THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

COMP 110
Introduction to Programming

Fall 2015

Time: TR 9:30 – 10:45

Room: AR 121 (Hanes Art Center)

Jay Aikat

FB 314, *aikat@cs.unc.edu*

Previous Class

- What did we discuss?

Today

- Announcements
  - Lab 2 : Due Wednesday, Oct 21 @ 11:55 PM
- Midterm solution: Sakai → Resources
- Arrays
- Classes and Methods

2D Arrays

- Arrays having more than one index are often useful
  - Tables
  - Grids
  - Board games

	0: Open	1: High	2: Low	3: Close
0: Apple Inc.	99.24	99.85	95.72	98.24
1: Walt Disney Co.	21.55	24.20	21.41	23.36
2: Google Inc.	333.12	341.15	325.33	331.14
3: Microsoft Corp.	21.32	21.54	21.00	21.50

Declaring and Creating 2D Arrays

- Two pairs of square brackets means 2D
  - **int[][] table = new int[3][4];**
- or
  - **int[][] table;**
  - **table = new int[3][4];**

Declaring and Creating 2D Arrays

- Array (or 1D array) gives you a list of variables
  - int[] score = new int[5] gives you score[0], score[1], … , score[5]
- 2D array gives you a table of variables
  - int[][] table = new int[3][4];

table[0][0]	table[0][1]	table[0][2]	table[0][3]
table[1][0]	table[1][1]	table[1][2]	table[1][3]
table[2][0]	table[2][1]	table[2][2]	table[2][3]

Using a 2D Array

- We use a loop to access 1D arrays

```
for (int i = 0; i < 5; i++) {
        scores[i] = keyboard.nextInt();
        scoreSum += scores[i];
}
```


Using a 2D Array

- We use nested loops for 2D arrays

```
int[][] table = new int[4][3];
for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 3; j++) {
                table[i][j] = i * 3 + j;
                System.out.println(table[i][j]);
        }
}
```

Multidimensional Arrays

- You can have more than two dimensions
  - `int[][][] cube = new int[4][3][4];`
- Use more nested loops to access all elements
  - for (int i…)
    - for (int j…)
      - for (int k…)

Todays topic : classes

```
public class Program1 {

      public static void main(String[] args) {
            System.out.println("Hello World");
      }

}
```

The start of a class

Classes and Objects

- Java programs (and programs in other object-oriented programming languages) consist of objects of various class types
  - Objects can represent objects in the real world
    - Automobiles, houses, students
  - Or abstract concepts
    - Colors, shapes, words
- *When designing a program, it's important to figure out what is a class/object in your program*

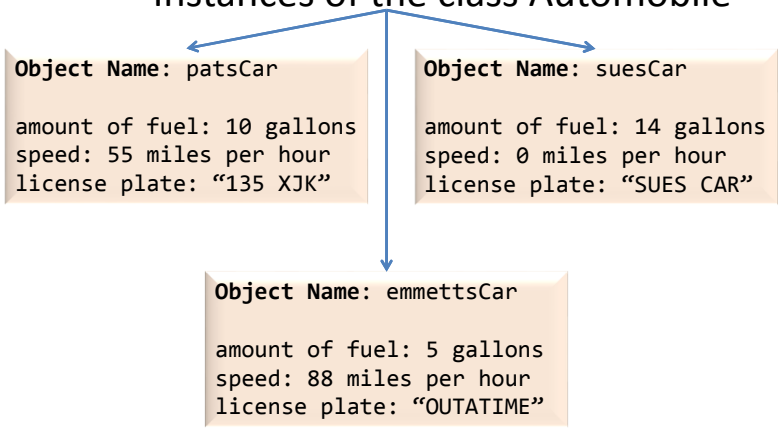
Class

- A *class* is the definition of a kind of object
  - A blueprint for constructing specific objects

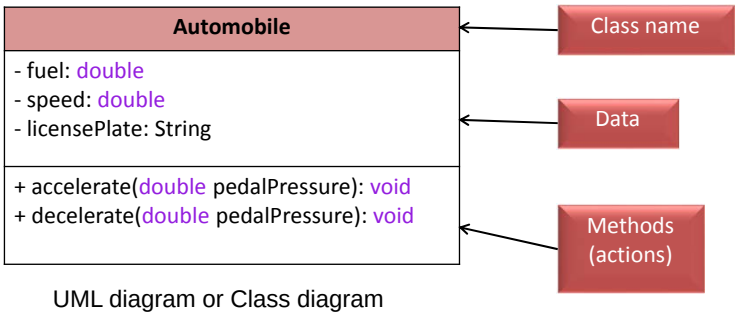
```
Class Name: Automobile

Data:
  amount of fuel
  speed
  license plate

Methods (actions):
  accelerate:
    Action: increase speed
  decelerate:
    Action: decrease speed
```

Objects (Instances)

- Instances of the class Automobile



UML (Universal Modeling Language)



UML diagram or Class diagram

Class Files and Separate Compilation

- Each Java class definition goes in its own .java file
- For a class named ClassName, you should save the file as **ClassName.java**
- Student.java shall and must include the class Student

Class Files and Separate Compilation

- What happens when you compile a .java file?
  - .java file gets compiled into a .class file
    - Contains Java bytecode (instructions)
    - Same filename except for .class instead of .java
- You must compile a Java class before you or a program can use it
- You can send the .class file to people who use it, without revealing your actual code

Class Student

- A general UML class specification

Class Name: Student
- Name - Year - GPA - Major - Credits - GPA sum
+ getName + getMajor + printData + increaseYear Action: increase year by 1

Class Student

- A detailed UML class specification (in Java)

Class Name: Student
- name: String - year: int - gpa: double - major: String - credits: int - gpaSum: double
+ getName(): String + getMajor(): String + printData(): void + increaseYear(): void

Defining 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++;
    }
}
```

Class name

Data
(or attributes, or instance variables)

Methods

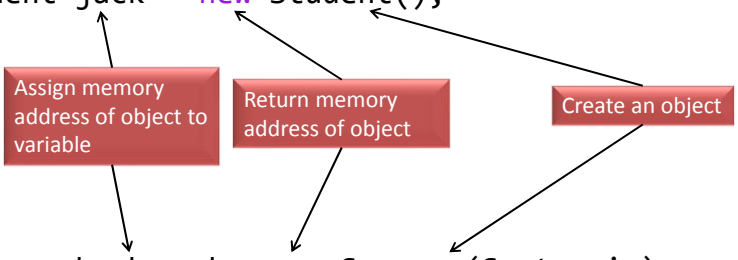
Creating an Object

- Syntax
  - **ClassName objectName = new ClassName();**
- What does the statement do?
  - The computer will create a new object, and assign its memory address to **objectName**
  - **objectName** is sometimes called a class type variable
    - It is a variable of class type **ClassName**
- Why do we need new?
  - So we know ClassName() is not executing a method but creating an object

Creating an Object

Create an object *jack* of class *Student*

```
Student jack = new Student();
```




```
Scanner keyboard = new Scanner(System.in);
```

Create an object *keyboard* of class *Scanner*

Instance Variables

- Data defined in the class are called *instance variables*

```
public String name;
public int    classYear;
public double gpa;
public String major;
```

variable names

public: no restrictions on how these instance variables are used (more details later – public is actually a bad idea in some cases)

type: int, double, String…

Using Instance Variables Inside 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++;
    }
}
```

Any instance variables can be freely used inside the class definition

Using `public` Instance Variables Inside a Class

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

    Student apu = new Student();
    apu.name = "Apu Nahasapeemapetilon";
    apu.major = "Biology";


    System.out.println(jack.name + " is majoring in " + jack.major);
    System.out.println(apu.name + " is majoring in " + apu.major);
}
```

Public instance variables can be used outside the class

You must use the object name to *invoke* the variable

- *jack.name* and *apu.name* are two different instance variables because they belong to different objects

Methods

```
public class Student
{
    public String name;
    public int classYear;
    public double gpa;
    public String major;
    // ...

    public String getMajor()
    {
        return major;
    }

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

Methods

- Two kinds of methods
  - Methods that return a value
    - Examples: String's **substring()** method, String's **indexOf()** method, String's **charAt()** method, etc.
  - Methods that return nothing
    - Example: **System.out.println()**
- "Return" means "give back"
  - A method can give back a value so that other parts of the program can use it, or simply perform some actions

Defining Methods that Return a Value

- Method heading: keywords
  - public: no restriction on how to use the method (more details later)
  - *Type*: the type of value the method returns
- Method body: statements executed
  - **Must be inside a pair of brackets**
  - **Must have a return statement**

```
public String getMajor()
{
    return major;
}
```


return Statement

- A method that returns a value must have *at least one* return statement
- Terminates the method, and returns a value
- Syntax:
  - return expression;
- expression can be any expression that produces a value of type specified by **the return type** in the method heading

Methods that Return a Value

As usual, inside a block (defined by braces), you can have multiple statements

```
public String getClassYear()
{
    if (classYear == 1)
        return "Freshman";
    else if (classYear == 2)
        return "Sophomore";
    else if ...
}
```


Calling Methods that Return a Value

- Object, followed by dot, then method name, then ()
  - **objectName.methodName();**
- Use them as a ***value*** of the type specified by the method's return type

```
Student jack = new Student();
jack.major = "Computer Science";

String m = jack.getMajor(); // Same as String m = "Computer Science"

System.out.println("Jack's full name is " + jack.getName());
// Same as System.out.println("Jack's full name is " + "Jack Smith");
System.out.println("Jack's major is " + m);
```

Defining Methods That Return Nothing

- Method heading: keywords
  - public: no restriction on how to use the method (more details later)
  - void: the method returns nothing
- Method body: statements executed when the method is called (invoked)
  - **Must be inside a pair of brackets**

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


Methods that Return Nothing

```
public void printData()
{
    System.out.println("Name: " + name);
    System.out.println("Major: " + major);
    System.out.println("GPA: " + gpa);
}
```

Calling Methods that Return Nothing

- Object, followed by dot, then method name, then ()
  - The same as a method that returns a value
  - **objectName.methodName();**
- Use them as *Java* ***statements***

```
Student jack = new Student();
jack.classYear = 1;

jack.increaseYear();

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


return Statement in void Methods

- Can also be used in methods that return nothing
- Simply terminates the method
- Syntax:
  - return;

```
public void increaseYear()
{
    if (classYear >= 4)
        return;
    classYear++;
}
```

Next class

- More on Classes and Methods