COMP 110-003
Introduction to Programming
Classes

February 19, 2013

Haohan Li
TR 11:00 – 12:15, SN 011
Spring 2013

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

What We've Learned So Far?

- Types and variables
  - int, double, char, String
- Branching statements
  - If, if-else, switch
- Loop Statements
  - While, do-while, for

Still, Procedural Programming

- Types and variables
  - How to save data
- Branching statements
  - If…then…
- Loop Statements
  - Repeat
- **Basically, we've learned how to manipulate data by programming – in a procedural manner**

Object-Oriented Programming

- Object-oriented programming (OOP) helps people to organize code and programs
  - How to organize data?
  - How to organize manipulations of data?
- OOP uses classes and objects to get good organization

Reusability

- How does good organization (or usually called "good design") help you?
  - If I can make it work, it is a good design?
- Good design means better **reusability**
  - You can use part of your program in another program
  - You can use part of your program in a new version
    - You can change only one part if you know other parts are good
  - Others can use part of, or the whole of your program
    - They don't even have to know the details if they trust you
    - That's how programmers collaborate

Reusability

- You have seen many program components that you can use without knowing the details
  - Scanner
    - next(), nextLine(), nextInt()
  - String
    - length(), indexOf(), substring(), trim()
- Scanner class has more than 1500 lines of code
  - But you can use it without copying a single line

Reusability

- Do you have to rewrite all you code if you need a program that can deal with 3 operands?
  - 2.5 + 3 + 3.5 = ?
- What if you have a piece of code that always returns the next word in the input string?
- What if you have a piece of code that records the current result and can calculate new results with new operators?
- Can you easily support 4 operands then?

Reusability

- The rules of reusability
  - Generic design
    - A component (a class in Java) should perform a general function
  - High cohesion
    - What's in a class (data and methods) should be closely related to each other
  - Low coupling
    - Classes should be independent of other classes

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, employee records
  - Or abstract concepts
    - Colors, shapes, words
- *When designing a program, it's important to figure out what is a class/object in your program – again, you can never copy a real world*

Class

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

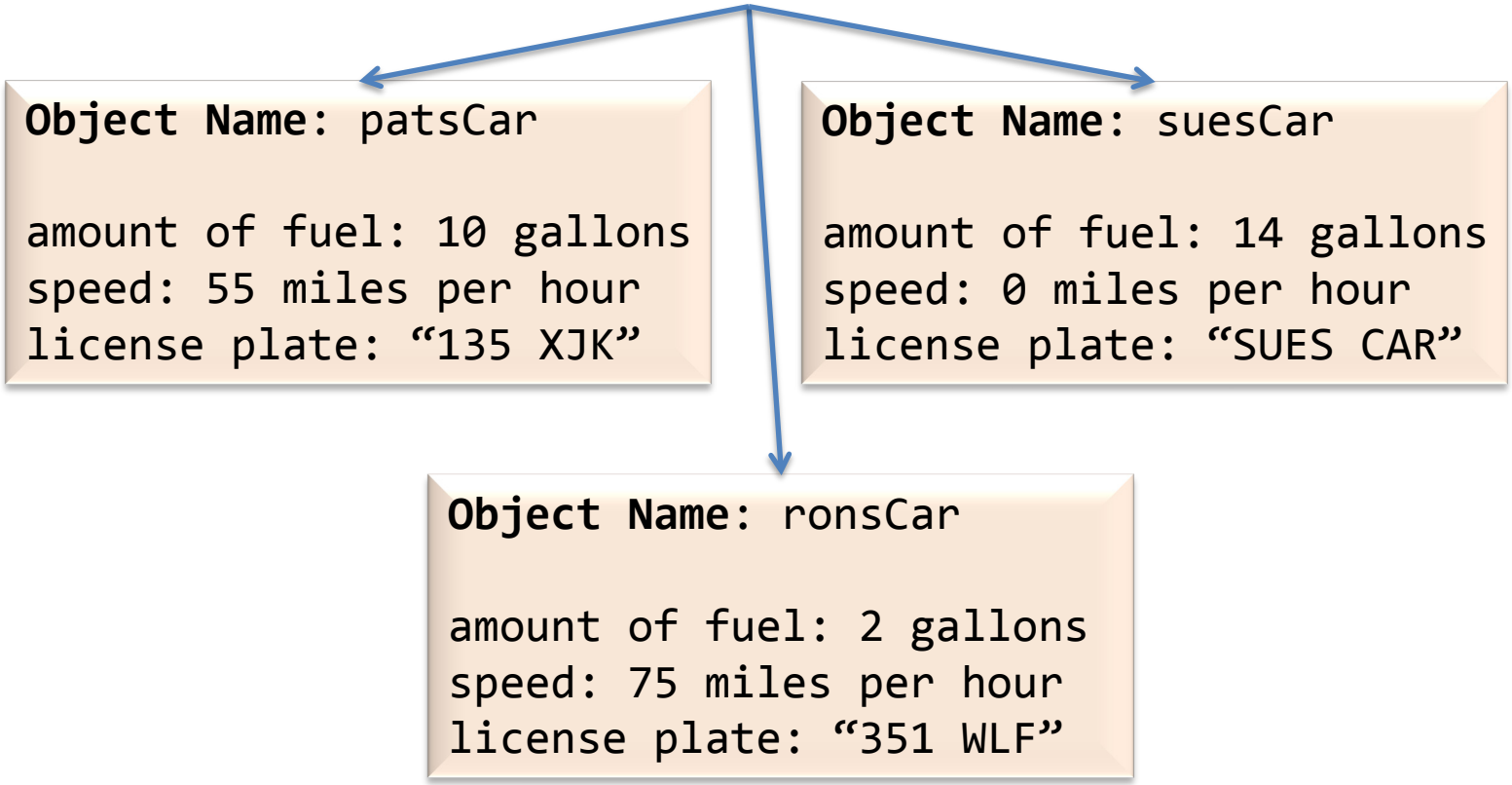
```
Class Name: Automobile

Data:
  amount of fuel
  speed
  license plate

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

Objects (Instances)

- Instances of the class Automobile



Objects

- **Important**: classes do not have data; individual objects have data
- Classes specify what kind of data objects have

Objects

- Only objects have the actual data

Object Name: patsCar

amount of fuel: 10 gallons
speed: 55 miles per hour
license plate: “135 XJK”

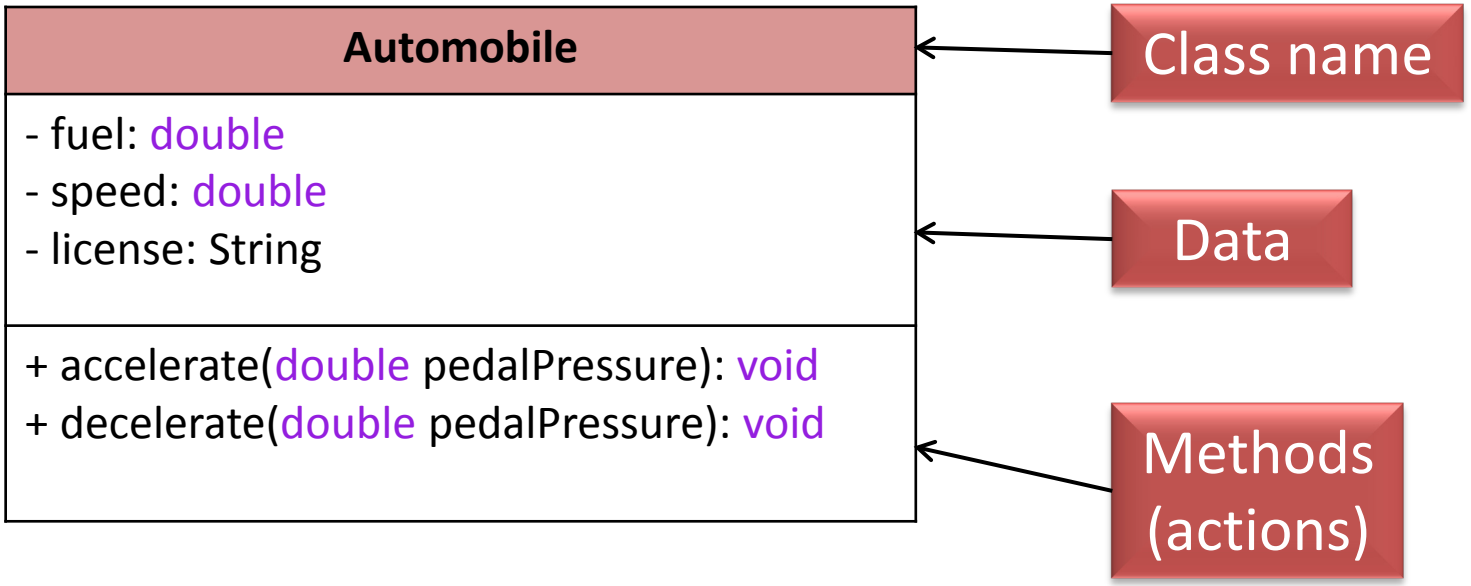
Object Name: suesCar

amount of fuel: 14 gallons
speed: 0 miles per hour
license plate: “SUES CAR”

Object Name: ronsCar

amount of fuel: 2 gallons
speed: 75 miles per hour
license plate: “351 WLF”

UML (Universal Modeling Language)



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 can compile a Java class before you have a program that uses 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)

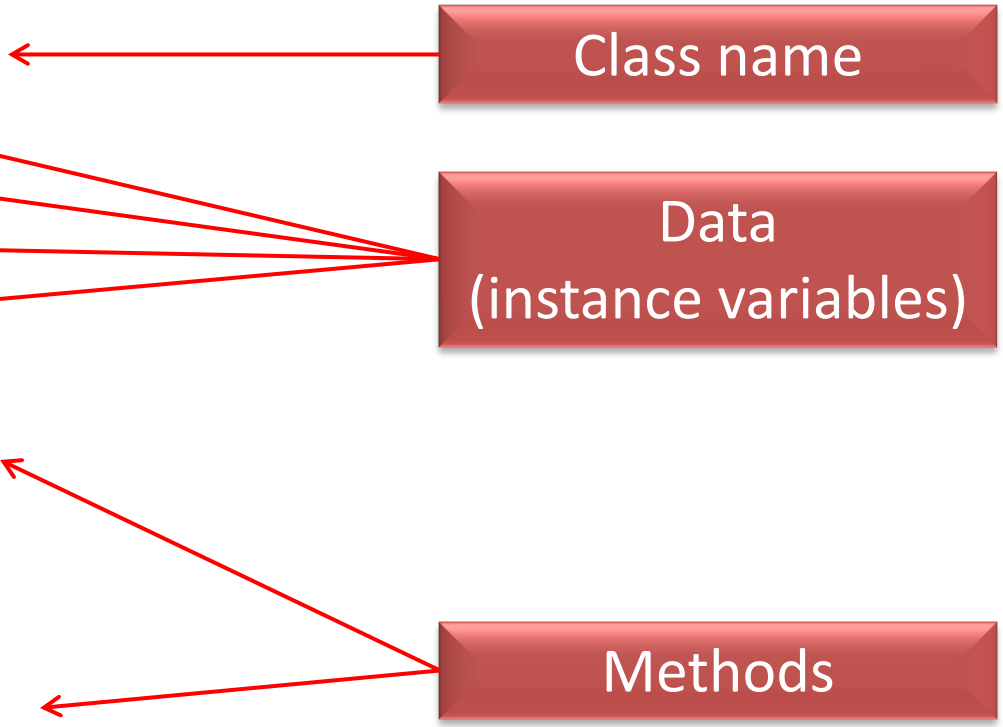
Class Name: Student
- name: String - year: int - gpa: double - major: String - credits: int - gpaSum: double
+ getName(): String + getMajor(): String + printData(): void + increaseYear(): void

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++;
    }
}
```



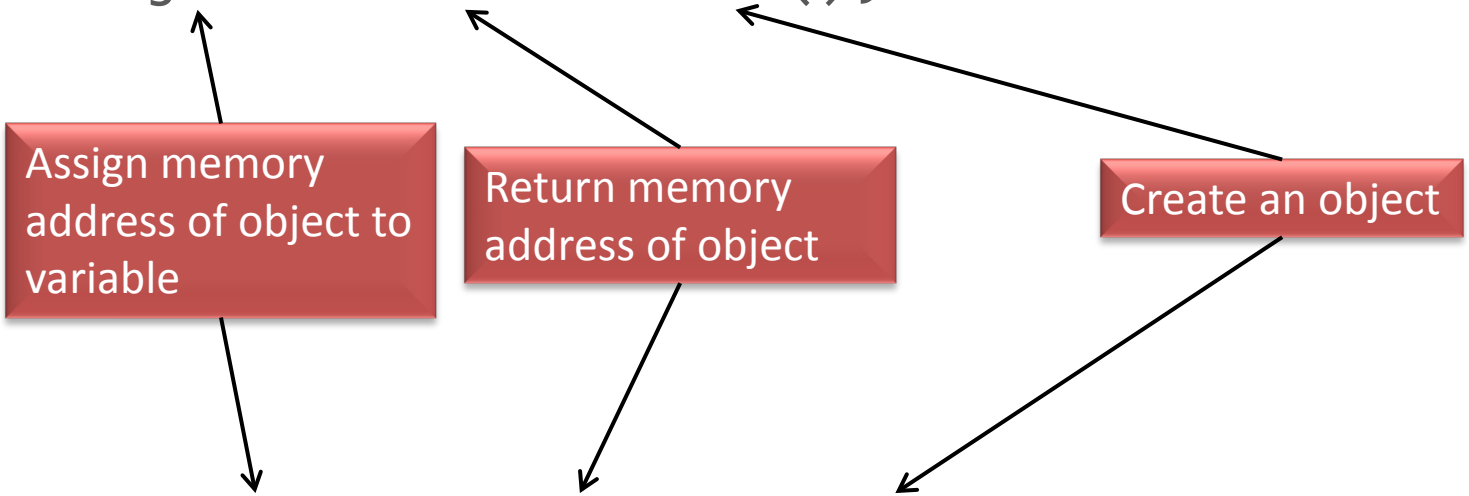
Creating an Object

- Syntax rule
  - **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 an 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 here)

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 (without invoking)

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Using public Instance Variables Outside a Class

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

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

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

    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

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

Methods

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

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

returns a String

return type

returns nothing

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 = "Freshman"

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

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

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


Announcement

- Make sure to run Student.java to understand today's content
- Finish Strings and Loops Review Worksheet before next lecture on Thursday
  - Next lecture will be a general discussion of problem solving skills in programming, and the explanations of the worksheet
- Program 3 will be released soon