

COMP 110-001 Classes

Yi Hong

May 22, 2015

Announcement

- Lab 2 & 3 due today

Review

- Q1: What are the three types of loops? What are their differences?
- Q2: Write a program that maintains the balance of an account
 - Ask for a balance-update from user in each iteration
 - Positive value: deposit
 - Negative value: withdraw
 - If the balance-update is 0 or the balance goes below 0, exit from loop and print out the remaining balance

Sample Code for Q2

```
double currentBalance = 0;
double updatedBalance = currentBalance;
int balanceUpdatedValue;
do
{
    currentBalance = updatedBalance;
    System.out.println("Current balance is " + currentBalance);
    System.out.println("Please input your balance-update: ");
    balanceUpdatedValue = keyboard.nextInt();
    updatedBalance += balanceUpdatedValue;
    if(balanceUpdatedValue > 0)
        System.out.println("Deposit " + balanceUpdatedValue);
    else if(balanceUpdatedValue < 0)
    {
        System.out.println("Withdraw " + (-balanceUpdatedValue));
        if(updatedBalance < 0)
            System.out.println("Low balance, no withdraw");
    }
}while(balanceUpdatedValue != 0 && updatedBalance >= 0);
System.out.println("Done. Current balance is " + currentBalance);
```

num++ v.s. ++num

- num++ does `num = num + 1;`
- So does ++num. But, there is a difference
 - `int num1 = 5;`
 - `System.out.println(num1++);`
 - Outputs num1 (5), then +1
 - `int num2 = 5;`
 - `System.out.println(++num2);`
 - +1, then outputs num2 (6)

Today

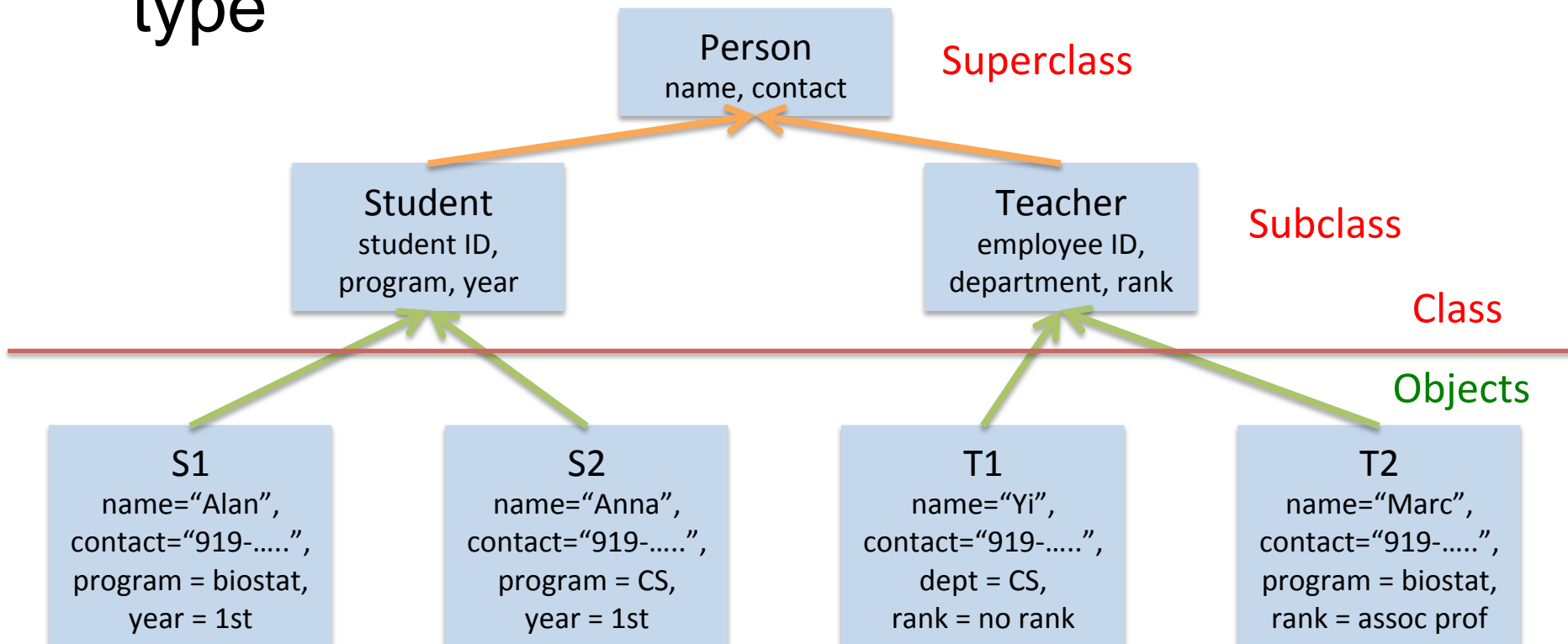
- Classes

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

Object Oriented Programming (OOP)

- Object: Attributes + Methods
- Class: the blueprint of objects of the same type



OOP in Practice

- Import class if necessary
 - E.g.: `import java.util.*;`
- Create object
 - `Class_Type variable_name = new ClassType(...);`
 - E.g.: `Scanner keyboard = new Scanner(System.in);`
`Polygon treeTop = new Polygon();`
- Access object members (attribute or method)
 - `int inputNumber = keyboard.nextInt();`
 - `treeTop.setColor(Color.green);`

Class

- A *class* is the definition of a kind of object
 - A blueprint for constructing specific objects
 - Specifies an object's attributes and defines its behaviors as methods
- Today, we will talk about how to create our own classes

Class Name: Automobile

Data:

amount of fuel _____

speed _____

license plate _____

Methods (actions):

accelerate:

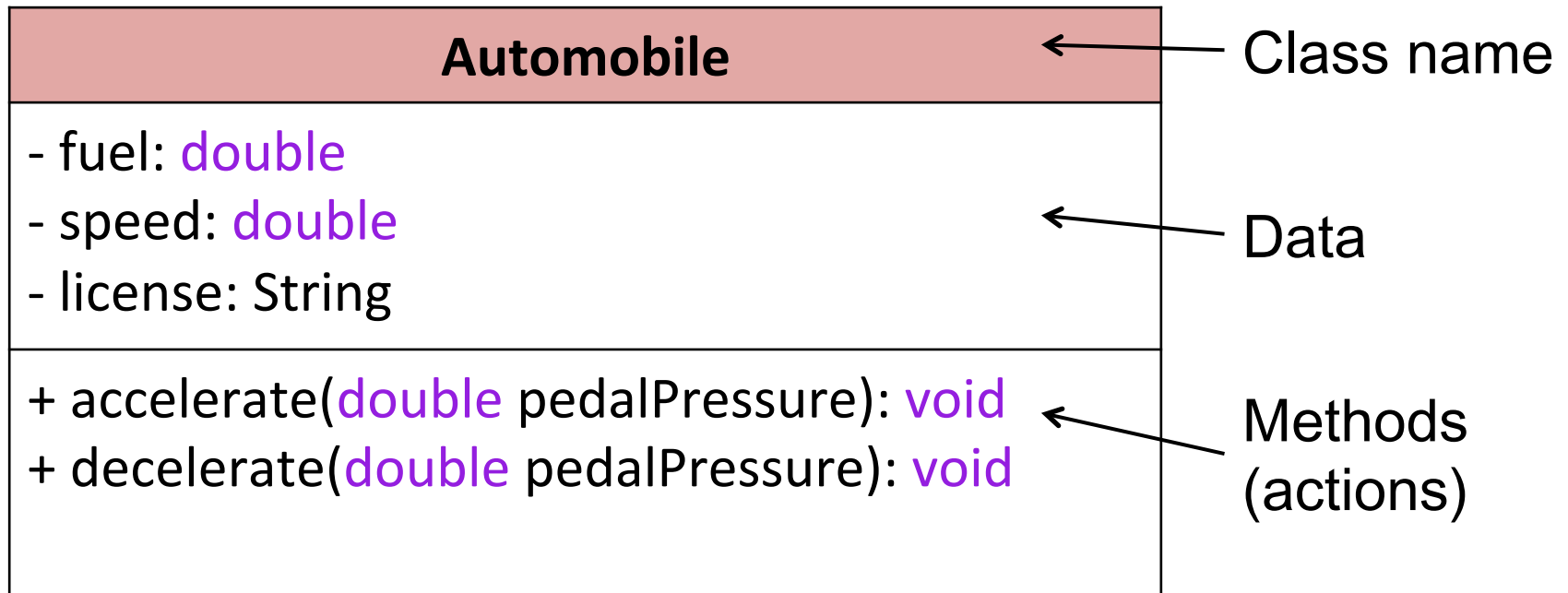
How: Press on gas pedal.

decelerate:

How: Press on brake pedal.

UML (Unified Modeling Language)

- Use a UML class diagram to help design a class



Objects, Instantiation

Object Name: patsCar

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

Object Name: ronsCar

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

Object Name: suesCar

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

Instantiations, or instances, of the class Automobile



Objects

- Classes specify the data type, what kind of data the objects have
- **Important:** classes **usually** do not have data; individual objects have data.
- **But**, a class can have variables that are static as well as methods that are **static**.
- Static variables and static methods belong to a class as a whole and not to an individual object (more discussion later)

Class Files and Separate Compilation

- Each Java class definition goes in its own, it is in a separate file
- `ClassName` → save the file as `ClassName.java`
- E.g.: `Student.java` includes 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
 - The same filename except for .class instead of .java
- You can compile a Java class before you have a program that uses it
- Don't worry about the compilation in this course as Eclipse does it automatically

Example: Class Student

| Class Name: Student | |
|---|--|
| - Name - Year - GPA - Major - Credits - GPA sum | |
| + getName + getMajor + printData + increaseYear How: increase year by 1 + calcGpa How: average grades | |

- : private
+ : public

In this lecture, we focus on **public** first, we will discuss about **private** members later

Example: Class Student

Class Name: Student

- name: String
- year: int
- gpa: double
- major: String
- credits: int
- gpaSum: double

+ getName(): String
+ getMajor(): String
+ printData(): void
+ increaseYear(): void
+ calcGpa(double grade): 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

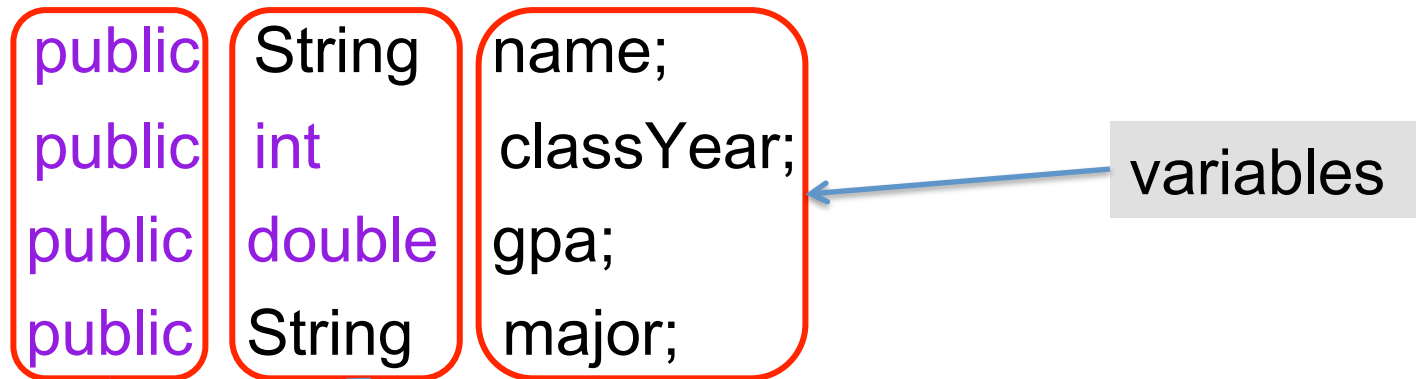
Data
(instance variables)

Methods

Instance variables and
methods are members
of a class

Instance Variables

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



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

Data type: `int`, `double`, `String`...

Using Instance Variables Inside the Class Definition

```
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

- Create an object *jack* of class *Student*

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

Assign memory address
of object to variable

Return memory
address of object

Create an object

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

- Create an object *keyboard* of class *Scanner*

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 lily = new Student();
    lily.name = "Lily Chase";
    lily.major = "Biology";

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

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

Local / Instance Variables

■ Instance variables

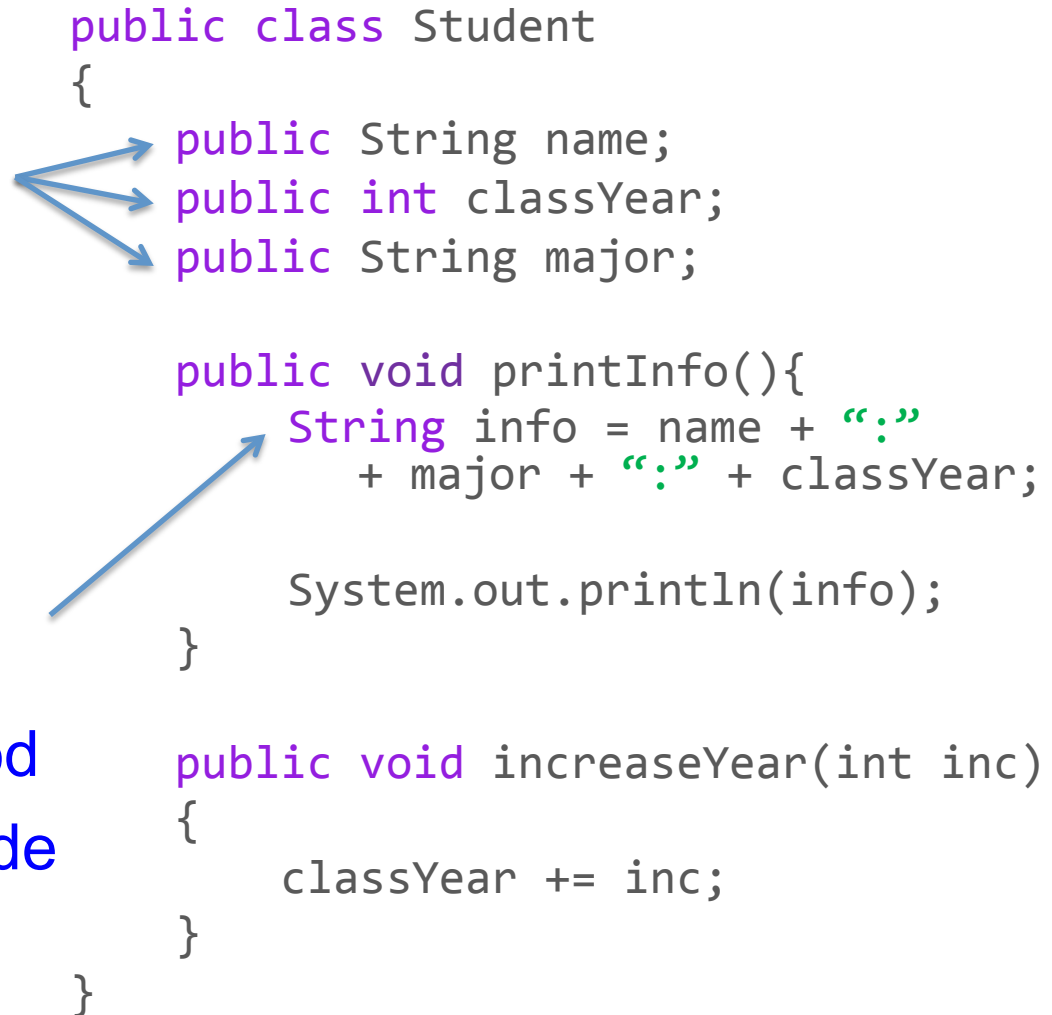
- Declared in a class
- Confined to the class
- Can be used in any method in this class

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

    public void printInfo(){
        String info = name + ":"
            + major + ":" + classYear;

        System.out.println(info);
    }

    public void increaseYear(int inc)
    {
        classYear += inc;
    }
}
```



■ Local variables

- Declared in a method
- Confined to the method
- Can only be used inside the method

An Example

```
public class Student
{
    public String name;
    public int classYear;
    public String major;
    public void printInfo()
    {
        String info = name + ":" + major + ":" + classYear ;
        System.out.println(info);
    }
    public void increaseYear(int inc)
    {
        classYear += inc;
        info = "info changed a bit"; } x
    }
```

- Java will not recognize *info*

An Example

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

    public void printInfo()
    {
        String info = name + ":" + major + ":" + classYear ;
        System.out.println(info);
    }

    public void increaseYear(int inc)
    {
        classYear += inc;
        String info = "classYear updated";
        System.out.println(info);
    }
}
```

- The two variables, *info*, will not affect each other

This will become more clear after we discuss *code block* later

Next Class

- Methods
- Code block