THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

COMP 110
Introduction to Programming

Tuesday October 21, 2014

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

Previous Class

- What did we discuss?
  - Hint: Arrays!

Announcements

- **Midterm – grades released**
- **Midterms will be returned on Thu, Oct 23**
  - **Discuss solutions then**

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 – you can never copy the real world*

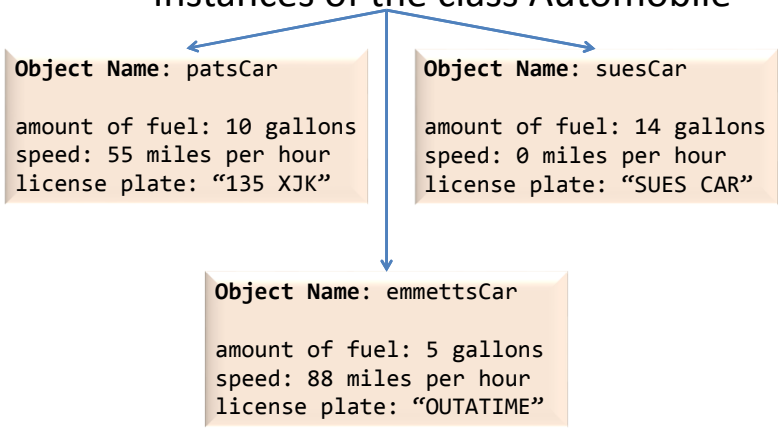
Class

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

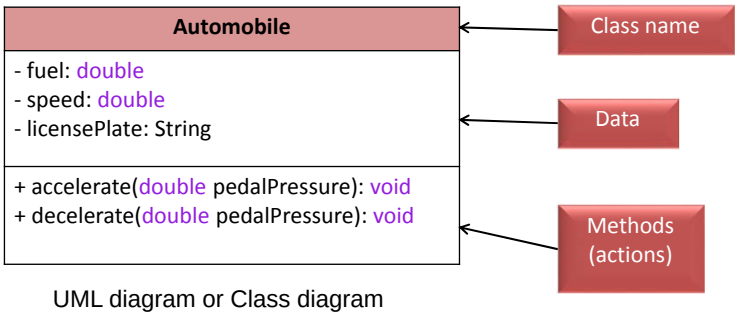
```
Class Name: Automobile

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 your 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

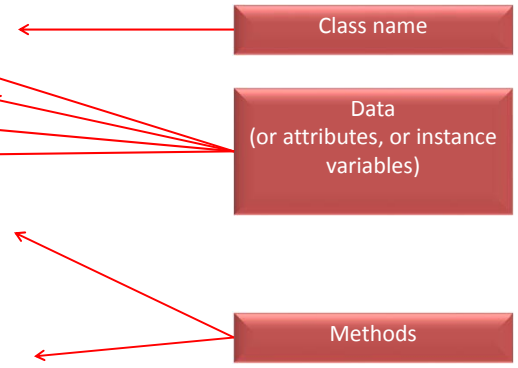
Class Student

- 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++;
    }
}
```



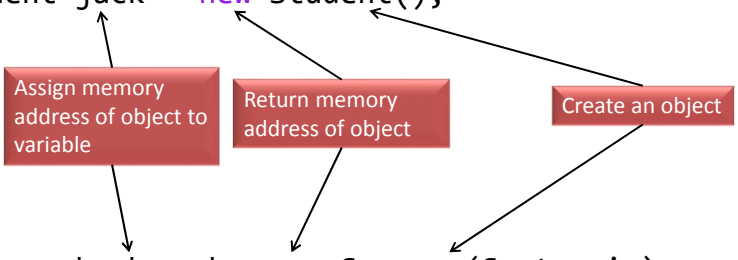
Creating an Object

- 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++;
}
```

Static Variables

- Static variables are shared by all objects of a class
- Only one instance of the variable exists
- It can be accessed by all instances of the class

```
public double gpa;
public static double highestGPA = 0.0;
public void setGPA(double newGPA) {
        if (newGPA > Student.highestGPA) {
                Student.highestGPA = newGPA;
        }
        gpa = newGPA;
}
```


Static Variables

- Static variables also called *class variables*
  - Contrast with *instance variables*
- Do not confuse class variables with variables of a class type
- Both static variables and instance variables are sometimes called *fields* or *data members*

Final Static Variables

- Variables declared **`static final`** are considered constants – value cannot be changed

```
public static final MAX_CLASS_YEAR = 6;
```

- Now, this won't work

```
public static void main(String[] args) {
   …
   Student.MAX_CLASS_YEAR = 12;
   …
}
```


Static Methods

- Some methods may have no relation to any type of object
- Example
  - Compute max of two integers
  - Convert character from upper- to lower case
- Static method declared in a class
  - Can be invoked without using an object
  - Instead use the class name

The Math Class

- Provides many standard mathematical methods
  - Automatically provided, no import needed
- Example methods, figure 6.3a

Name	Description	Argument Type	Return Type	Example	Value Returned
pow	Power	double	double	Math.pow(2.0,3.0)	8.0
abs	Absolute value	int, long, float, or double	Same as the type of the argument	Math.abs(-7) Math.abs(7) Math.abs(-3.5)	7 7 3.5
max	Maximum	int, long, float, or double	Same as the type of the arguments	Math.max(5, 6) Math.max(5.5, 5.3)	6 5.5

The Math Class

- Example methods

Name	Description	Argument Type	Return Type	Example	Value Returned
min	Minimum	int, long, float, or double	Same as the type of the arguments	Math.min(5, 6) Math.min(5.5, 5.3)	5 5.3
round	Rounding	float or double	int or long, respectively	Math.round(6.2) Math.round(6.8)	6 7
ceil	Ceiling	double	double	Math.ceil(3.2) Math.ceil(3.9)	4.0 4.0
floor	Floor	double	double	Math.floor(3.2) Math.floor(3.9)	3.0 3.0
sqrt	Square root	double	double	sqrt(4.0)	2.0

Random Numbers

- `Math.random()` returns a random double that is greater than or equal to zero and less than 1
- Java also has a `Random` class to generate random numbers
- Can scale using addition and multiplication; the following simulates rolling a six sided die

```
int die = (int) (6.0 * Math.random()) + 1;
```


Next class (Thu, Oct 23)

- More on methods…