTest {
  public static void main(String[] args) {
     Student jack = new Student();
     jack.name = "Jack Smith";
     jack.major = "Computer Science";
     jack.classYear = 1;
     jack.GPA = 3.5;

     String m = jack.getMajor(); //
     System.out.println("Jack's major is " + m);

     jack.increaseYear();

     System.out.println("Jack's class year is now
" + jack.classYear);

  }
}
```

Instance Variable and Local Variable

- Instance variables
  - Declared in a class
  - Confined to the class
    - Can be used anywhere in the class that declares the variable, including inside the class' methods
- Local variables
  - Declared in a method
  - Confined to the method
    - Can only be used inside the method that declares the variable

Accessors and Mutators

- How do you access private instance variables?
- Accessor methods (a.k.a. get methods, **getters**)
  - Allow you to look at data in private instance variables
- Mutator methods (a.k.a. set methods, **setters**)
  - Allow you to change data in private instance variables

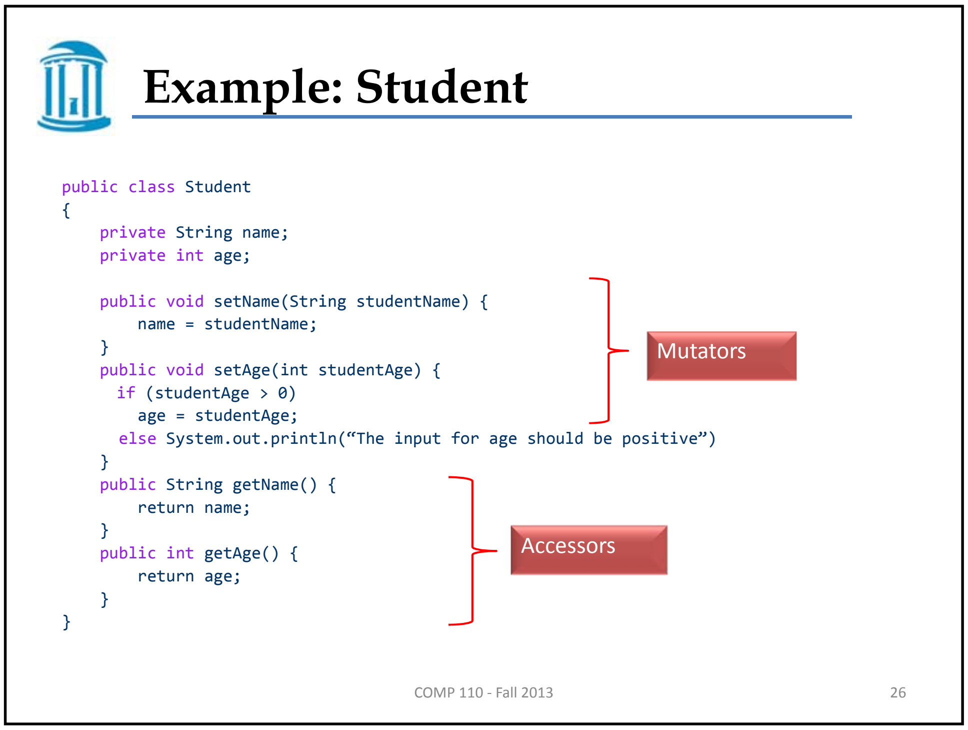


Example: Student

```
public class Student
{
    private String name;
    private int age;

    public void setName(String studentName) {
        name = studentName;
    }
    public void setAge(int studentAge) {
        age = studentAge;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}
```

Mutators

Accessors



Example: Student

```
public class Student
{
    private String name;
    private int age;

    public void setName(String studentName) {
        name = studentName;
    }
    public void setAge(int studentAge) {
      if (studentAge > 0)
        age = studentAge;
      else System.out.println("The input for age should be positive")
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}
```

Mutators

Accessors

Private Methods

- **Why make methods private?**
- Helper methods that will only be used from inside a class should be private
  - External users have no need to call these methods

The Modifier Static

- In the method heading, specifies that the method can be invoked by using the name of the class
  - no object has to be created in order to use the method
  - can't call a non-static method from a static method
  - can't access non-static variables from a static method
- If used to declare data member, data member invoked by using the class name
  - no object has to be created in order to use the variables

Static Variables

- Shared among all objects of the class (e.g. name, midterm, finalExam)
- Memory created for static variables when class is loaded
  - memory created for instance variables (non-static) when an object is instantiated (using `new`)
- If one object changes the value of the static variable, it is changed for all objects of that class

Static Variables and Methods

- Instance variables

```
private int age;
private String name;
```

- Methods

```
public int getAge()
{
    return age;
}
```

- Calling methods on objects

```
Student std = new Student();
std.setAge(20);
System.out.println(std.getAge());
```

Static Variables and Methods

- Recall that "classes do not have data; individual objects have data"
- This is not always true – classes can have data, too
  - **static** variables and methods **belong to a class as a whole**, not to an individual object
  - When would you want a method that does not need an object?
    - If the method perform a general function instead of actions on an object

Static Variables and Methods

```
// Returns x raised to the yth power, where y >= 0.
public int pow(int x, int y)
{
    int result = 1;
    for (int i = 0; i < y; i++)
    {
        result *= x;
    }
    return result;
}
```

Do we need an object to call this method?

Static Variables and Methods

- We have seen static variables and methods before
  - `private static final int DIAMETER = 200;`
    - Recall that “final” means “not changeable”
  - `public static void main(String[] args)`
  - Static can describe more than constants and main method
    - Static variables are sometimes referred as “global variables”, which record the global status of all objects in the same class
    - Static methods are used for actions that do not relate to a certain object
      - main method is a static method because if you execute a program, this entrance is not owned by an object

Instance vs. Static

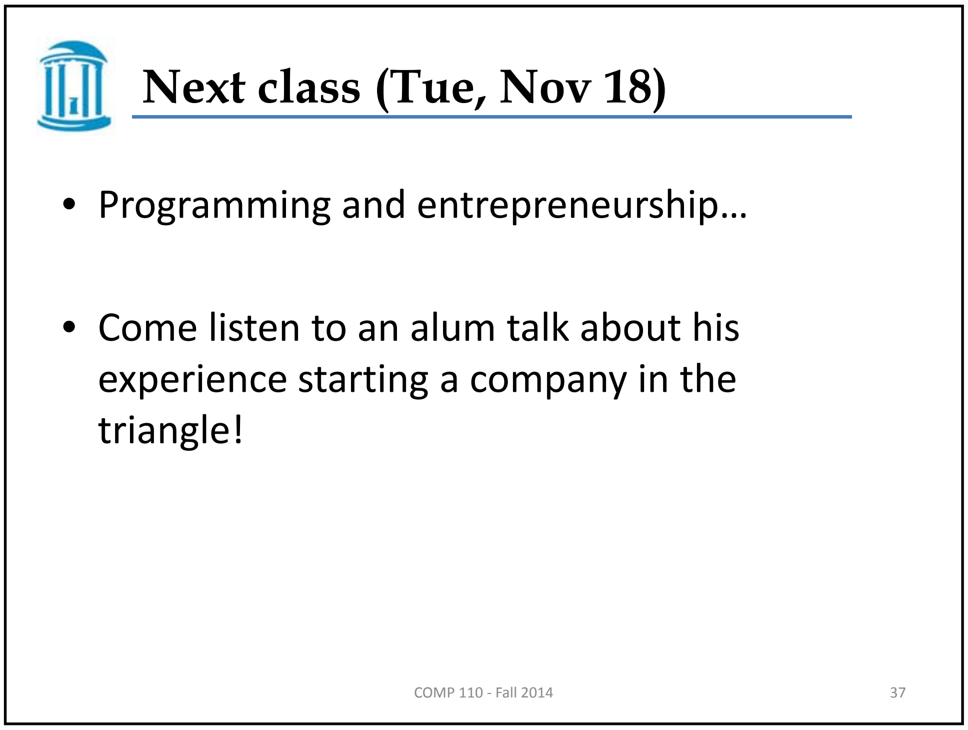
- Instance variables and methods
  - `private int name;`
  - `public void setName(String newName){}`
- Static variables and methods
  - `private static int totalNumber;`
  - `public static int getTotalNumber(){}`

Instance vs. Static

- In an instance method
  - Instance variables/methods can be called
  - Static variables/methods can also be called
    - Eg: you can call a static method pow(x,y) anywhere in a class
- In a static method
  - **Only** static variables/methods can be called
  - Instance variables/methods can be only called if they are invoked from an object
    - Instance variables include “**this**”

Summary: Static Variables/Methods

- Static variables and methods belong to a class instead of an object
- Every object has its own instance variables; all objects in the same type share the same static variables
- Pay attention to: what can be accessed in different methods



Next class (Tue, Nov 18)

- Programming and entrepreneurship…
- Come listen to an alum talk about his experience starting a company in the triangle!

COMP 110 - Fall 2014 37