CSC 207

Lecture 2

Question 1) When can a svn conflict occur?

Question 1) When can a svn conflict occur?

- A) Multiple developers working on different files.

Question 1) When can a svn conflict occur?

- A) Multiple developers working on different files.

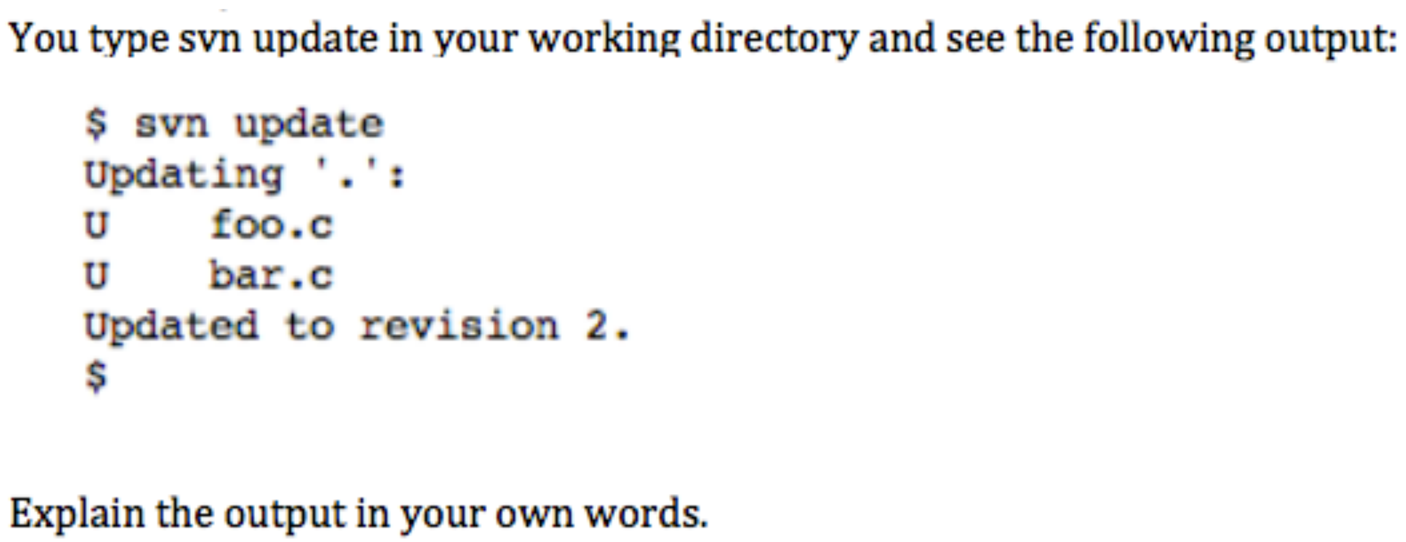
Question 1) When can a svn conflict occur?

- A) Multiple developers working on different files.

- B) Multiple developers working on the same file and on common lines.

Question 2)

•



You type svn update in your working directory and see the following output:

```
$ svn update
Updating '.':
U    foo.c
U    bar.c
Updated to revision 2.
$
```

Explain the output in your own words.

What is a primitive data type in Java?

•

What is a primitive data type in Java?

-
    - The Java programming language supports ***eight*** primitive data types.

What is a primitive data type in Java?

-
    - The Java programming language supports ***eight*** primitive data types.

What is a primitive data type in Java?

-
  - The Java programming language supports ***eight*** primitive data types.

What is a primitive data type in Java?

-
    - The Java programming language supports ***eight*** primitive data types.

What is a primitive data type in Java?

-
    - The Java programming language supports ***eight*** primitive data types.
    - A primitive type is predefined by the Java language and is named by a reserved keyword.

What is a primitive data type in Java?

-
  - The Java programming language supports ***eight*** primitive data types.
  - A primitive type is predefined by the Java language and is named by a reserved keyword.

What is a primitive data type in Java?

-
    - The Java programming language supports ***eight*** primitive data types.

    - A primitive type is predefined by the Java language and is named by a reserved keyword.

What is a primitive data type in Java?

-
  - The Java programming language supports ***eight*** primitive data types.
  - A primitive type is predefined by the Java language and is named by a reserved keyword.

What is a primitive data type in Java?

-
  - The Java programming language supports ***eight*** primitive data types.
  - A primitive type is predefined by the Java language and is named by a reserved keyword.
  - Primitive values do not share state with other primitive values.

What is a primitive data type in Java?

-
    - The Java programming language supports ***eight*** primitive data types.
    - A primitive type is predefined by the Java language and is named by a reserved keyword.
    - Primitive values do not share state with other primitive values.

What is a primitive data type in Java?

-
  - The Java programming language supports ***eight*** primitive data types.
  - A primitive type is predefined by the Java language and is named by a reserved keyword.
  - Primitive values do not share state with other primitive values.

What is a primitive data type in Java?

-
  - The Java programming language supports ***eight*** primitive data types.
  - A primitive type is predefined by the Java language and is named by a reserved keyword.
  - Primitive values do not share state with other primitive values.
  - Primitive types can be directly used ‘out of the box’ by your program without much effort from your end.

int primitive data type

- int number=1;

int is the type (it is a primitive type)
number is the variable
1 is the value that is assigned to number

int type is **32 bit signed** integer.
- It has the minimum value of **-2147483648**
- And a max value of **2147483647**

Other primitive data types

- **byte**: 8 bit signed integer
- **short**: 16 bit signed integer
- **long**: 64 bit signed integer
- **float**: Single precision 32 bit floating point values. (More on this later, week 12)
- **double**: Double precision 64 bit floating point values. (More on this later, week 12)
- **boolean**: Is a 1 bit value. Why is this? What values does a boolean hold? Is 1 bit sufficient for it?
- **char**: 16 bits unsigned integer.

Strings

Strings

- `String s = "Hello World!";`

Strings

- `String s = "Hello World!";`

Strings

- `String s = "Hello World!";`

Strings

- `String s = "Hello World!";`

- String objects are ***immutable***, which means once created ***its value cannot be changed.***

Strings

- `String s = "Hello World!";`

- String objects are ***immutable***, which means once created ***its value cannot be changed.***

Strings

- `String s = "Hello World!";`

- String objects are ***immutable***, which means once created ***its value cannot be changed.***

- Above is an example of a String literal which is more efficient than:

Strings

- `String s = "Hello World!";`

- String objects are ***immutable***, which means once created ***its value cannot be changed.***

- Above is an example of a String literal which is more efficient than:

Strings

- `String s = "Hello World!";`

- String objects are ***<u>immutable</u>***, which means once created ***<u>its value cannot be changed.</u>***

- Above is an example of a String literal which is more efficient than:

- String s= new String (“Hello World”)

Strings

- `String s = "Hello World!";`

- String objects are ***immutable***, which means once created ***its value cannot be changed.***

- Above is an example of a String literal which is more efficient than:

- String s= new String (“Hello World”)

Strings

- `String s = "Hello World!";`

- String objects are ***immutable***, which means once created ***its value cannot be changed.***

- Above is an example of a String literal which is more efficient than:

- String s= new String ("Hello World")

- We will revisit this later in Week4, when we cover Java Memory Model, why the former is more efficient than the latter.

Imagine in a certain class in high school there are 3 students.

- Each of the student is enrolled in 3 courses (Math, Physics, Chemistry).
- Each course is out of 100 marks.
- The professor of this class, likes to get the average mark of each student at the end of the semester.

A small Java program

-

A small Java program

•

A small Java program

-

A small Java program

-

A small Java program

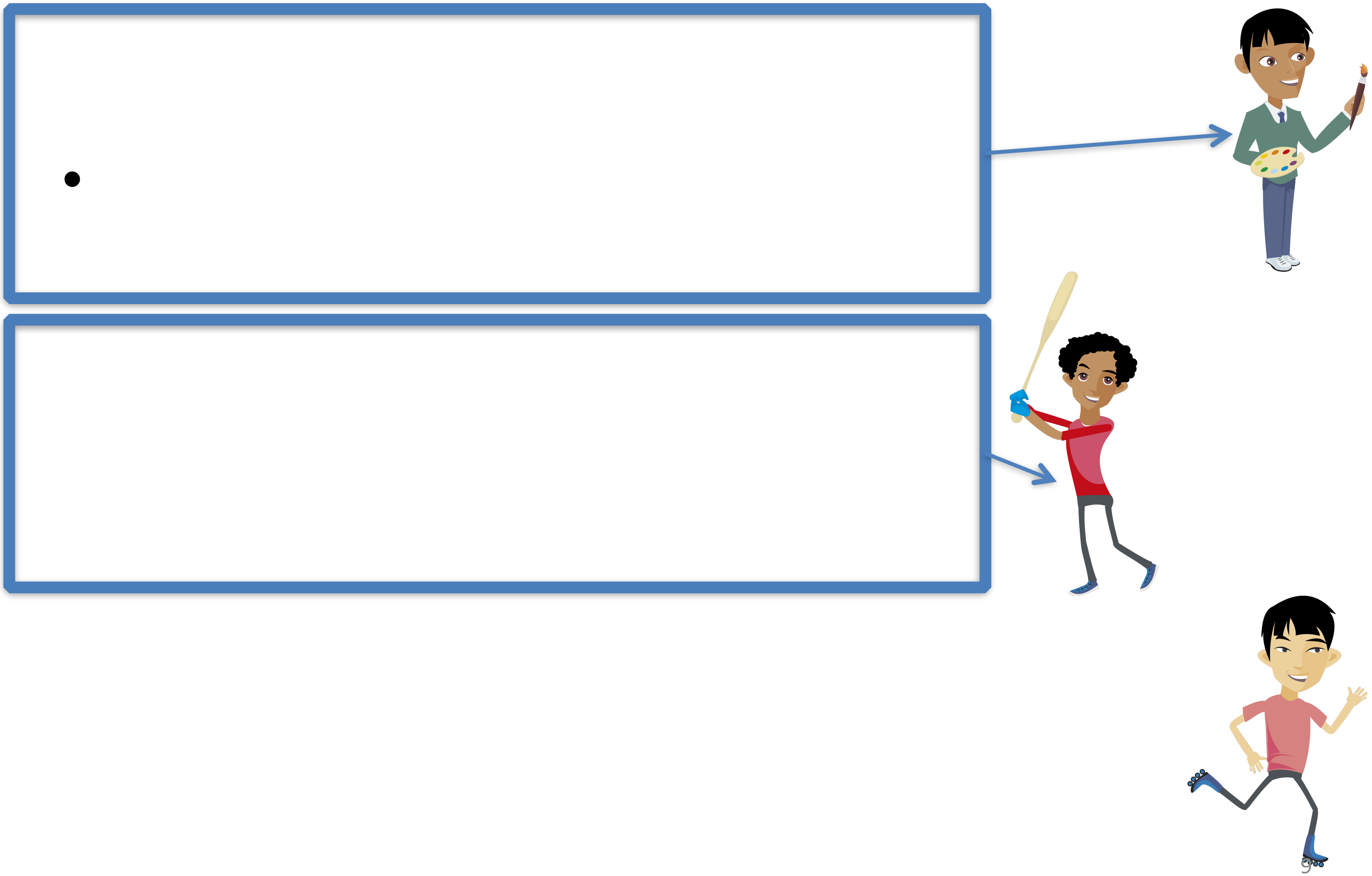
-

A small Java program

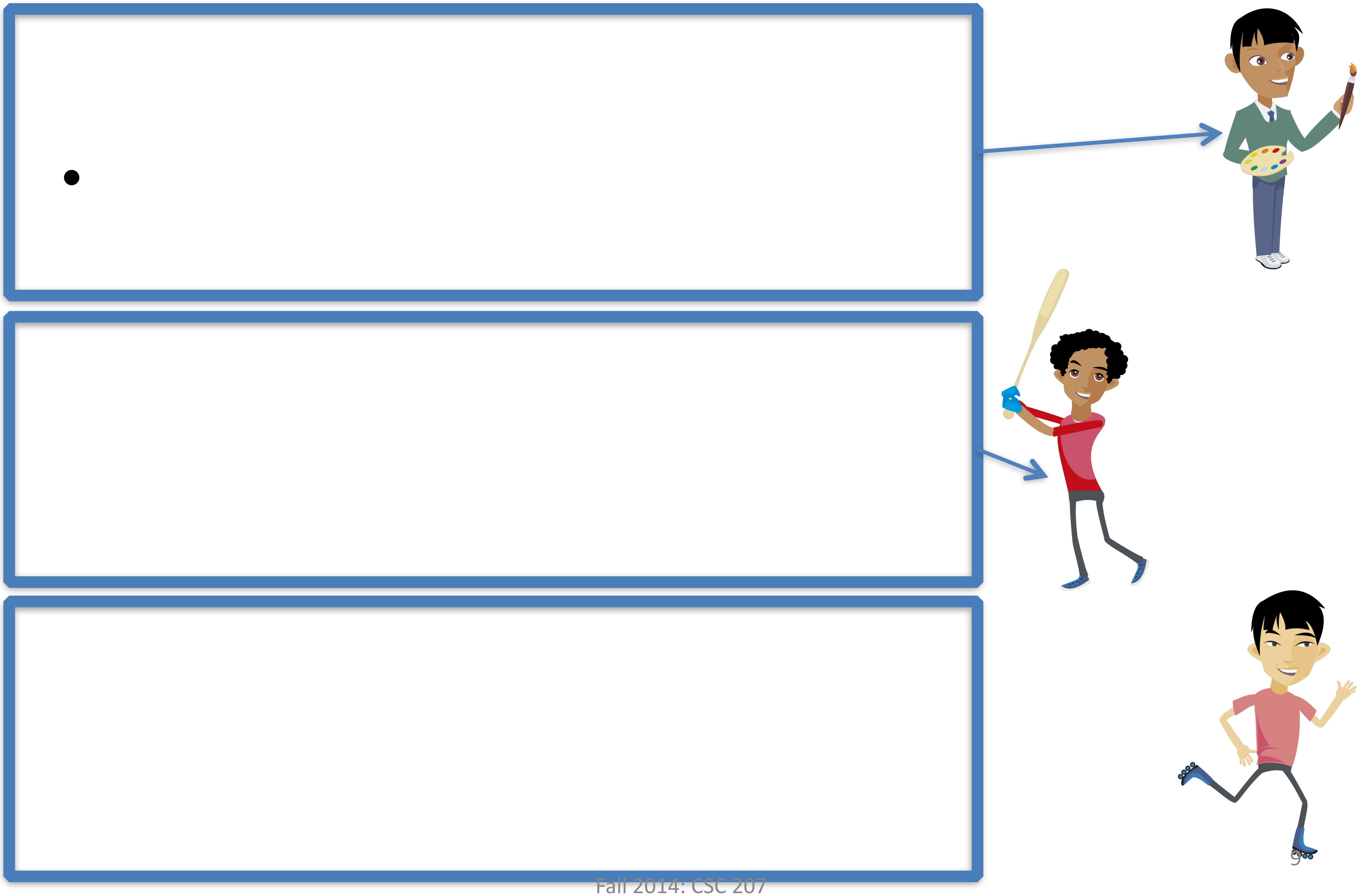
•

A small Java program

A small Java program

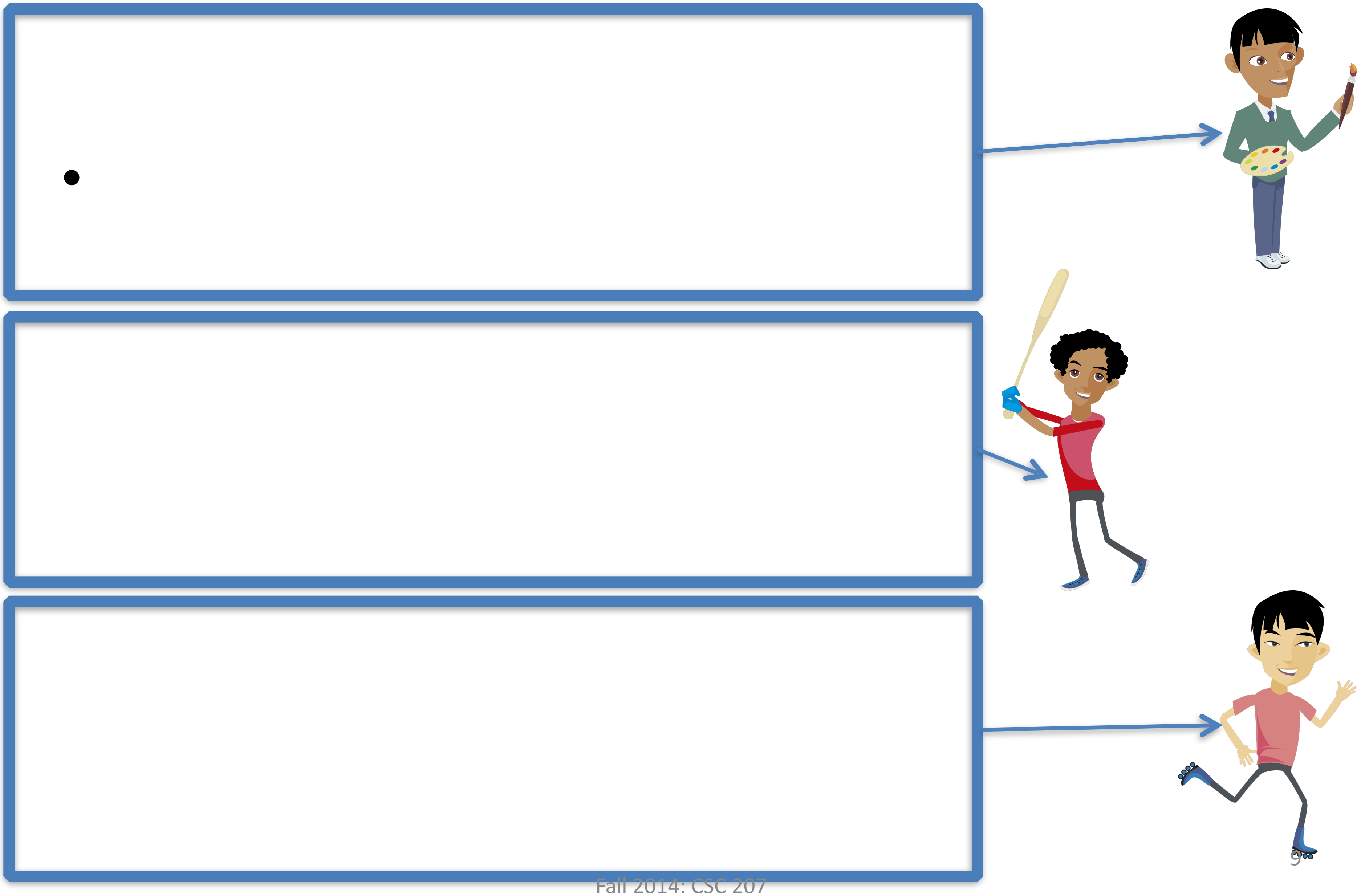


A small Java program

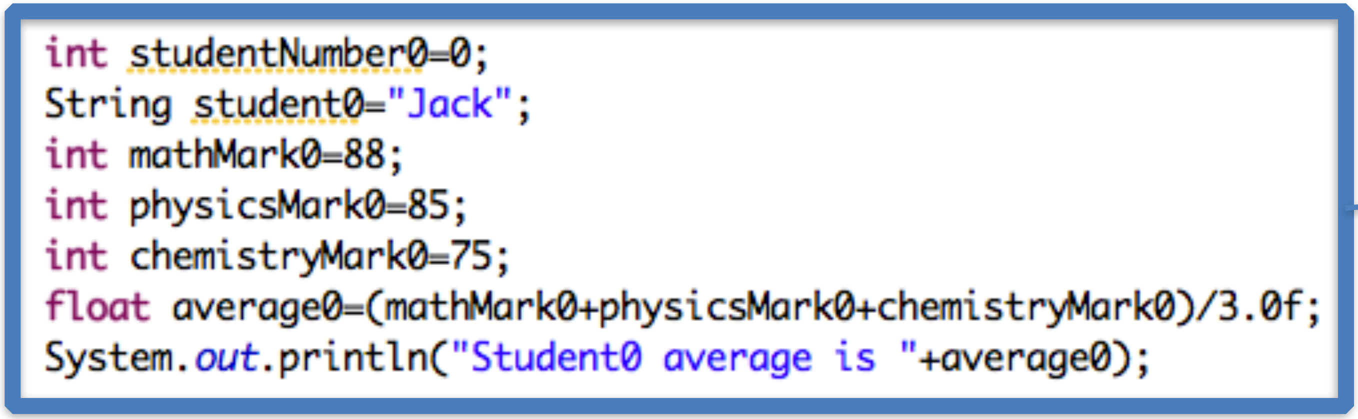


-

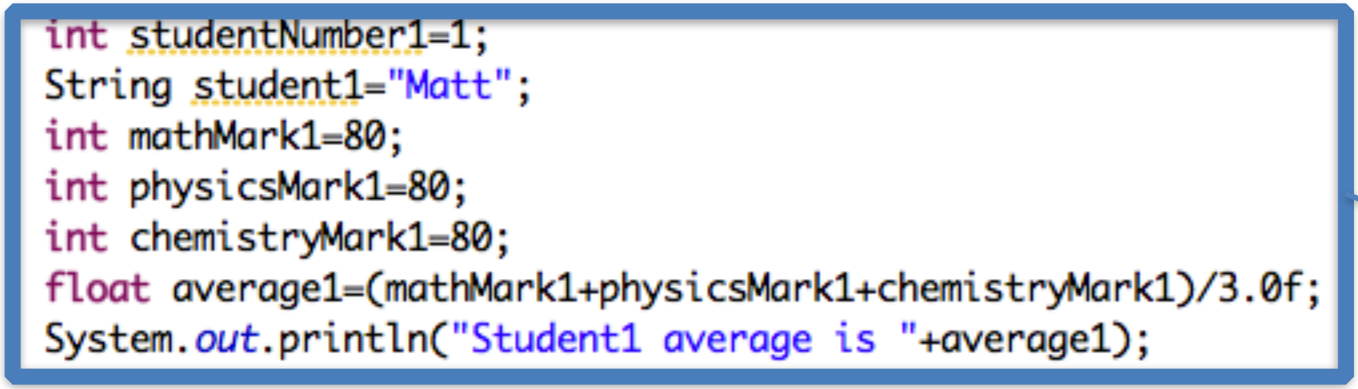
A small Java program



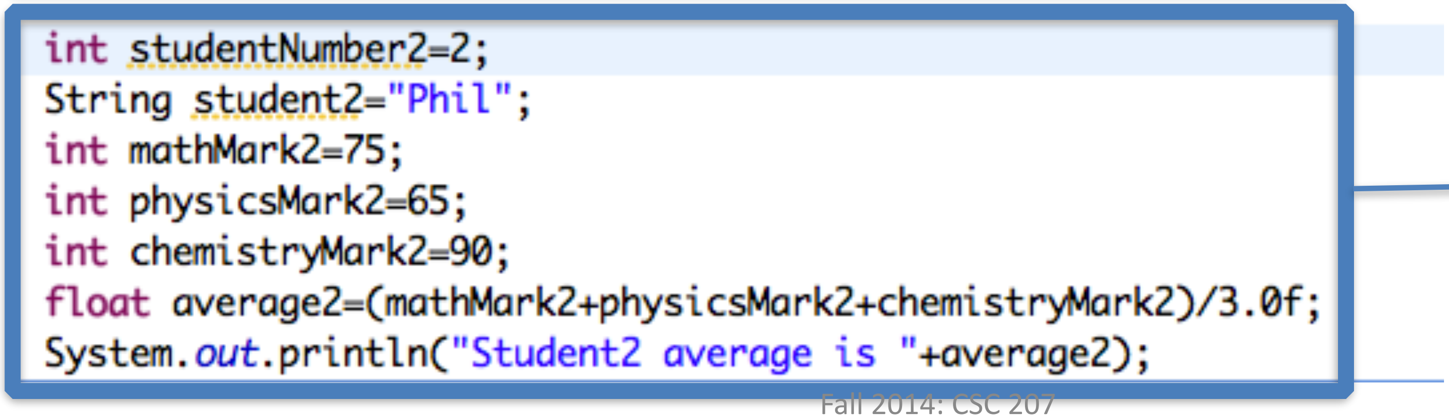
A small Java program



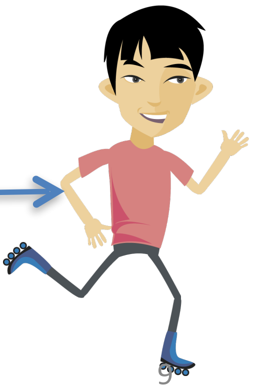
```java
int studentNumber0=0;
String student0="Jack";
int mathMark0=88;
int physicsMark0=85;
int chemistryMark0=75;
float average0=(mathMark0+physicsMark0+chemistryMark0)/3.0f;
System.out.println("Student0 average is "+average0);
```



```java
int studentNumber1=1;
String student1="Matt";
int mathMark1=80;
int physicsMark1=80;
int chemistryMark1=80;
float average1=(mathMark1+physicsMark1+chemistryMark1)/3.0f;
System.out.println("Student1 average is "+average1);
```



```java
int studentNumber2=2;
String student2="Phil";
int mathMark2=75;
int physicsMark2=65;
int chemistryMark2=90;
float average2=(mathMark2+physicsMark2+chemistryMark2)/3.0f;
System.out.println("Student2 average is "+average2);
```



Some problems with the previous solution.

Some problems with the previous solution.

- What happens when the number of students increase from **3 to 100**?
- What happens when the number of courses per students increases from **3 to 10**?
- How will your current solution **scale up**?
- Lot of **code being duplicated** (at present) across 3 students?
- Why is **duplicated code bad**?

•

•

Classes and Objects

- **<u>Classes</u>:**

1) We like to create our own custom ‘data types’.
2) Is a blue print or prototype from which Objects are created.
3) It models the state and behaviour of a real world object.

- **<u>Objects</u>:**

1) An object is created out of a blue print (classes see above).
2) An object is a software bundle of related state and behavior.

What does the blue print contain for a student?

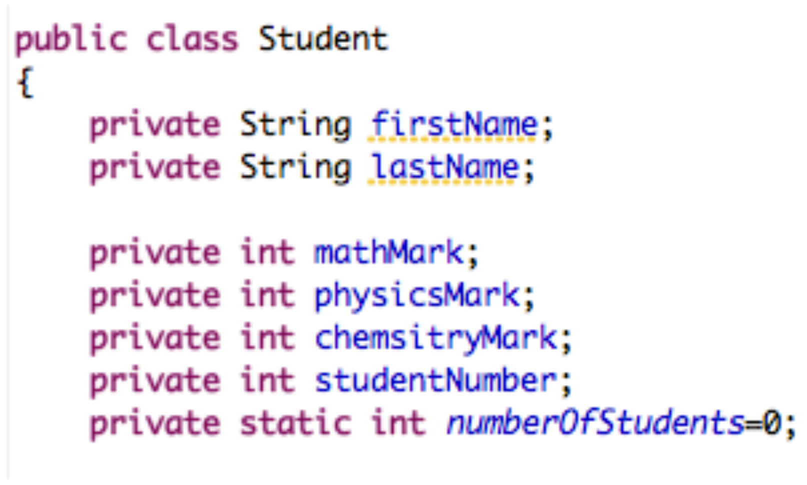
- The blue print for a student should contain atleast the following:

1) First Name of a student
2) Last Name of a student
3) Math mark
4) Physics mark
5) Chemistry mark

- Of course you can add more to this list, but lets assume we need the above five to be present in a student's blue print.

Lets create a **blue print** for a **student**.

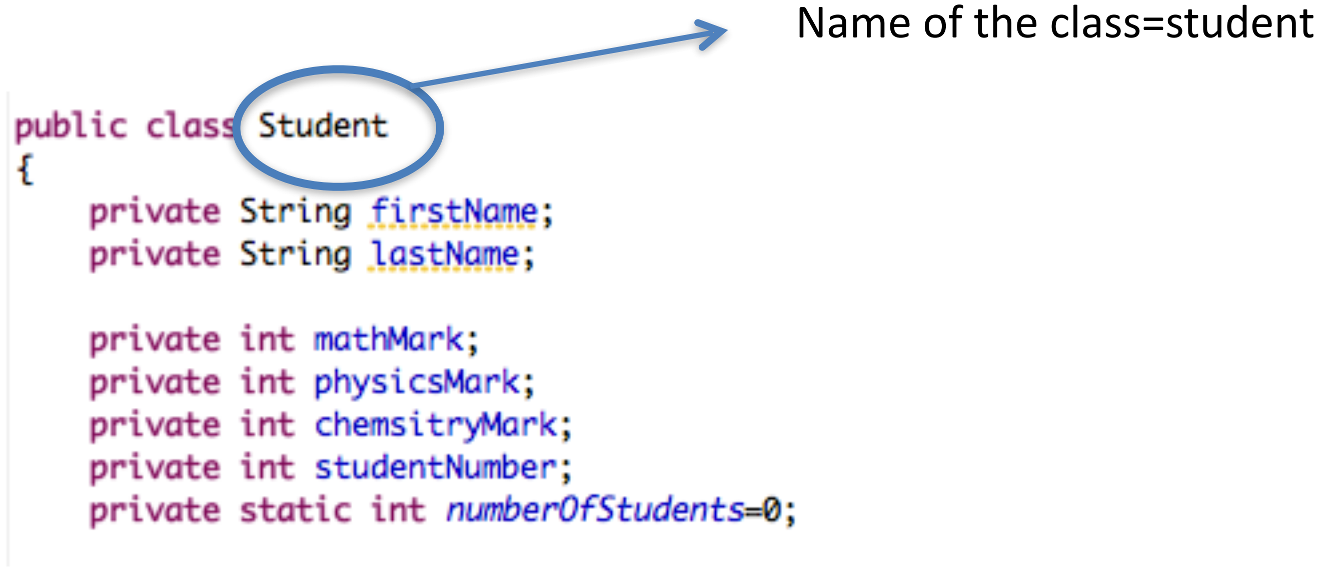
•



```java
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;
```

Lets create a **blue print** for a **student**.

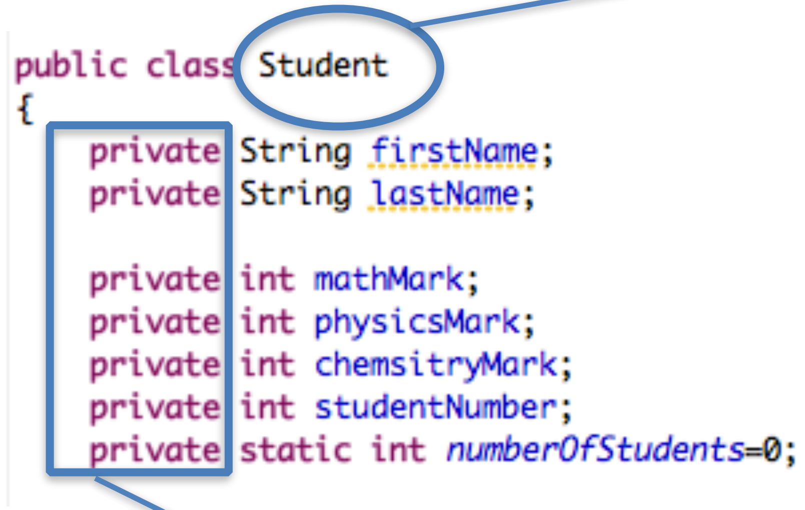
-



Name of the class=student

```
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;
```

Lets create a **blue print** for a **student**.

- 

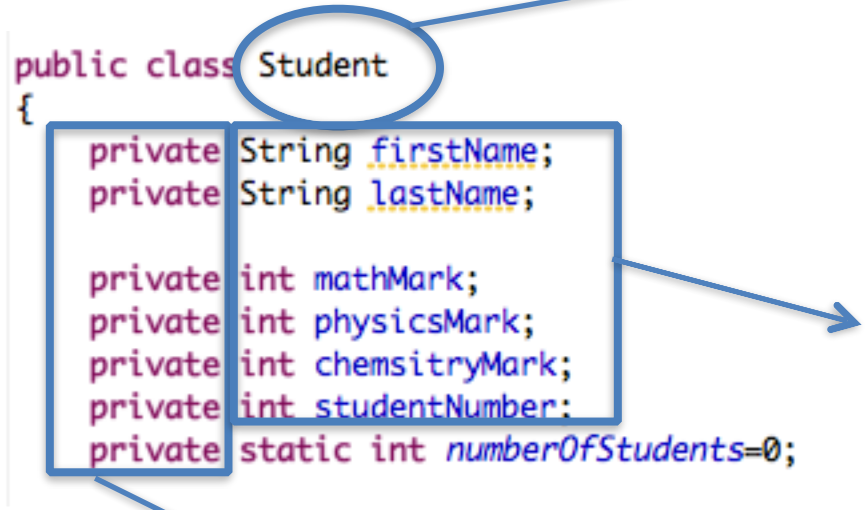
```
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;
```

Name of the class=student

Access modifier: Could be **public, protected** or **private**. We will cover them in the next hour.

Lets create a **blue print** for a **student**.

•



```
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;
```

Name of the class=student

Instance variables, we have 5 of them.

Access modifier: Could be **public, protected** or **private**. We will cover them in the next hour.

What do we do with the blue print?

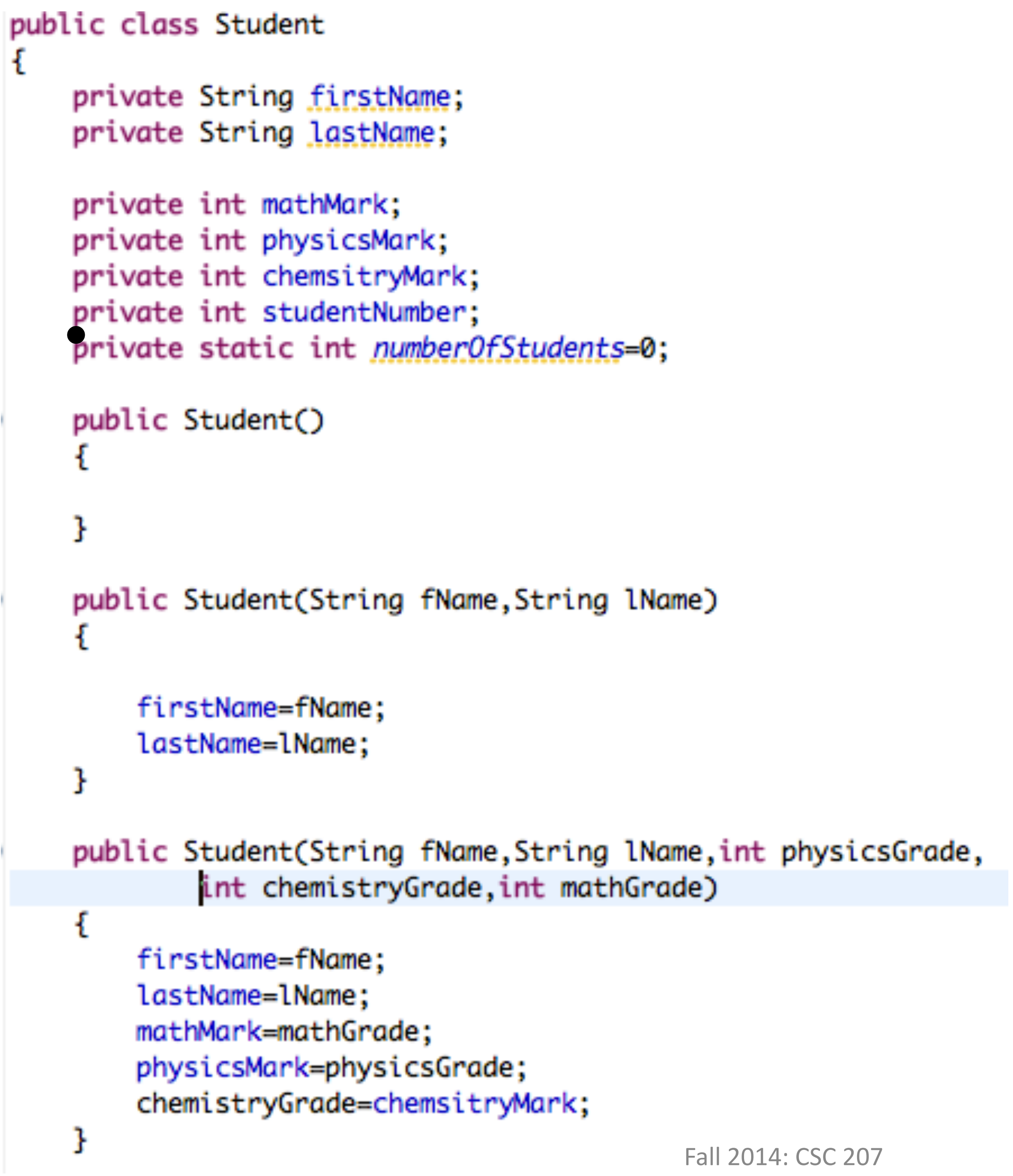
- Well, you like to create **objects** out of the blue print.
- More specifically you like to create student objects out of the student blue print.
- Before creating student objects, lets revisit the blue print and add a constructor.

What is a constructor?

A class contains **constructors** that are invoked to **create objects** from the class blueprint.

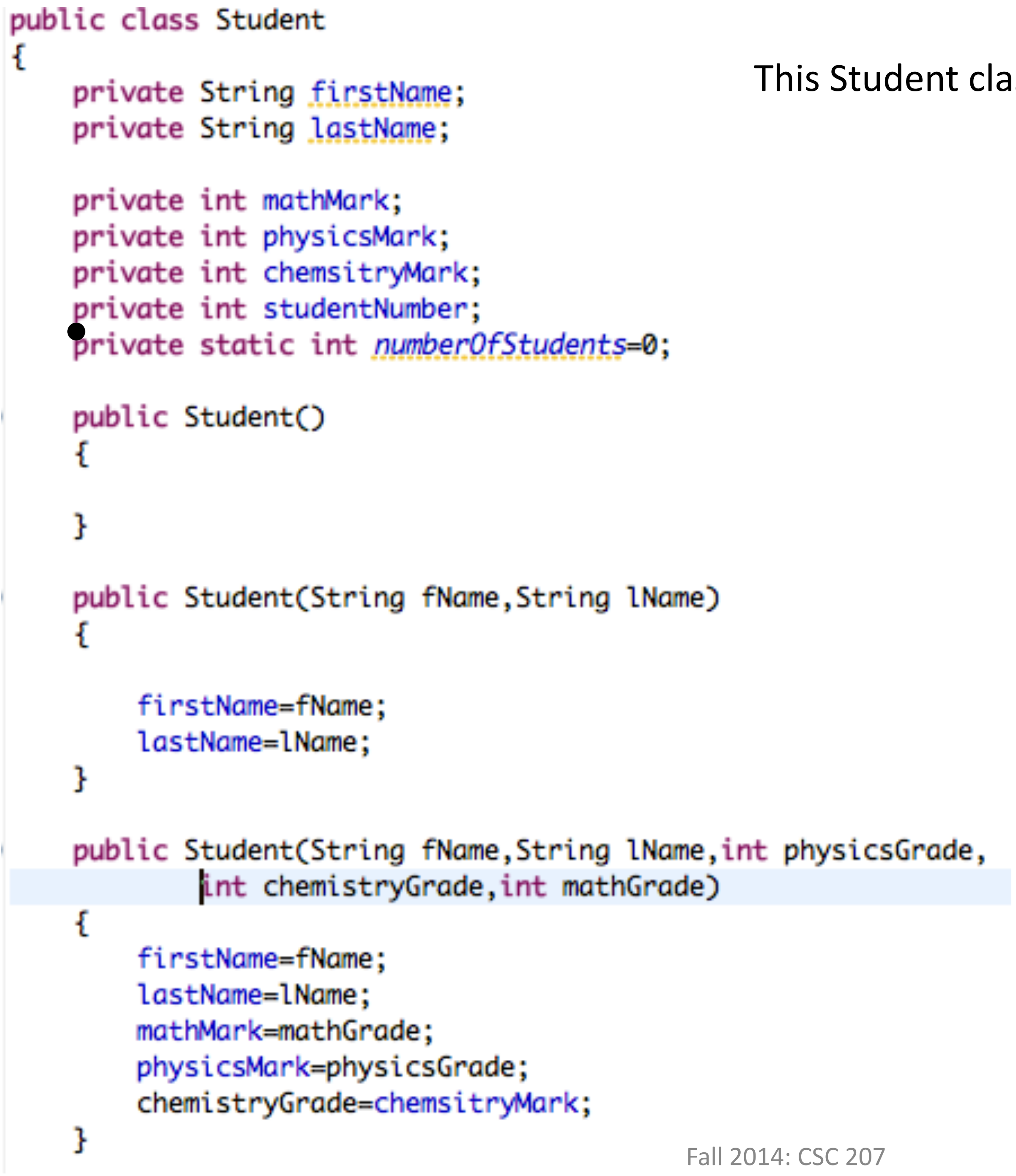
Constructor declarations look like method declarations—except that they use the **name of the class** and **have no return type**.

You don't have to provide any constructors for your class, but you must be careful when doing this. The compiler automatically provides a no-argument, default constructor for any class without constructors.



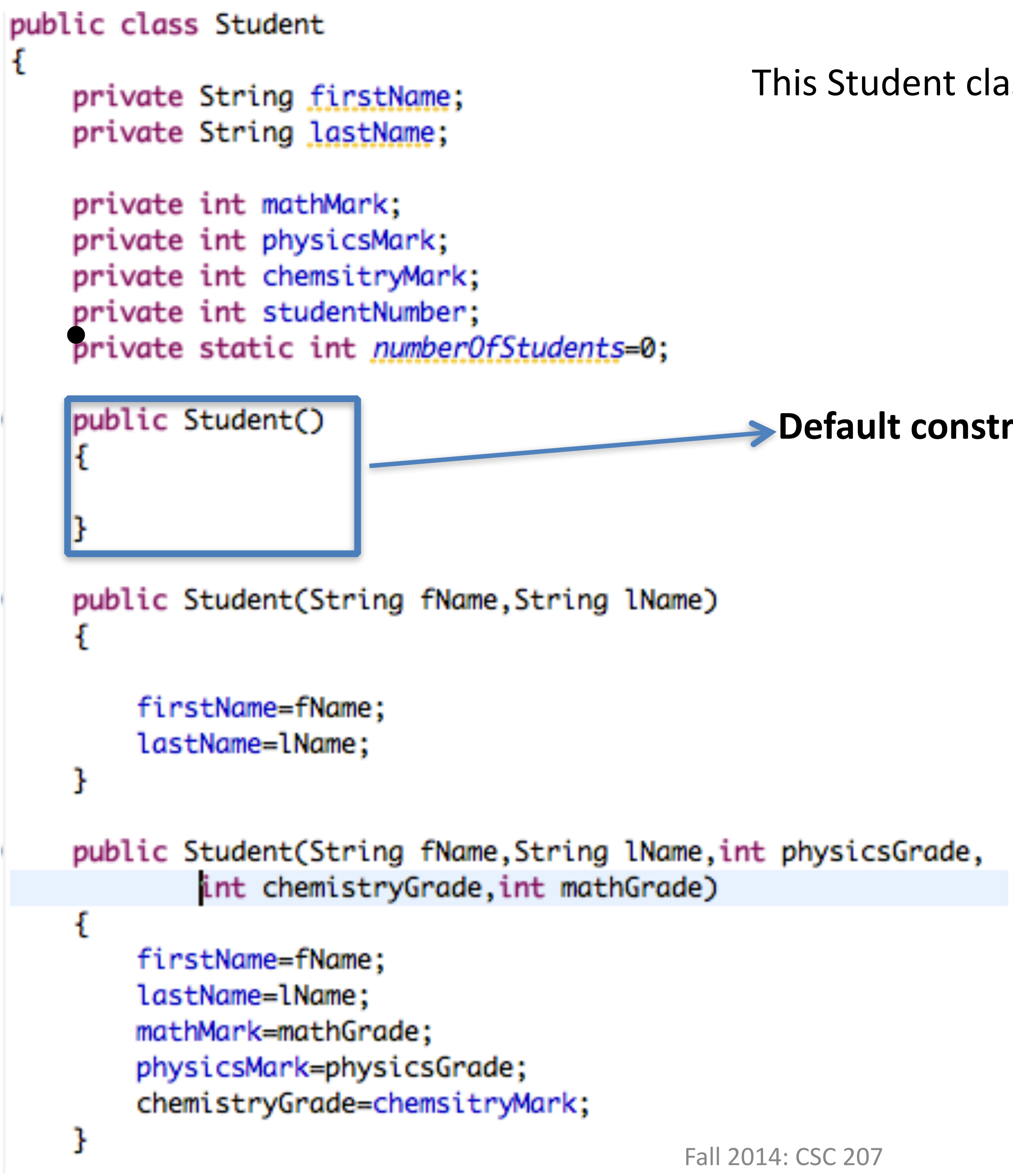
```java
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;

    public Student()
    {

    }

    public Student(String fName,String lName)
    {

        firstName=fName;
        lastName=lName;
    }

    public Student(String fName,String lName,int physicsGrade,
            int chemistryGrade,int mathGrade)
    {
        firstName=fName;
        lastName=lName;
        mathMark=mathGrade;
        physicsMark=physicsGrade;
        chemistryGrade=chemsitryMark;
    }
```

This Student class has **three** constructors



```
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;

    public Student()
    {

    }

    public Student(String fName,String lName)
    {

        firstName=fName;
        lastName=lName;
    }

    public Student(String fName,String lName,int physicsGrade,
            int chemistryGrade,int mathGrade)
    {
        firstName=fName;
        lastName=lName;
        mathMark=mathGrade;
        physicsMark=physicsGrade;
        chemistryGrade=chemsitryMark;
    }
```

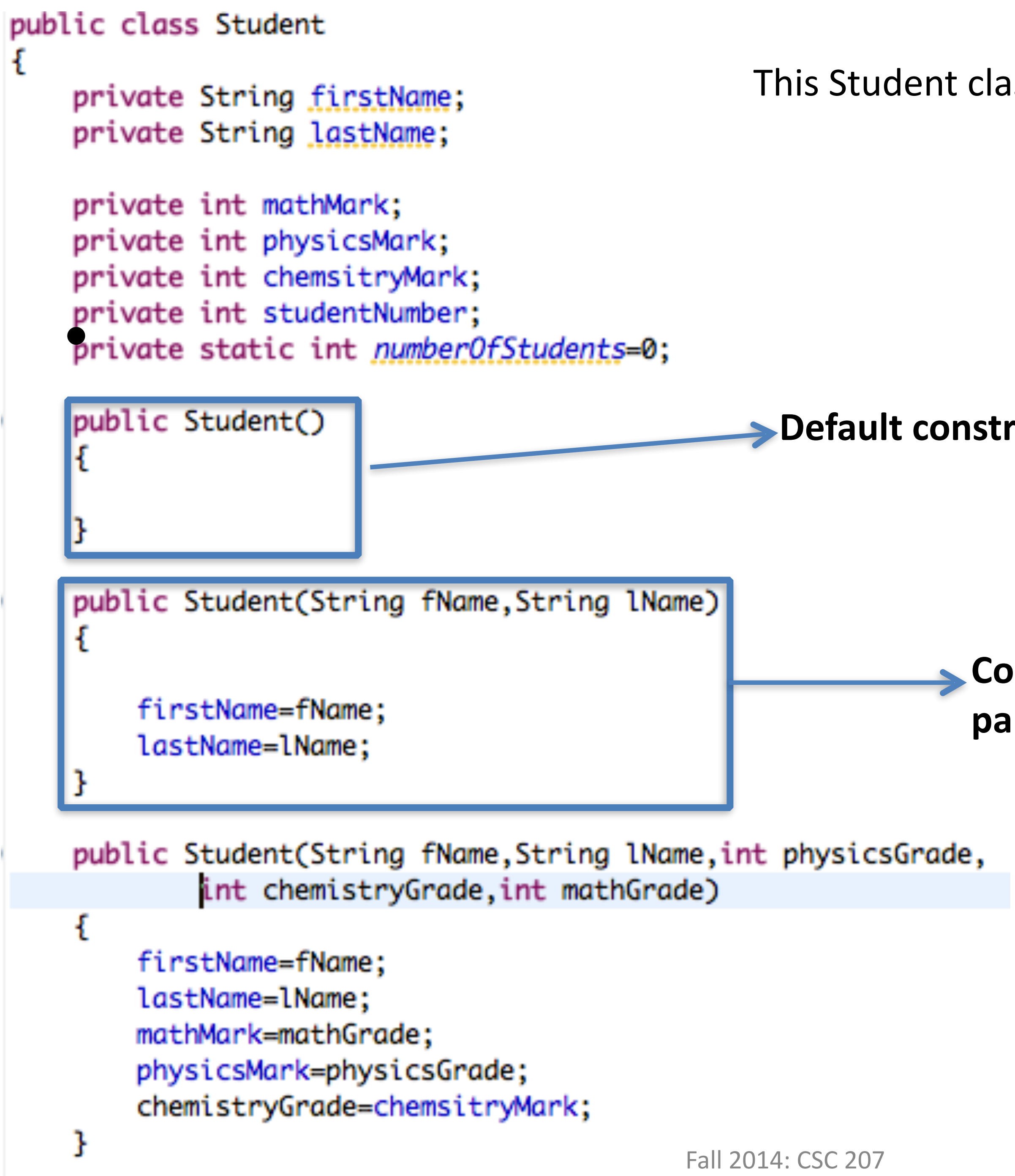
This Student class has **three** constructors



```
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;

    public Student()
    {

    }

    public Student(String fName,String lName)
    {

        firstName=fName;
        lastName=lName;
    }

    public Student(String fName,String lName,int physicsGrade,
            int chemistryGrade,int mathGrade)
    {
        firstName=fName;
        lastName=lName;
        mathMark=mathGrade;
        physicsMark=physicsGrade;
        chemistryGrade=chemsitryMark;
    }
```

Default constructor.

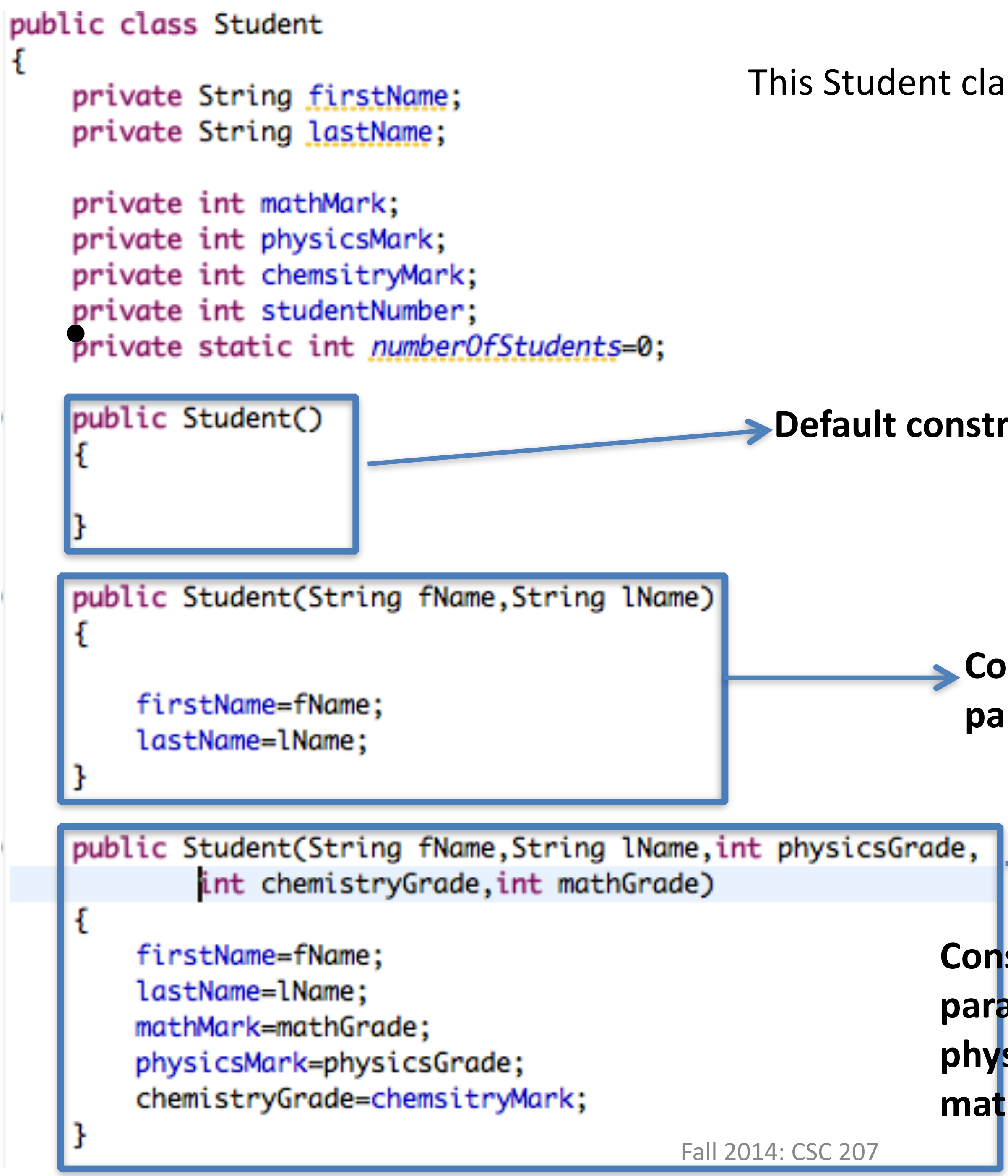
This Student class has **three** constructors



```java
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;

    public Student()
    {

    }

    public Student(String fName,String lName)
    {

        firstName=fName;
        lastName=lName;
    }

    public Student(String fName,String lName,int physicsGrade,
            int chemistryGrade,int mathGrade)
    {
        firstName=fName;
        lastName=lName;
        mathMark=mathGrade;
        physicsMark=physicsGrade;
        chemistryGrade=chemsitryMark;
    }
```

Default constructor.

Constructor that takes in two input parameters i.e. fName and lName



```java
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;

    public Student()
    {

    }

    public Student(String fName,String lName)
    {

        firstName=fName;
        lastName=lName;
    }

    public Student(String fName,String lName,int physicsGrade,
            int chemistryGrade,int mathGrade)
    {
        firstName=fName;
        lastName=lName;
        mathMark=mathGrade;
        physicsMark=physicsGrade;
        chemistryGrade=chemsitryMark;
    }
```

This Student class has **three** constructors

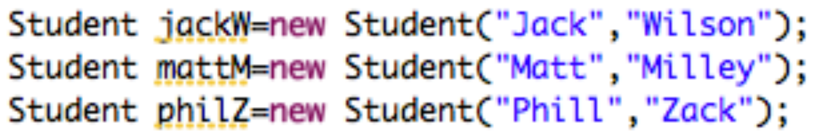
Default constructor.

Constructor that takes in two input parameters i.e. fName and lName

Constructor that takes in five input parameters i.e. fName, lName, physicsGrade,chemistryGrade and mathGrade

Lets create our three Students from this blue print.

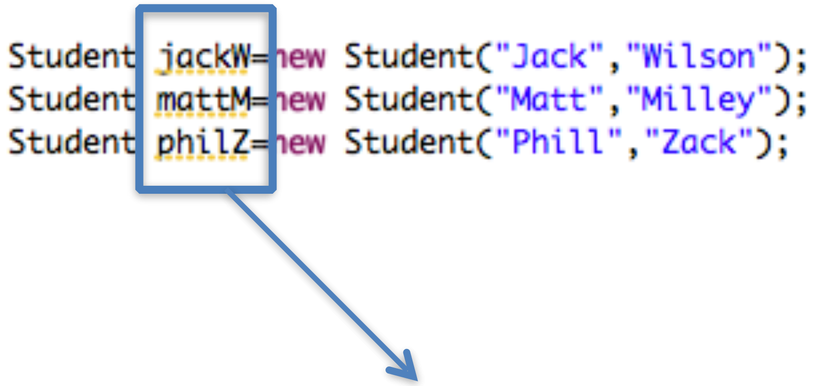
•



```
Student jackW=new Student("Jack","Wilson");
Student mattM=new Student("Matt","Milley");
Student philZ=new Student("Phill","Zack");
```

Lets create our three Students from this blue print.

-

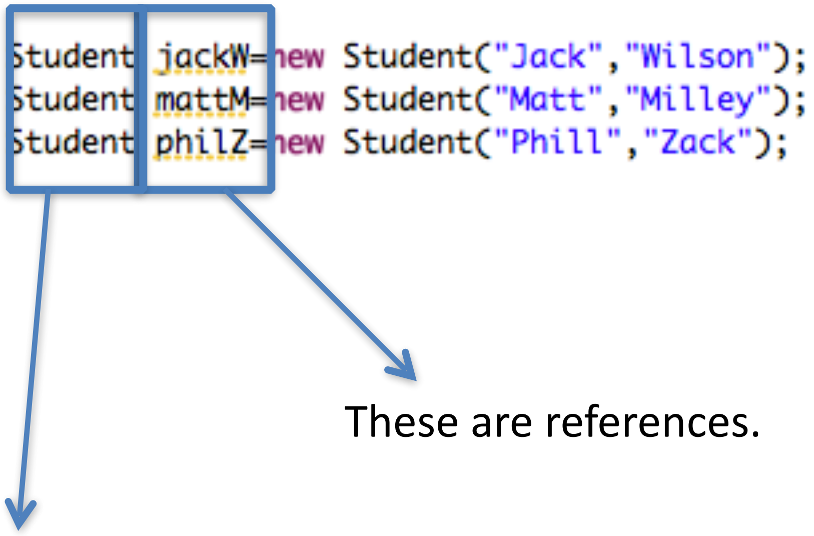


```
Student jackW=new Student("Jack","Wilson");
Student mattM=new Student("Matt","Milley");
Student philZ=new Student("Phill","Zack");
```

These are references.

Lets create our three Students from this blue print.

•



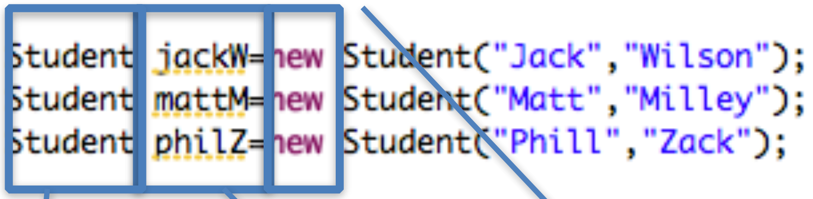
```
Student jackW=new Student("Jack","Wilson");
Student mattM=new Student("Matt","Milley");
Student philZ=new Student("Phill","Zack");
```

These are references.

The **type** of the references are **Student**

Lets create our three Students from this blue print.

•



```
Student jackW=new Student("Jack","Wilson");
Student mattM=new Student("Matt","Milley");
Student philZ=new Student("Phill","Zack");
```

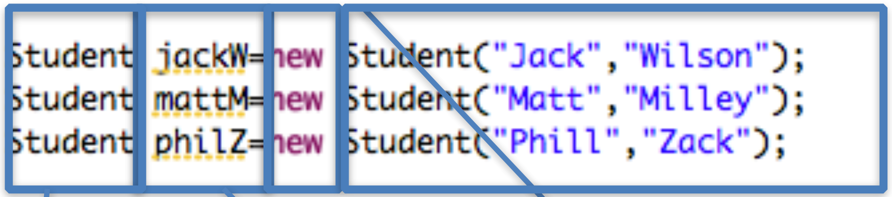
New indicates the creation of object followed by the name of the constructor

These are references.

The **type** of the references are **Student**

Lets create our three Students from this blue print.

•



```
Student jackW=new Student("Jack","Wilson");
Student mattM=new Student("Matt","Milley");
Student philZ=new Student("Phill","Zack");
```

Call the constructor that takes in **two String** as input

New indicates the creation of object followed by the name of the constructor

These are references.

The **type** of the references are **Student**

- So we have three student objects created with their `firstName` and `lastName` instance variable set.

- We still need to set their:

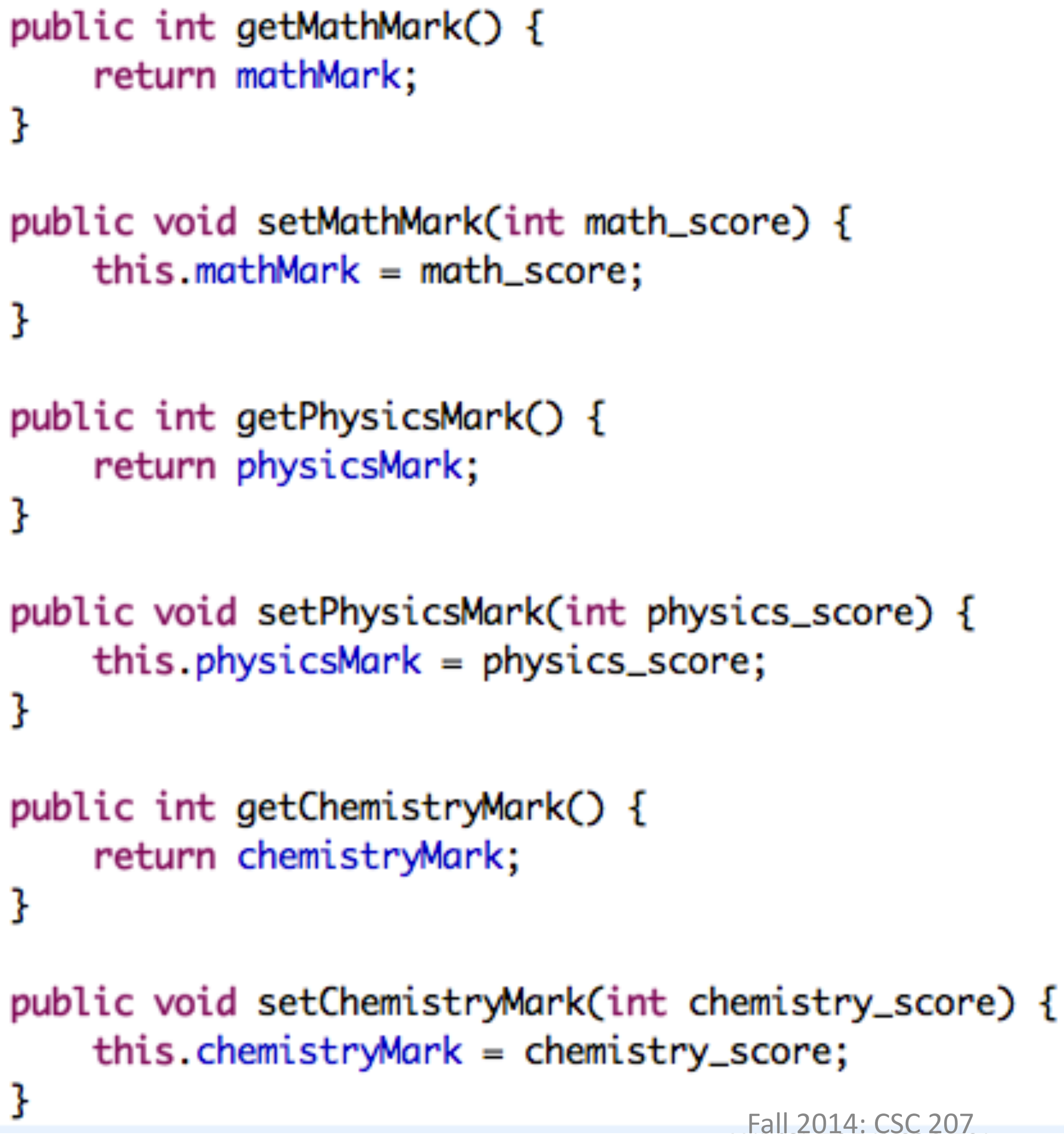
1) Math (`mathMark`)
2) Physics (`physicsMark`)
3) Chemistry(`chemistryMark`)

Lets add *setters* and *getters*.

- Setters are also called ***mutator method*** and is a method used to control changes to a variable.

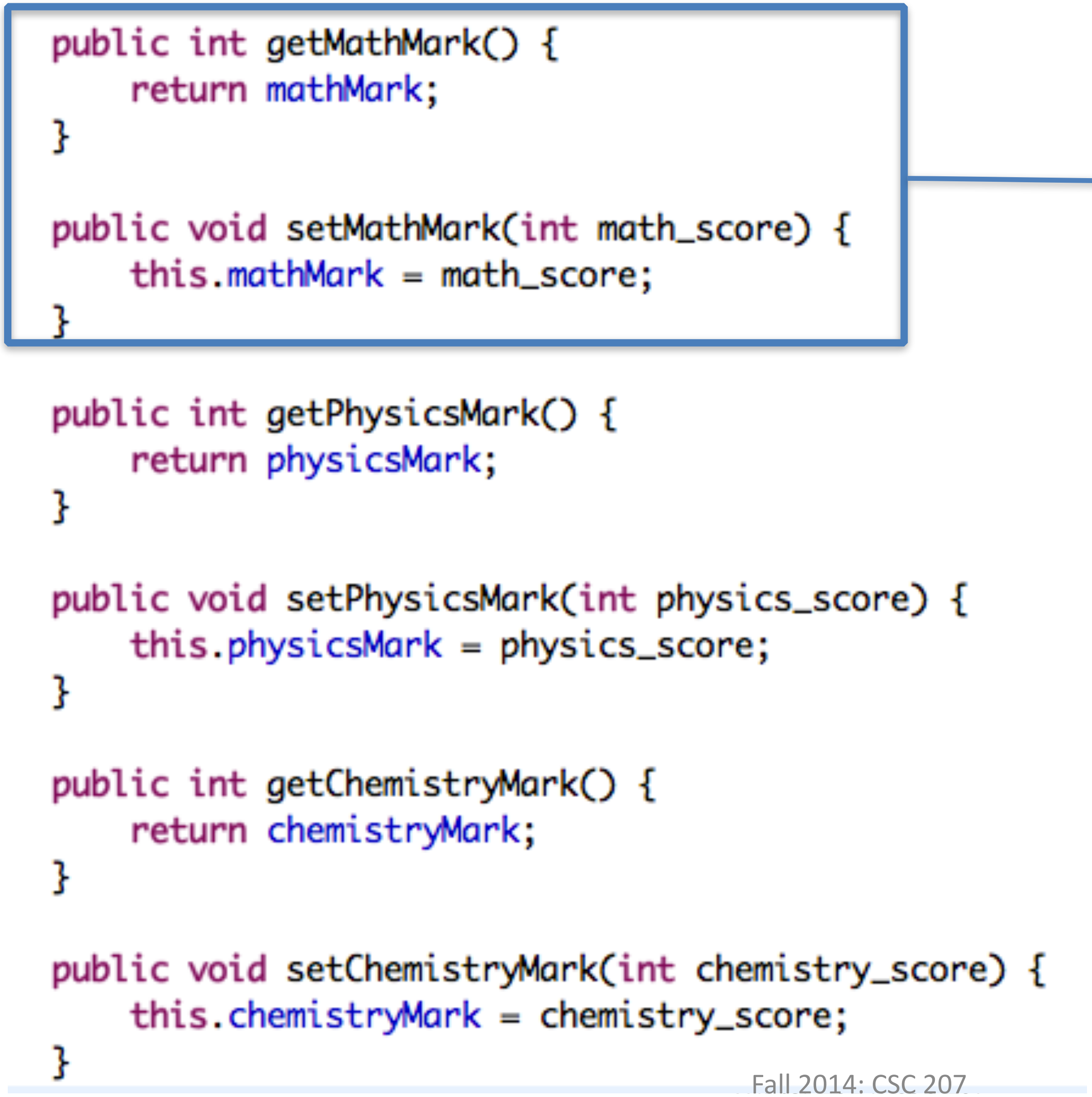
- Getters are also called ***accessor method*** that returns the value of the private instance variable.

Inside the Student class, add the following



```
public int getMathMark() {
    return mathMark;
}

public void setMathMark(int math_score) {
    this.mathMark = math_score;
}

public int getPhysicsMark() {
    return physicsMark;
}

public void setPhysicsMark(int physics_score) {
    this.physicsMark = physics_score;
}

public int getChemistryMark() {
    return chemistryMark;
}

public void setChemistryMark(int chemistry_score) {
    this.chemistryMark = chemistry_score;
}
```

Inside the Student class, add the following



```
public int getMathMark() {
    return mathMark;
}

public void setMathMark(int math_score) {
    this.mathMark = math_score;
}

public int getPhysicsMark() {
    return physicsMark;
}

public void setPhysicsMark(int physics_score) {
    this.physicsMark = physics_score;
}

public int getChemistryMark() {
    return chemistryMark;
}

public void setChemistryMark(int chemistry_score) {
    this.chemistryMark = chemistry_score;
}
```

Getter and Setter of the `mathMark` instance variable.

Inside the Student class, add the following

```
public int getMathMark() {
    return mathMark;
}

public void setMathMark(int math_score) {
    this.mathMark = math_score;
}
```

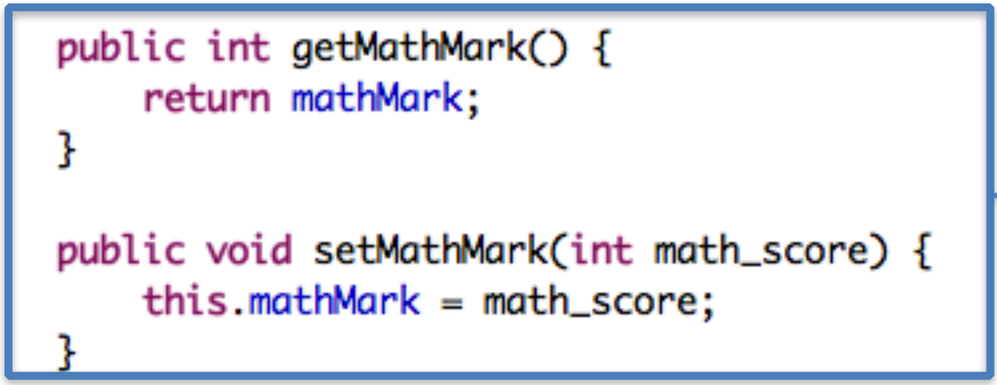
Getter and Setter of the `mathMark` instance variable.

```
public int getPhysicsMark() {
    return physicsMark;
}

public void setPhysicsMark(int physics_score) {
    this.physicsMark = physics_score;
}
```

Getter and Setter of the `physicsMark` instance variable.

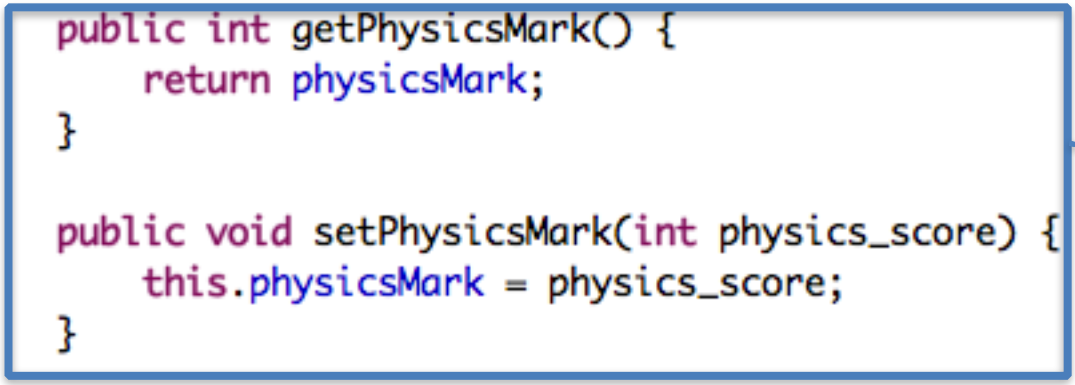
```
public int getChemistryMark() {
    return chemistryMark;
}

public void setChemistryMark(int chemistry_score) {
    this.chemistryMark = chemistry_score;
}
```

Inside the Student class, add the following



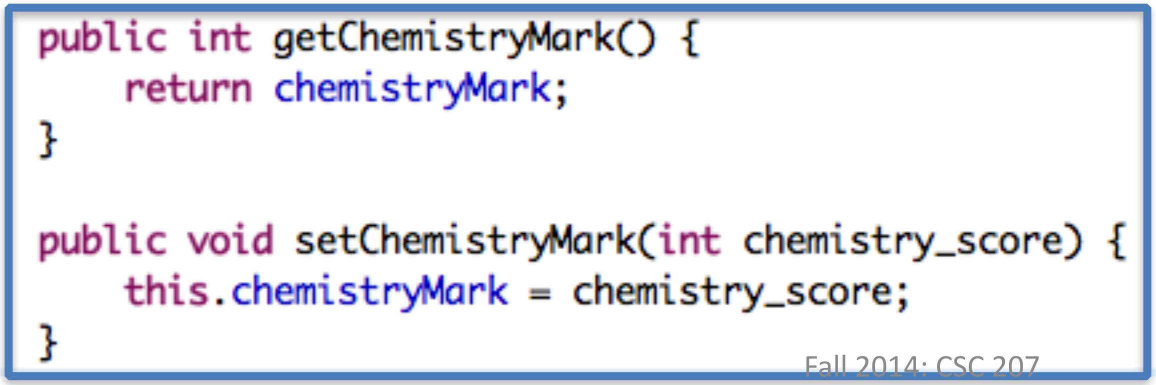
```java
public int getMathMark() {
    return mathMark;
}

public void setMathMark(int math_score) {
    this.mathMark = math_score;
}
```

Getter and Setter of the `mathMark` instance variable.



```java
public int getPhysicsMark() {
    return physicsMark;
}

public void setPhysicsMark(int physics_score) {
    this.physicsMark = physics_score;
}
```

Getter and Setter of the `physicsMark` instance variable.



```java
public int getChemistryMark() {
    return chemistryMark;
}

public void setChemistryMark(int chemistry_score) {
    this.chemistryMark = chemistry_score;
}
```

Getter and Setter of the `chemistryMark` instance variable.

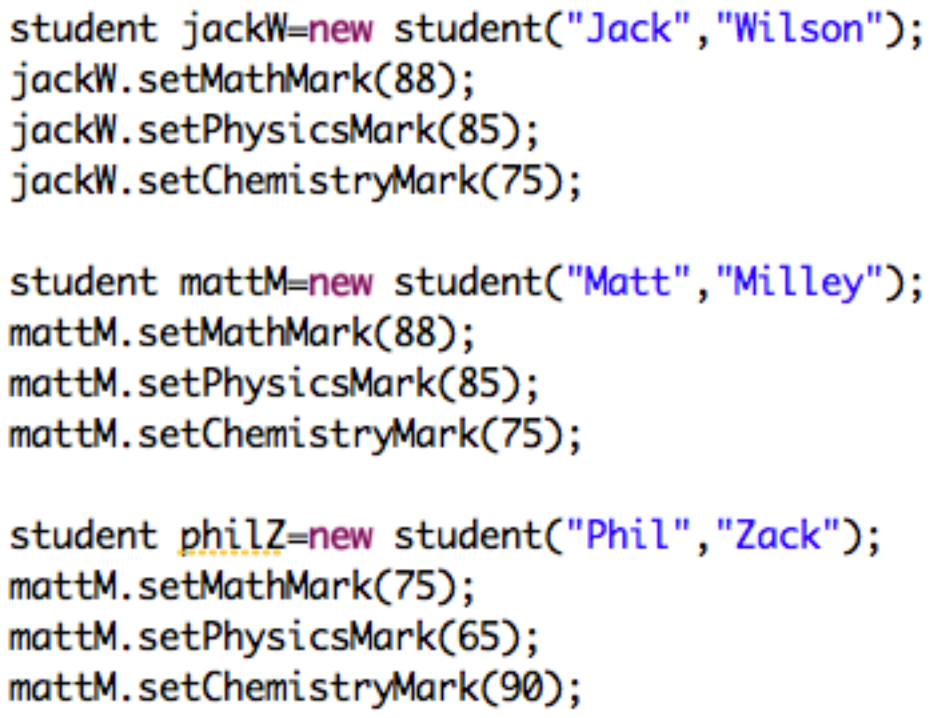
- You can name your getters and setters anything that you like.

- However it is a good practice to give them meaningful names so that it becomes easy to understand the purpose of such methods.

- These methods that we just added (getters and setters) are also called as ***instance methods***.

How can we use these getters and setters that we just created?

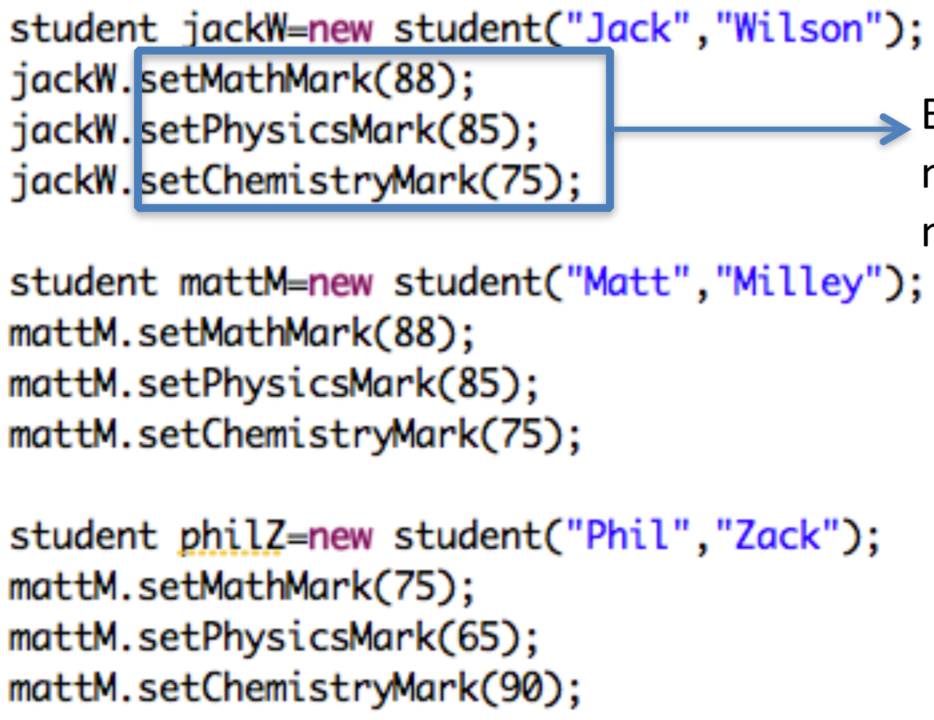
-



```
student jackW=new student("Jack","Wilson");
jackW.setMathMark(88);
jackW.setPhysicsMark(85);
jackW.setChemistryMark(75);

student mattM=new student("Matt","Milley");
mattM.setMathMark(88);
mattM.setPhysicsMark(85);
mattM.setChemistryMark(75);

student philZ=new student("Phil","Zack");
mattM.setMathMark(75);
mattM.setPhysicsMark(65);
mattM.setChemistryMark(90);
```

How can we use these getters and setters that we just created?

•

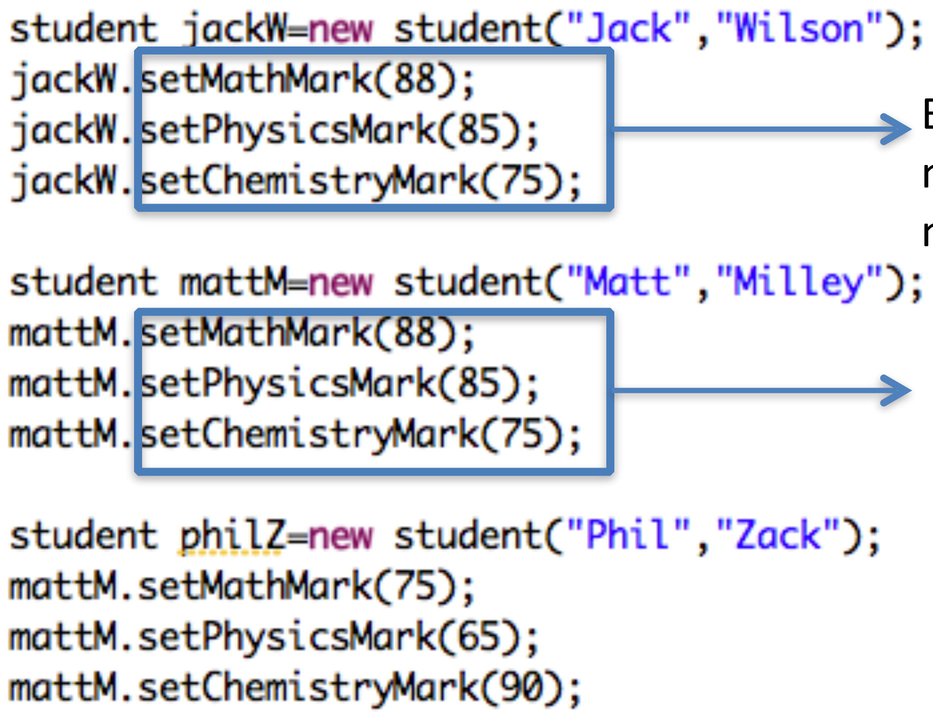


```
student jackW=new student("Jack","Wilson");
jackW.setMathMark(88);
jackW.setPhysicsMark(85);
jackW.setChemistryMark(75);

student mattM=new student("Matt","Milley");
mattM.setMathMark(88);
mattM.setPhysicsMark(85);
mattM.setChemistryMark(75);

student philZ=new student("Phil","Zack");
mattM.setMathMark(75);
mattM.setPhysicsMark(65);
mattM.setChemistryMark(90);
```

By using the setter, I am adding the marks for Jack Wilson. Inside the setter methods **this** refers to jackW.

How can we use these getters and setters that we just created?

•



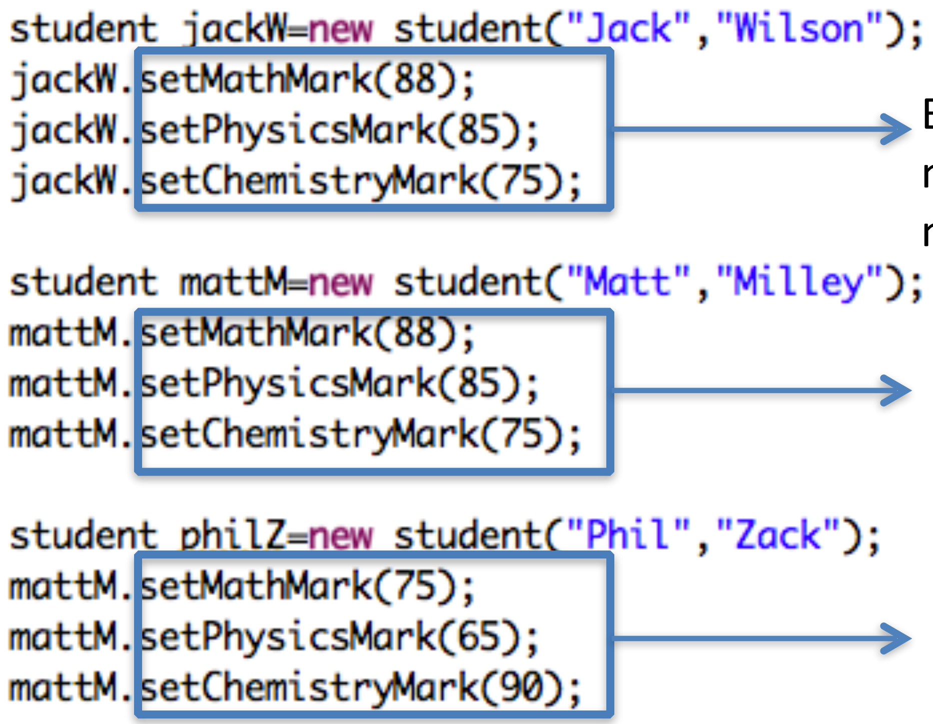
```
student jackW=new student("Jack","Wilson");
jackW.setMathMark(88);
jackW.setPhysicsMark(85);
jackW.setChemistryMark(75);

student mattM=new student("Matt","Milley");
mattM.setMathMark(88);
mattM.setPhysicsMark(85);
mattM.setChemistryMark(75);

student philZ=new student("Phil","Zack");
mattM.setMathMark(75);
mattM.setPhysicsMark(65);
mattM.setChemistryMark(90);
```

By using the setter, I am adding the marks for Jack Wilson. Inside the setter methods **this** refers to jackW.

By using the setter, I am adding the marks for Matt Milley. Inside the setter methods **this** refers to mattM

How can we use these getters and setters that we just created?

•



```
student jackW=new student("Jack","Wilson");
jackW.setMathMark(88);
jackW.setPhysicsMark(85);
jackW.setChemistryMark(75);

student mattM=new student("Matt","Milley");
mattM.setMathMark(88);
mattM.setPhysicsMark(85);
mattM.setChemistryMark(75);

student philZ=new student("Phil","Zack");
mattM.setMathMark(75);
mattM.setPhysicsMark(65);
mattM.setChemistryMark(90);
```

By using the setter, I am adding the marks for Jack Wilson. Inside the setter methods **this** refers to jackW.

By using the setter, I am adding the marks for Matt Milley. Inside the setter methods **this** refers to mattM

By using the setter, I am adding the marks for Phil Zack. Inside the setter methods **this** refers to philZ

- You can also add getters/setters for the **fName** and **lName** instance variables in your student class.

- What kind of student object gets created if I create it using the **first constructor** in the blue print i.e. `Student()` ?

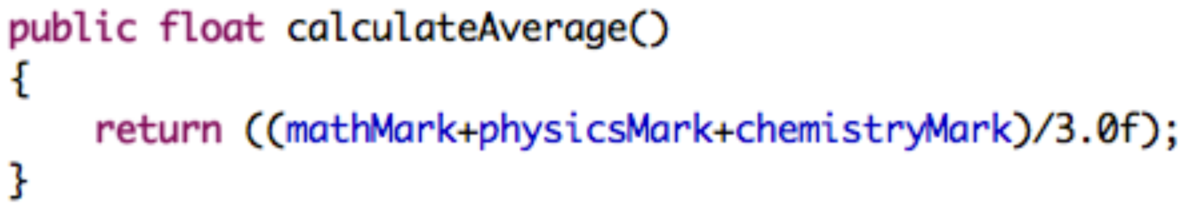
Answer: It is a student object which does not have a fName, or a lName, or a mathMark or a physicsMark or a chemistryMark.

- What kind of student object gets created if I create it using the **second constructor** in the blue print i.e. `Student(String f_name, String l_name)`

Answer: It is a student object which has only the fName and the lName set.

Lets add an instance method that calculates the average in the student class.

•

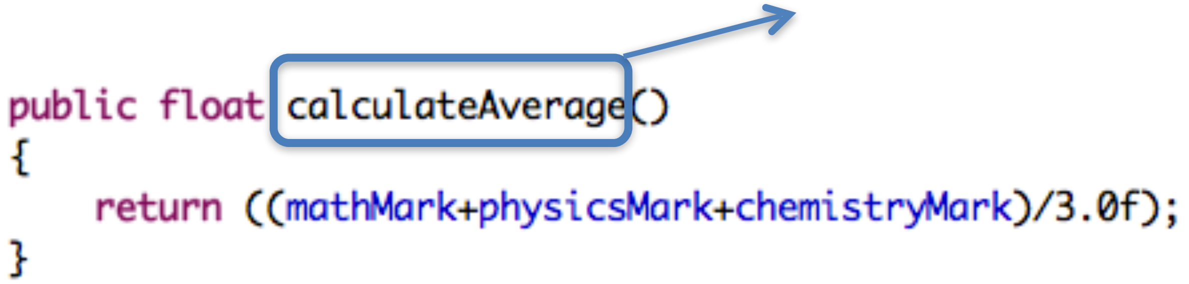


```
public float calculateAverage()
{
    return ((mathMark+physicsMark+chemistryMark)/3.0f);
}
```

Lets add an instance method that calculates the average in the student class.

•

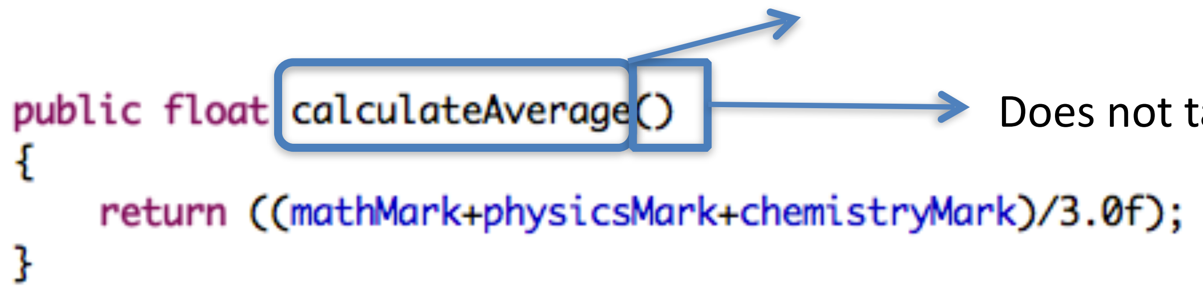
This is the name of the function



```
public float calculateAverage()
{
    return ((mathMark+physicsMark+chemistryMark)/3.0f);
}
```

Lets add an instance method that calculates the average in the student class.

•

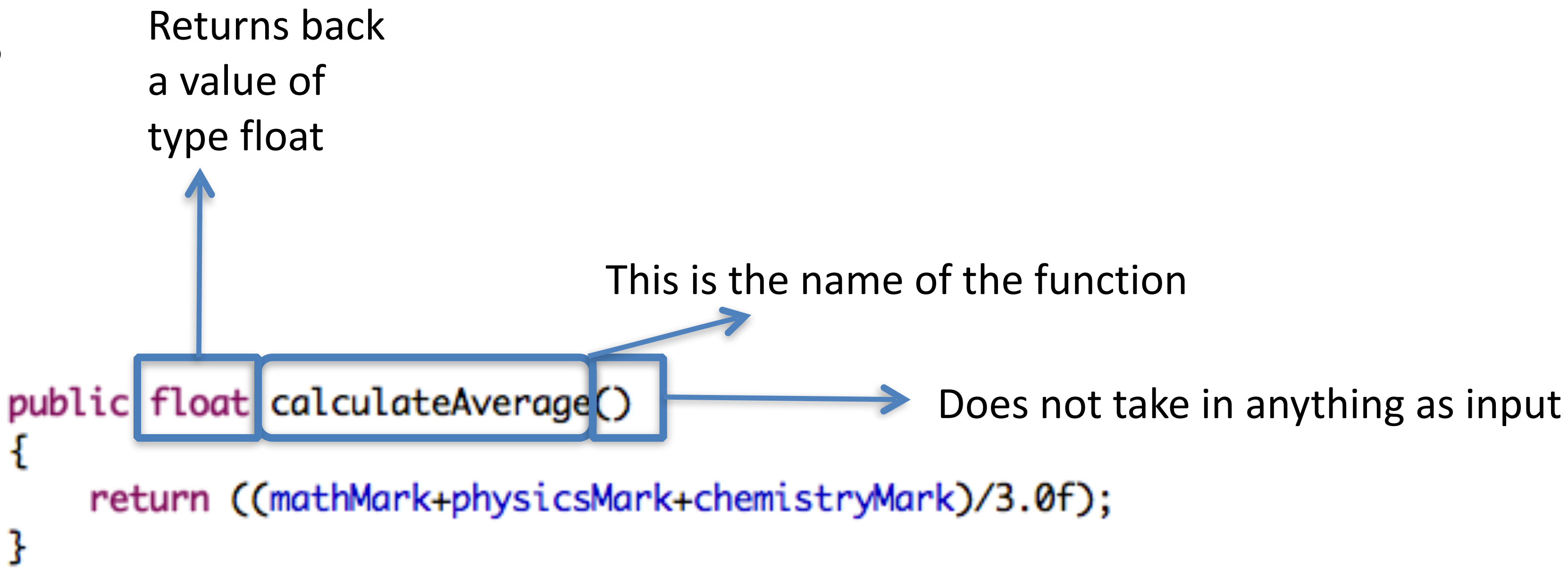


This is the name of the function

```
public float calculateAverage()
{
    return ((mathMark+physicsMark+chemistryMark)/3.0f);
}
```

Does not take in anything as input

Lets add an instance method that calculates the average in the student class.

- 

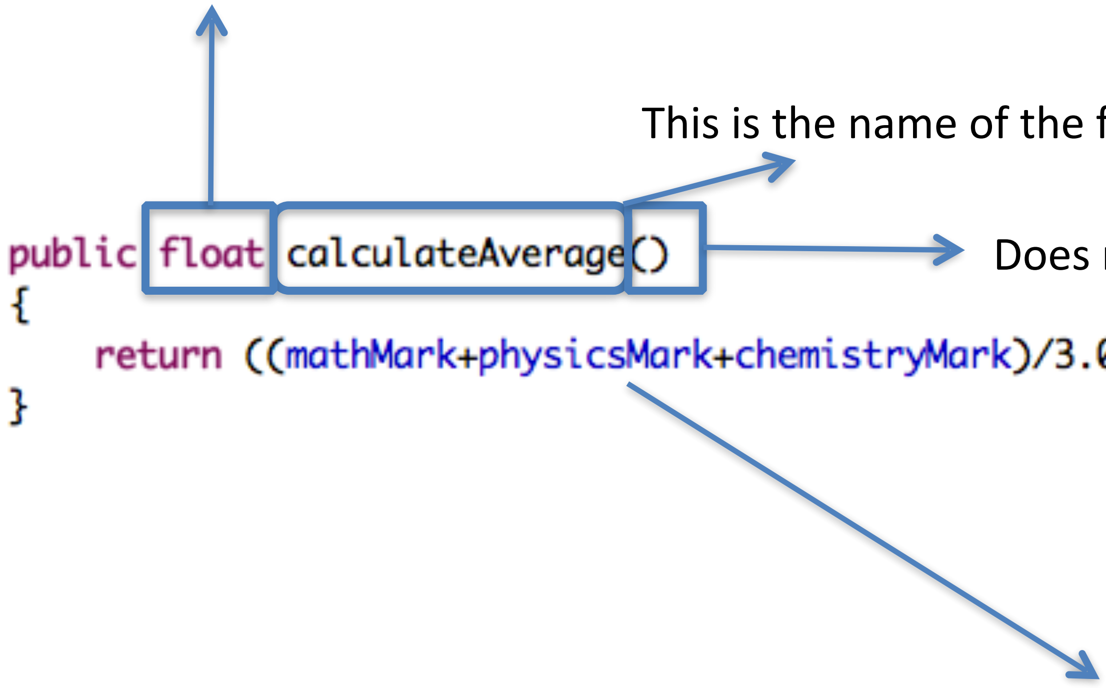
```
public float calculateAverage()
{
    return ((mathMark+physicsMark+chemistryMark)/3.0f);
}
```

Returns back a value of type float
This is the name of the function
Does not take in anything as input

Lets add an instance method that calculates the average in the student class.

- Returns back a value of type float

This is the name of the function

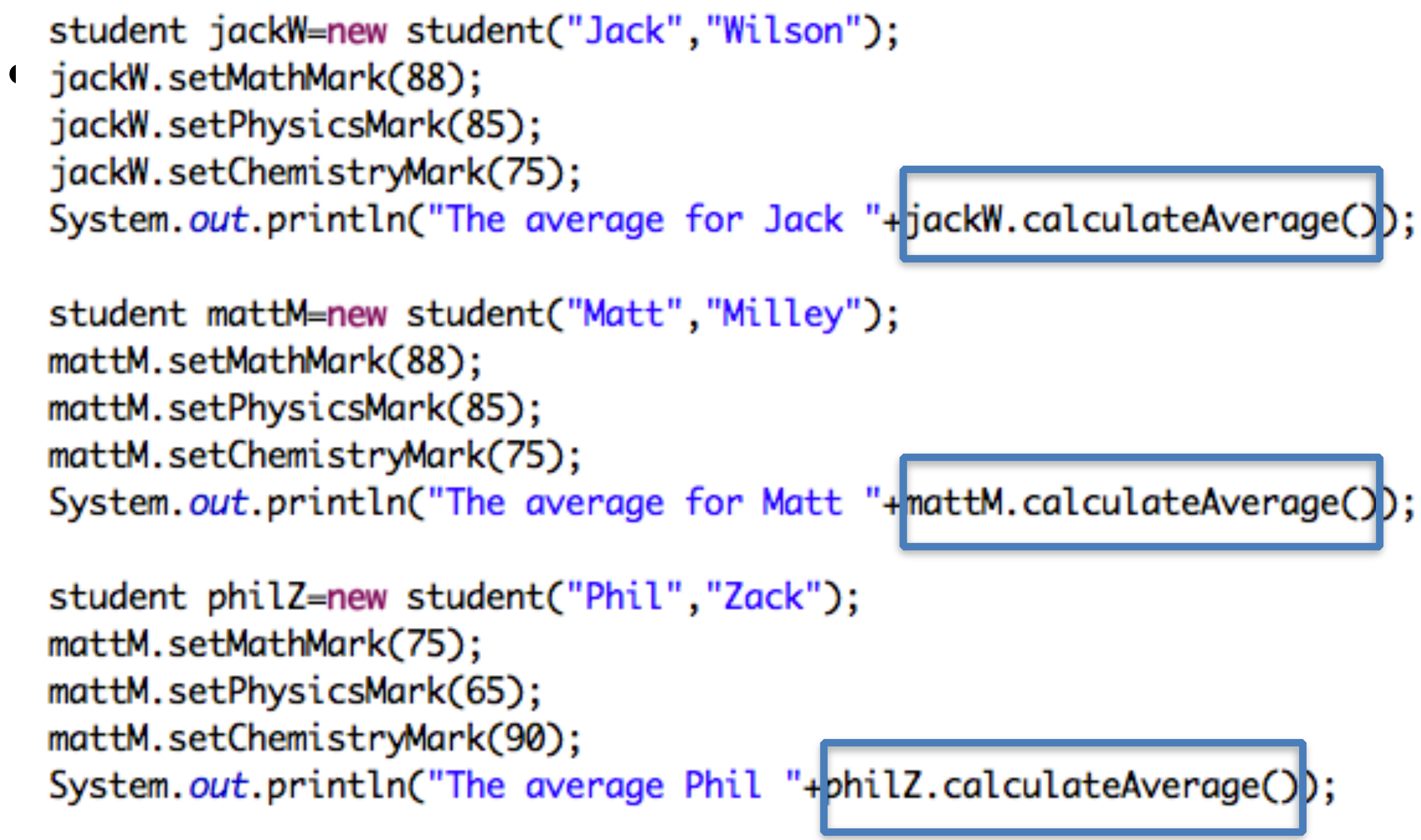


```
public float calculateAverage()
{
    return ((mathMark+physicsMark+chemistryMark)/3.0f);
}
```

Does not take in anything as input

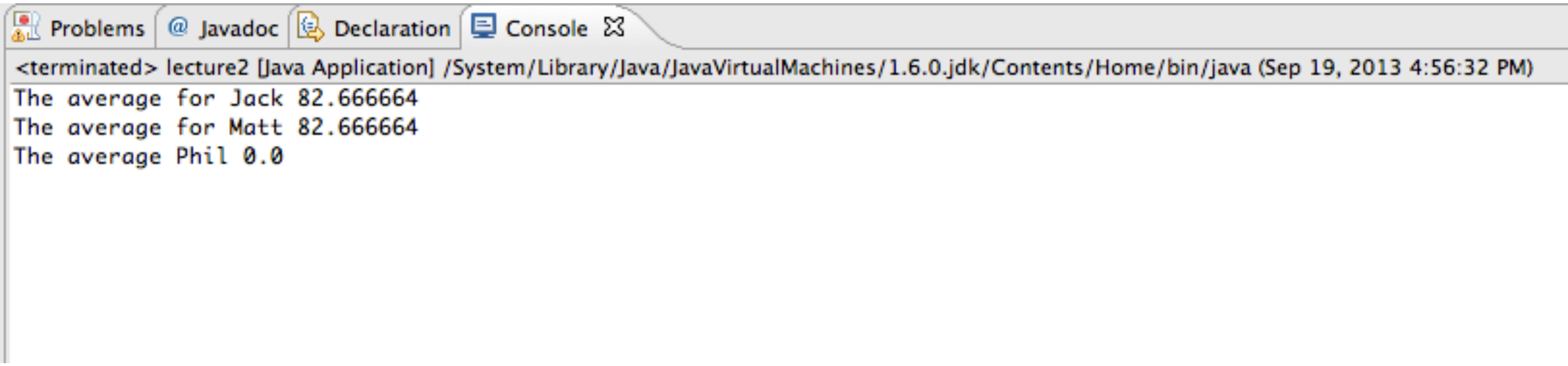
Whose mathMark , physicsMark, chemistryMark are we referring to? Jack or Matt or Phil?

Lets use this method.



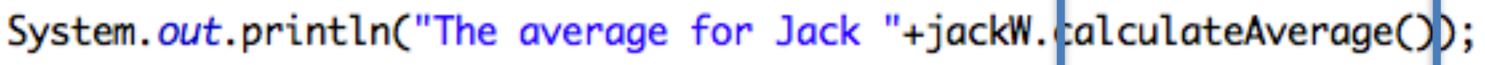
```
student jackW=new student("Jack","Wilson");
jackW.setMathMark(88);
jackW.setPhysicsMark(85);
jackW.setChemistryMark(75);
System.out.println("The average for Jack "+jackW.calculateAverage());

student mattM=new student("Matt","Milley");
mattM.setMathMark(88);
mattM.setPhysicsMark(85);
mattM.setChemistryMark(75);
System.out.println("The average for Matt "+mattM.calculateAverage());

student philZ=new student("Phil","Zack");
mattM.setMathMark(75);
mattM.setPhysicsMark(65);
mattM.setChemistryMark(90);
System.out.println("The average Phil "+philZ.calculateAverage());
```

Output from the previous slide.



```
<terminated> lecture2 [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Sep 19, 2013 4:56:32 PM)
The average for Jack 82.666664
The average for Matt 82.666664
The average Phil 0.0
```

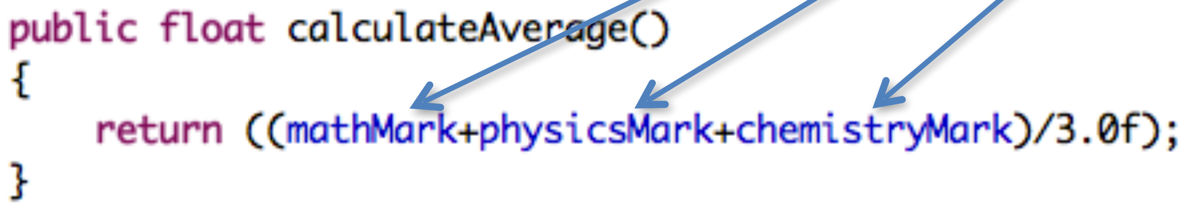
What went wrong?



```
System.out.println("The average for Jack "+jackW.calculateAverage());
```

•

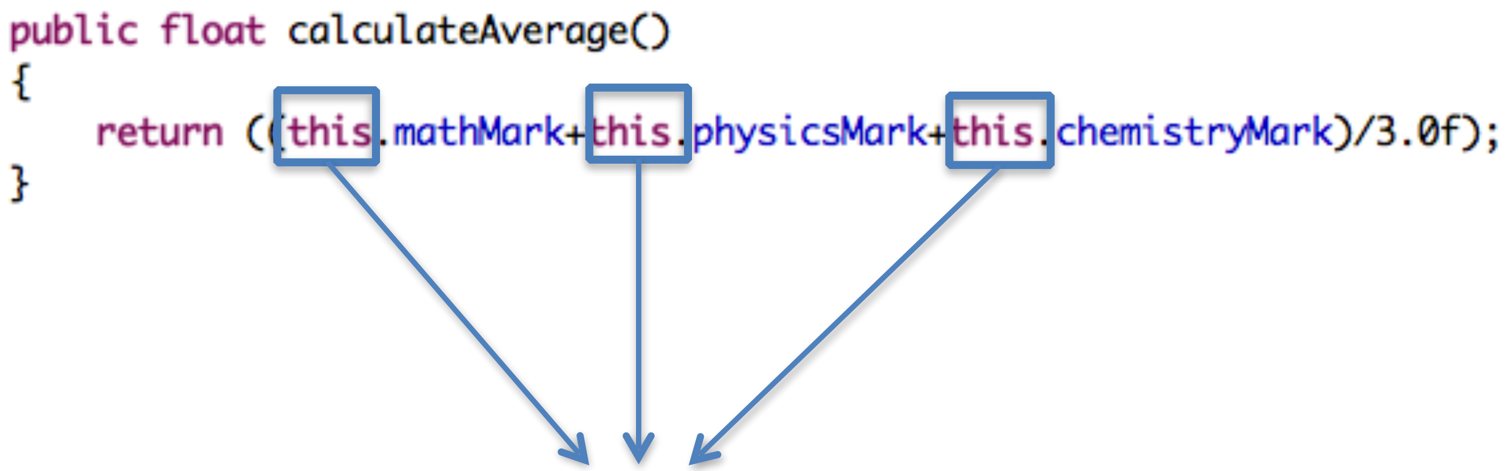
The method **calculateAverage()**
Is being called on the **jackW reference**



```
public float calculateAverage()
{
    return ((mathMark+physicsMark+chemistryMark)/3.0f);
}
```

This is identical to the previous method

•



```
public float calculateAverage()
{
    return ((this.mathMark+this.physicsMark+this.chemistryMark)/3.0f);
}
```

What does 'this' refer to?

Recap:

- **Instance Methods**: Are methods that depend on an instance.
- **What is an instance?** Instances are objects that you create using the ‘new’ keyword followed by the name of the constructor.
- Some **examples of instance methods**: calculateAverage, setMathMark, getMathMark, etc.

Demo time

- Lets understand what ***public*** and ***private*** means.

- We will look at ***protected*** next week.

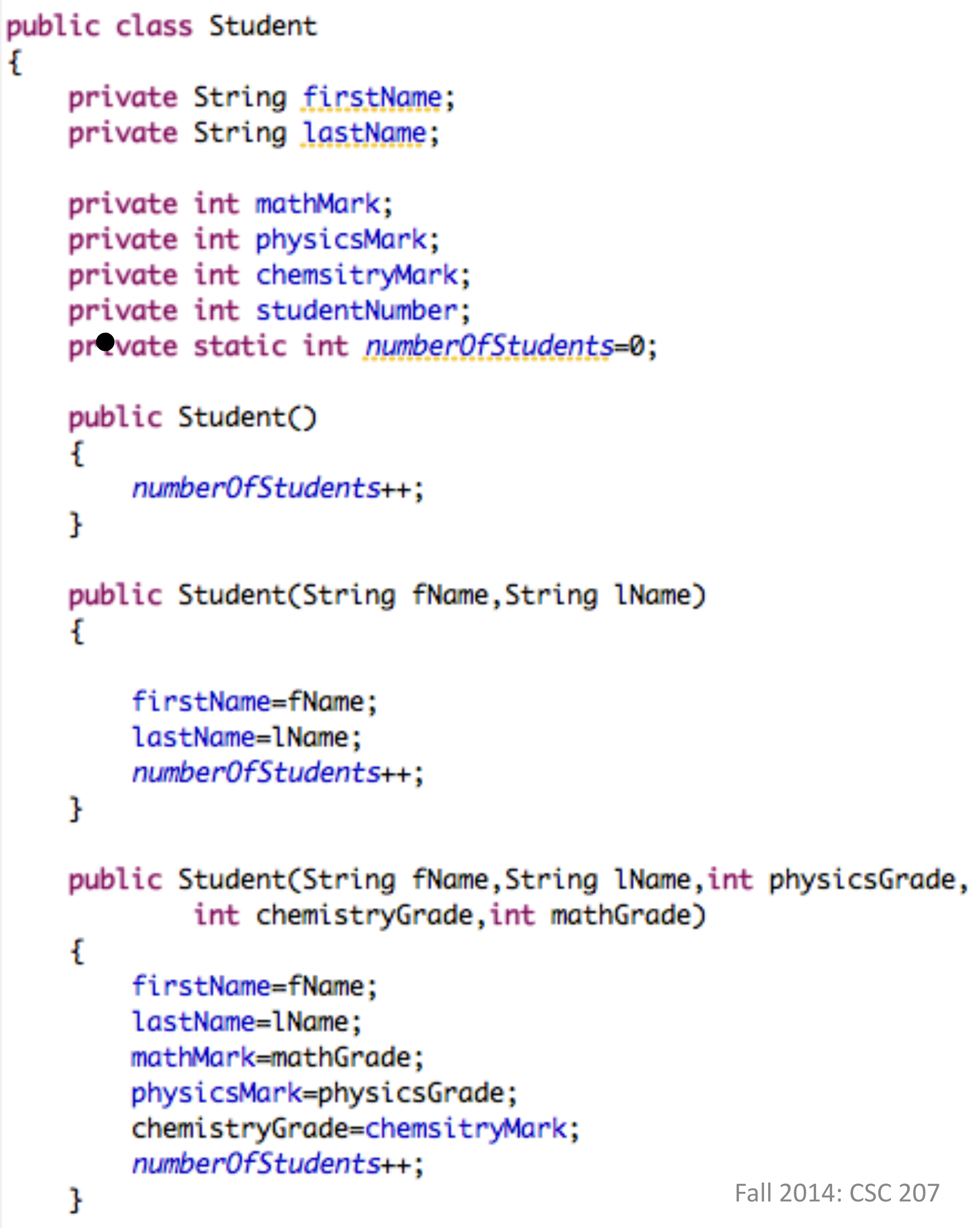
Static fields

- Sometimes you want variables that are common across all objects of a certain class.
- Static fields are associated with a **class** rather than an **instance** or an object.
- Lets say that you like to keep track of how many students object have been created.
- Ask yourself, does this field depend on any one particular instance or object? If you recall, mathMark instance variable depended on the instance (Jack or Matt or Phil) but keeping track of number of students does not depend on any single instance.

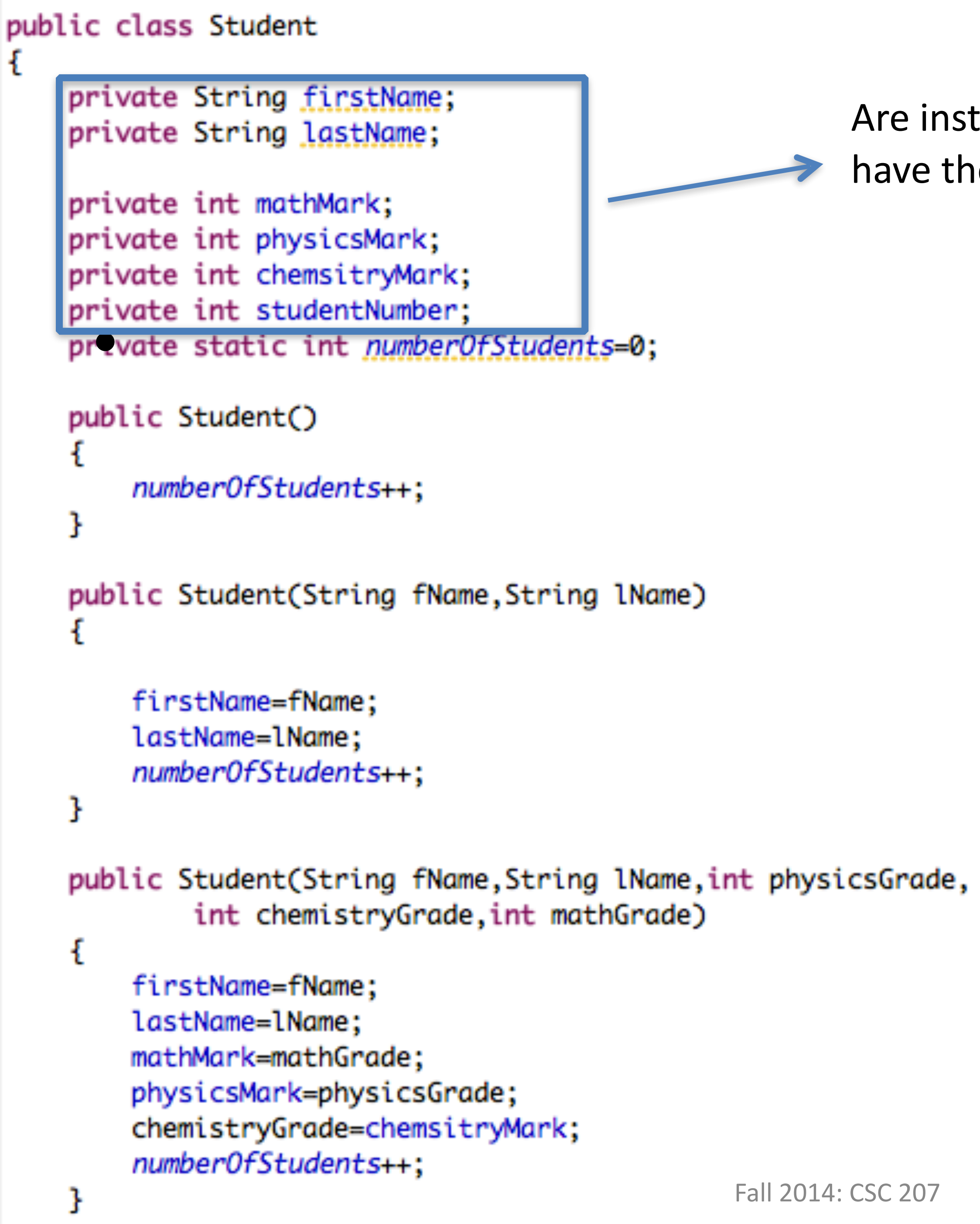
When to use static fields in Java?

- Does it make sense to call this method even if no object of that class has been created yet?
- If yes, then that method must be static.
- So in a class Car you might have a method `double convertMpgToKpl(double mpg)` which would be static, because one might want to know what 35mpg converts to, even if nobody has ever built a Car.
- But `void setMileage(double mpg)` (which sets the efficiency of one particular Car) can't be static since it's inconceivable to call the method before any Car has been constructed.

http://stackoverflow.com/questions/2671496/java-when-to-use-static-methods

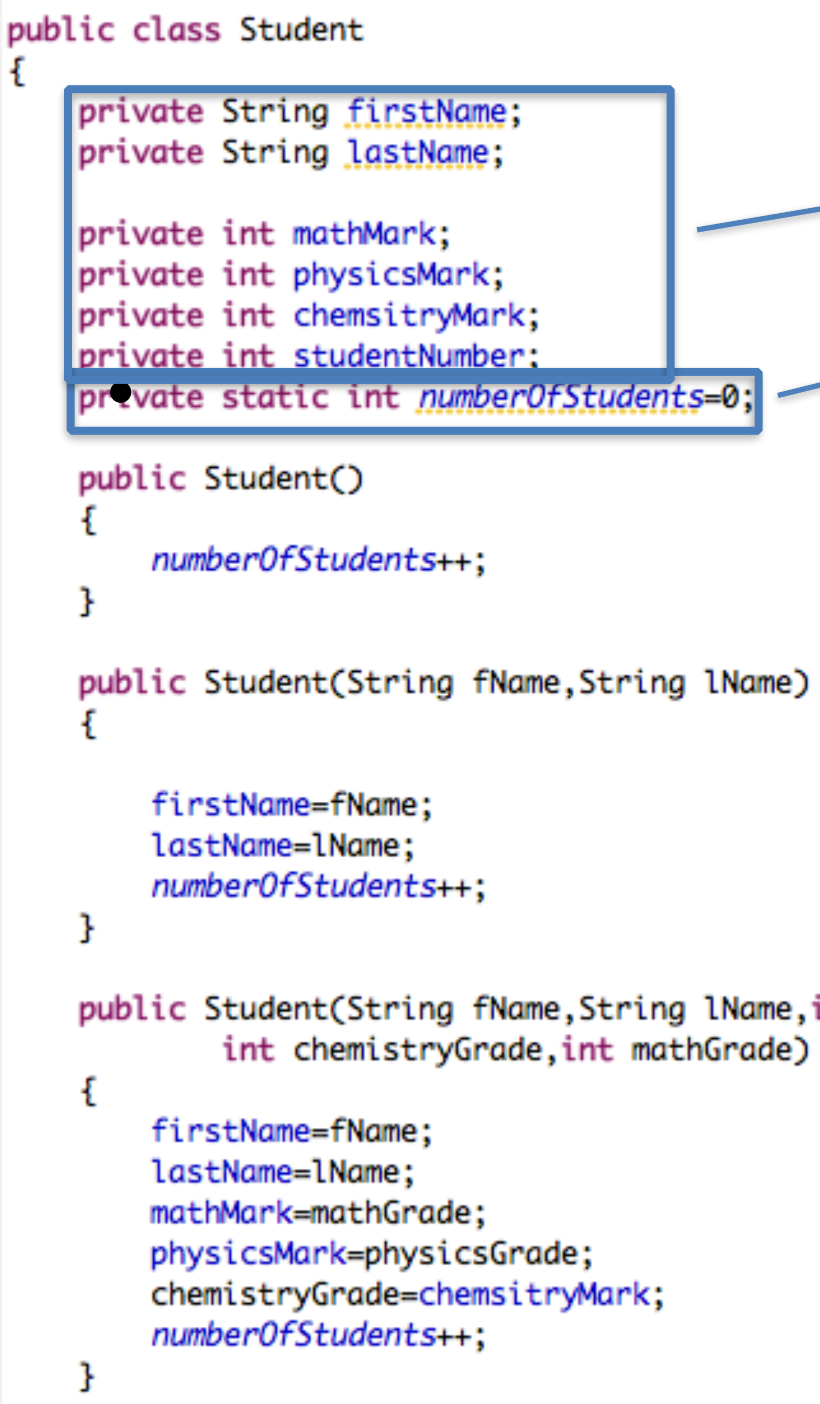


```java
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;

    public Student()
    {
        numberOfStudents++;
    }

    public Student(String fName,String lName)
    {

        firstName=fName;
        lastName=lName;
        numberOfStudents++;
    }

    public Student(String fName,String lName,int physicsGrade,
            int chemistryGrade,int mathGrade)
    {
        firstName=fName;
        lastName=lName;
        mathMark=mathGrade;
        physicsMark=physicsGrade;
        chemistryGrade=chemsitryMark;
        numberOfStudents++;
    }
```



```java
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;

    public Student()
    {
        numberOfStudents++;
    }

    public Student(String fName,String lName)
    {

        firstName=fName;
        lastName=lName;
        numberOfStudents++;
    }

    public Student(String fName,String lName,int physicsGrade,
            int chemistryGrade,int mathGrade)
    {
        firstName=fName;
        lastName=lName;
        mathMark=mathGrade;
        physicsMark=physicsGrade;
        chemistryGrade=chemsitryMark;
        numberOfStudents++;
    }
```

Are instance variables because they don't have the `static` keyword



```java
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;

    public Student()
    {
        numberOfStudents++;
    }

    public Student(String fName,String lName)
    {

        firstName=fName;
        lastName=lName;
        numberOfStudents++;
    }

    public Student(String fName,String lName,int physicsGrade,
            int chemistryGrade,int mathGrade)
    {
        firstName=fName;
        lastName=lName;
        mathMark=mathGrade;
        physicsMark=physicsGrade;
        chemistryGrade=chemsitryMark;
        numberOfStudents++;
    }
```

Are instance variables because they don’t have the `static` keyword

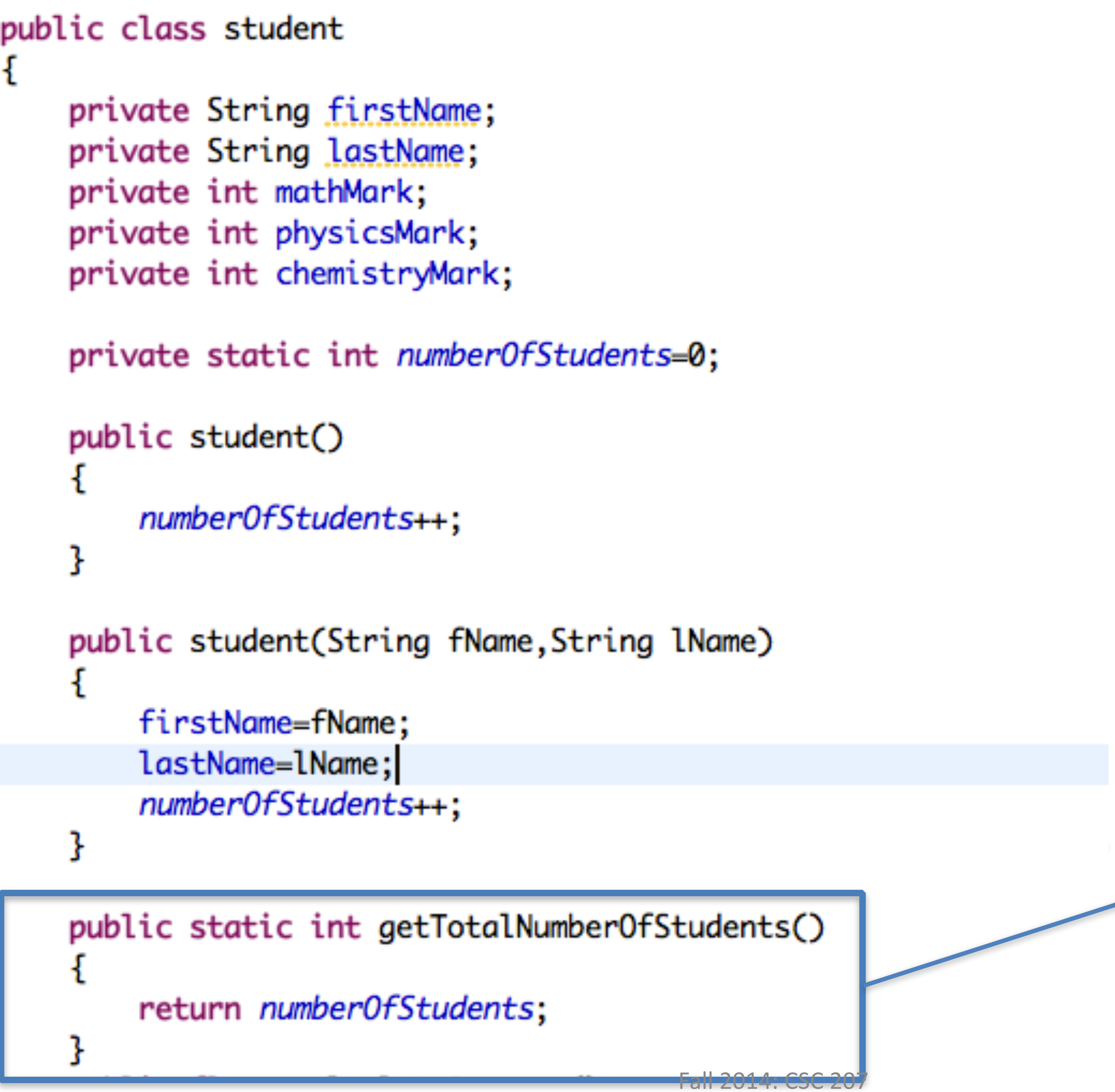
`numberOfStudents` is a static variable

```java
public class Student
{
    private String firstName;
    private String lastName;

    private int mathMark;
    private int physicsMark;
    private int chemsitryMark;
    private int studentNumber;
    private static int numberOfStudents=0;

    public Student()
    {
        numberOfStudents++;
    }

    public Student(String fName,String lName)
    {

        firstName=fName;
        lastName=lName;
        numberOfStudents++;
    }

    public Student(String fName,String lName,int physicsGrade,
            int chemistryGrade,int mathGrade)
    {
        firstName=fName;
        lastName=lName;
        mathMark=mathGrade;
        physicsMark=physicsGrade;
        chemistryGrade=chemsitryMark;
        numberOfStudents++;
    }
```

Are instance variables because they don't have the `static` keyword

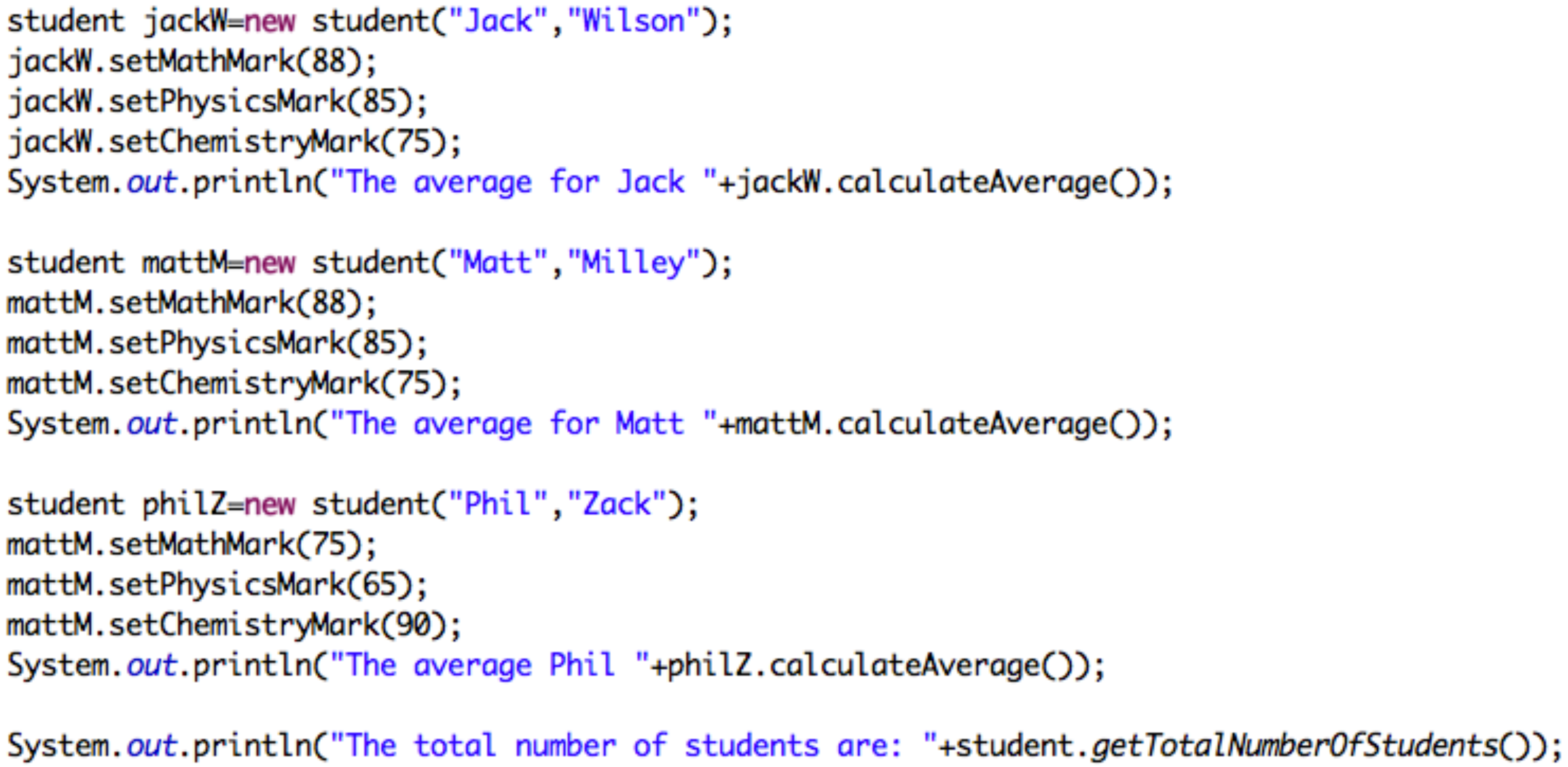
`numberOfStudents` is a static variable

I increment `numberOfStudents` in the constructor because the constructor is called only when the `Student` object is created.

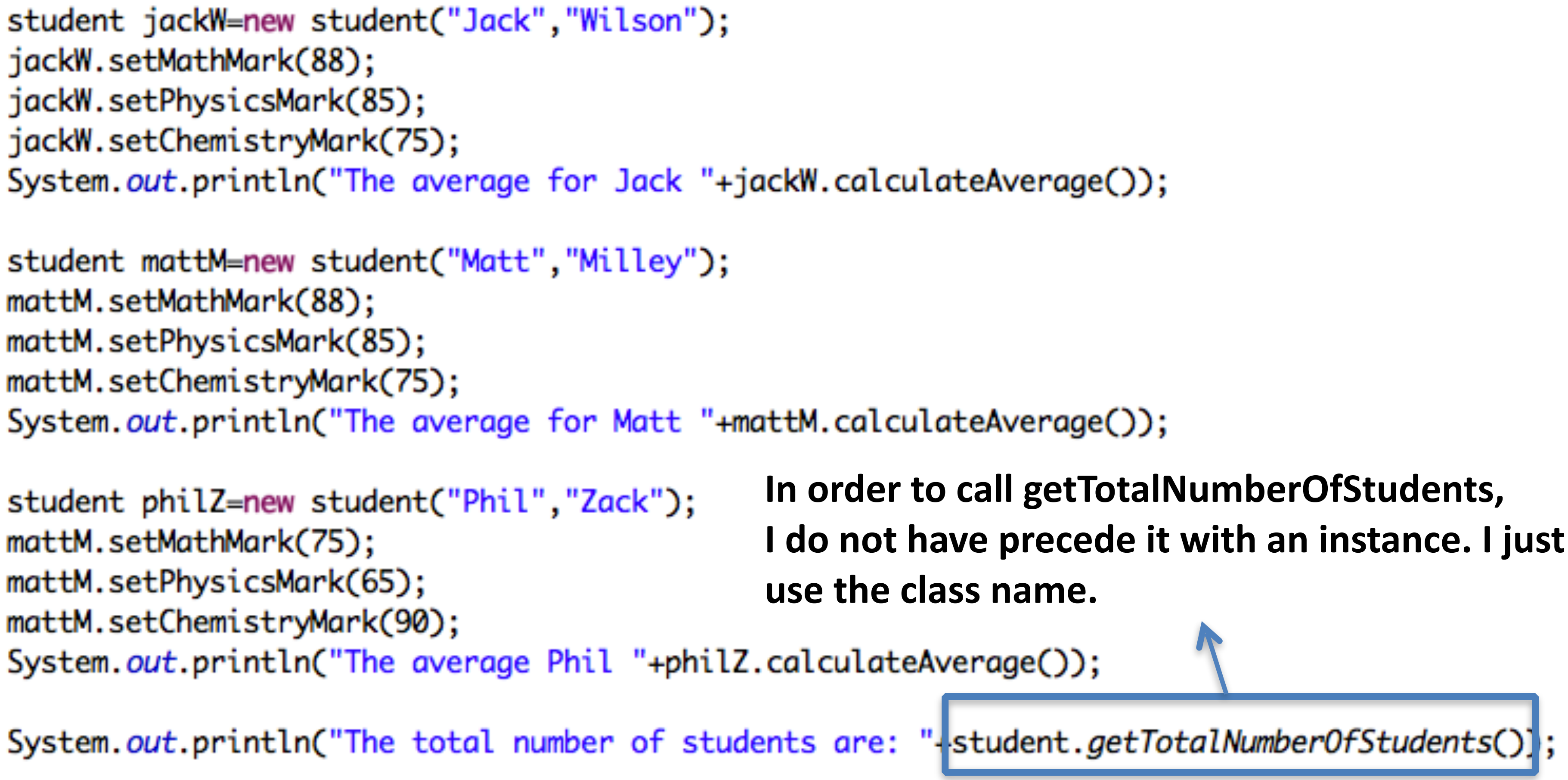


```
public class student
{
    private String firstName;
    private String lastName;
    private int mathMark;
    private int physicsMark;
    private int chemistryMark;

    private static int numberOfStudents=0;

    public student()
    {
        numberOfStudents++;
    }

    public student(String fName,String lName)
    {
        firstName=fName;
        lastName=lName;
        numberOfStudents++;
    }

    public static int getTotalNumberOfStudents()
    {
        return numberOfStudents;
    }
```

Static method that also does not depend on the instances



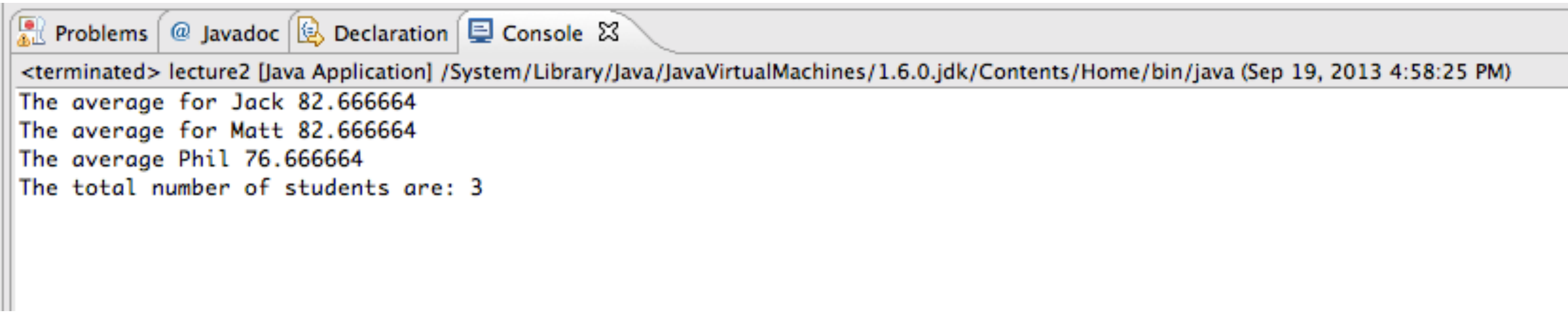
```
student jackW=new student("Jack","Wilson");
jackW.setMathMark(88);
jackW.setPhysicsMark(85);
jackW.setChemistryMark(75);
System.out.println("The average for Jack "+jackW.calculateAverage());

student mattM=new student("Matt","Milley");
mattM.setMathMark(88);
mattM.setPhysicsMark(85);
mattM.setChemistryMark(75);
System.out.println("The average for Matt "+mattM.calculateAverage());

student philZ=new student("Phil","Zack");
mattM.setMathMark(75);
mattM.setPhysicsMark(65);
mattM.setChemistryMark(90);
System.out.println("The average Phil "+philZ.calculateAverage());

System.out.println("The total number of students are: "+student.getTotalNumberOfStudents());
```



```
student jackW=new student("Jack","Wilson");
jackW.setMathMark(88);
jackW.setPhysicsMark(85);
jackW.setChemistryMark(75);
System.out.println("The average for Jack "+jackW.calculateAverage());

student mattM=new student("Matt","Milley");
mattM.setMathMark(88);
mattM.setPhysicsMark(85);
mattM.setChemistryMark(75);
System.out.println("The average for Matt "+mattM.calculateAverage());

student philZ=new student("Phil","Zack");
mattM.setMathMark(75);
mattM.setPhysicsMark(65);
mattM.setChemistryMark(90);
System.out.println("The average Phil "+philZ.calculateAverage());

System.out.println("The total number of students are: "+student.getTotalNumberOfStudents());
```

In order to call getTotalNumberOfStudents, I do not have precede it with an instance. I just use the class name.

Output from previous slide.



```
<terminated> lecture2 [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Sep 19, 2013 4:58:25 PM)
The average for Jack 82.666664
The average for Matt 82.666664
The average Phil 76.666664
The total number of students are: 3
```

Local variables **vs** instance variables?

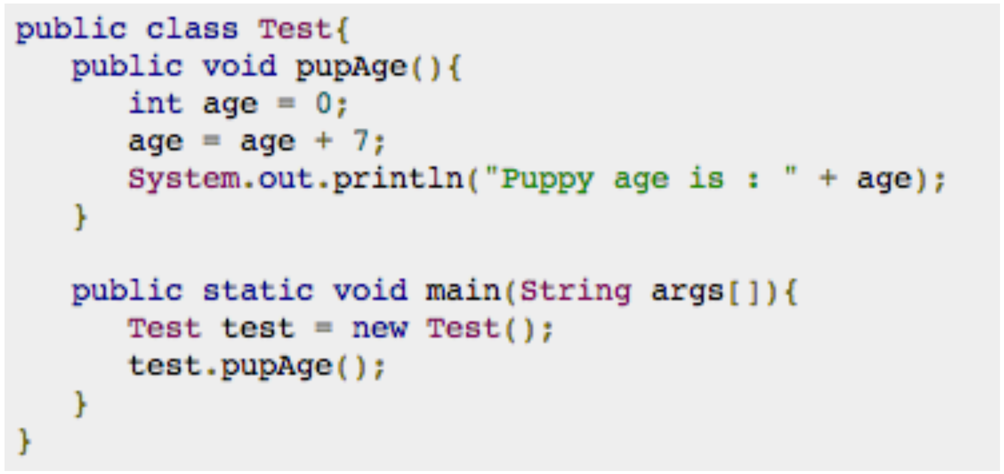
- <u>Local Variables:</u>

1) Declared in methods or constructors or blocks.
2) Created when the constructor, method, or block is entered and destroyed once it exists the method, constructor or block.
3) Visible only within the declared method, constructor or block.
4) Are created on the stack in memory

- <u>Instance Variables:</u>

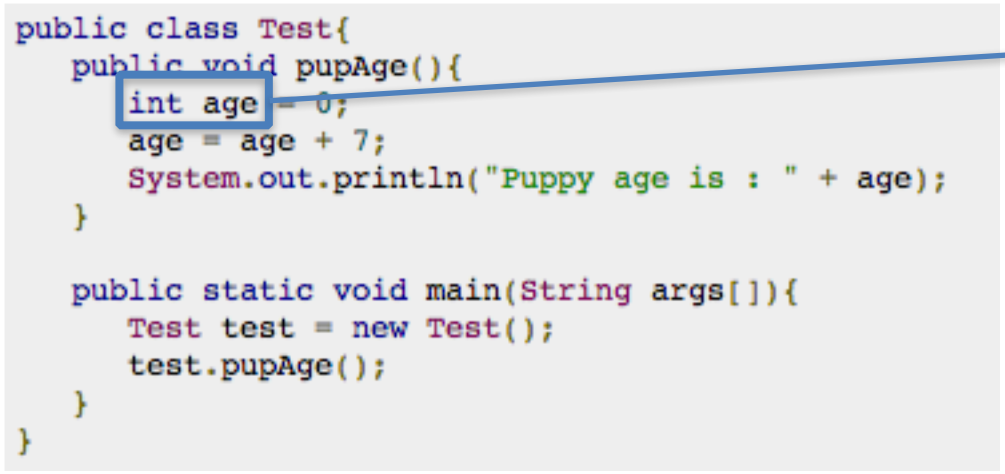
1) Instance variables are declared in class, but outside a method, constructor or any block.
2) When memory is allocated for an object in the heap, a slot for each instance variable is created on the heap
3) Instance variables are created when the object is created with the use of the keyword new and destroyed when the object is destroyed.
4) Visible across all methods, constructors and blocks in the class.

Example of **local** and **instance** variables.

- 

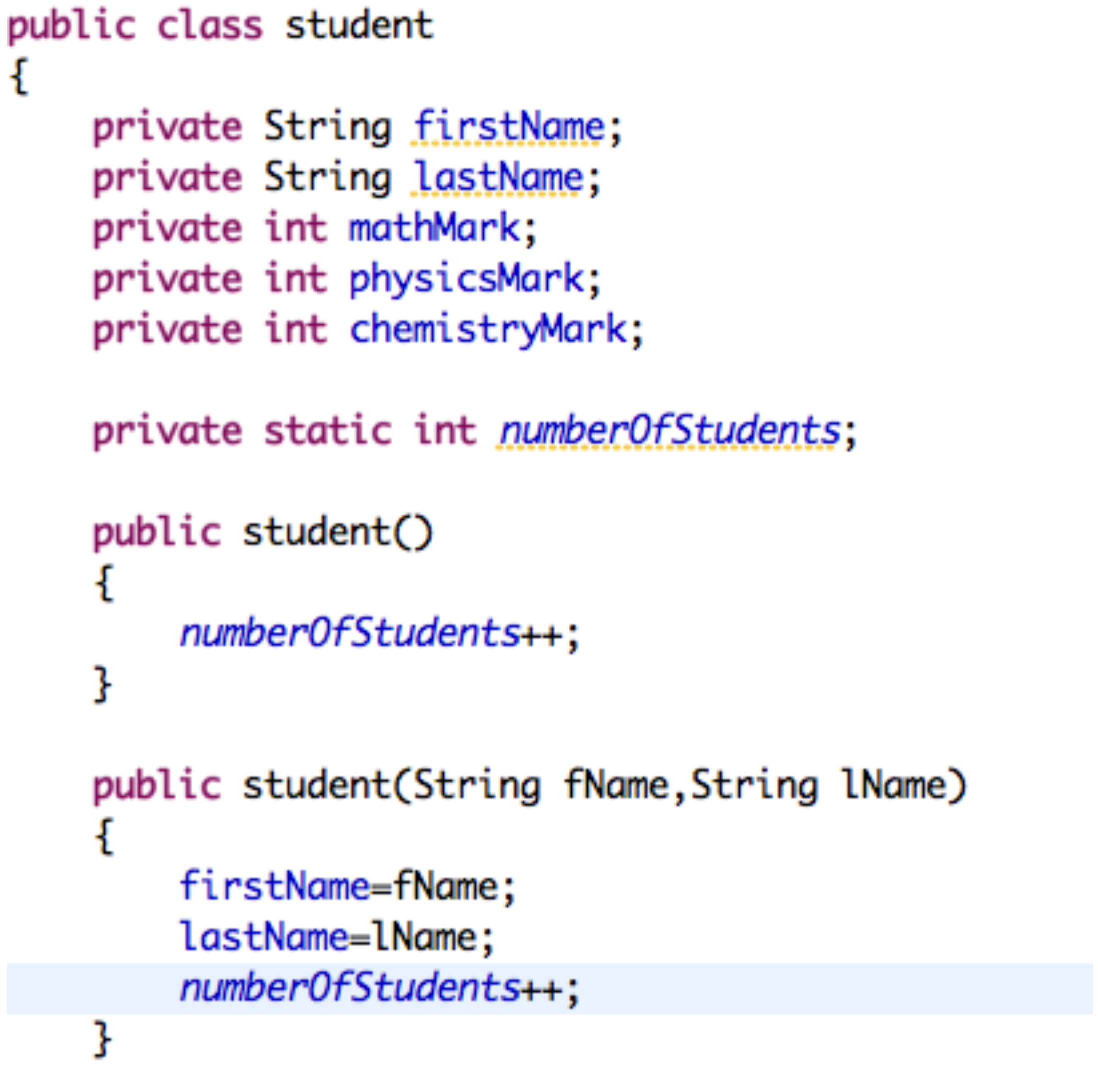
```
public class Test{
   public void pupAge(){
      int age = 0;
      age = age + 7;
      System.out.println("Puppy age is : " + age);
   }

   public static void main(String args[]){
      Test test = new Test();
      test.pupAge();
   }
}
```

Example of **local** and **instance** variables.

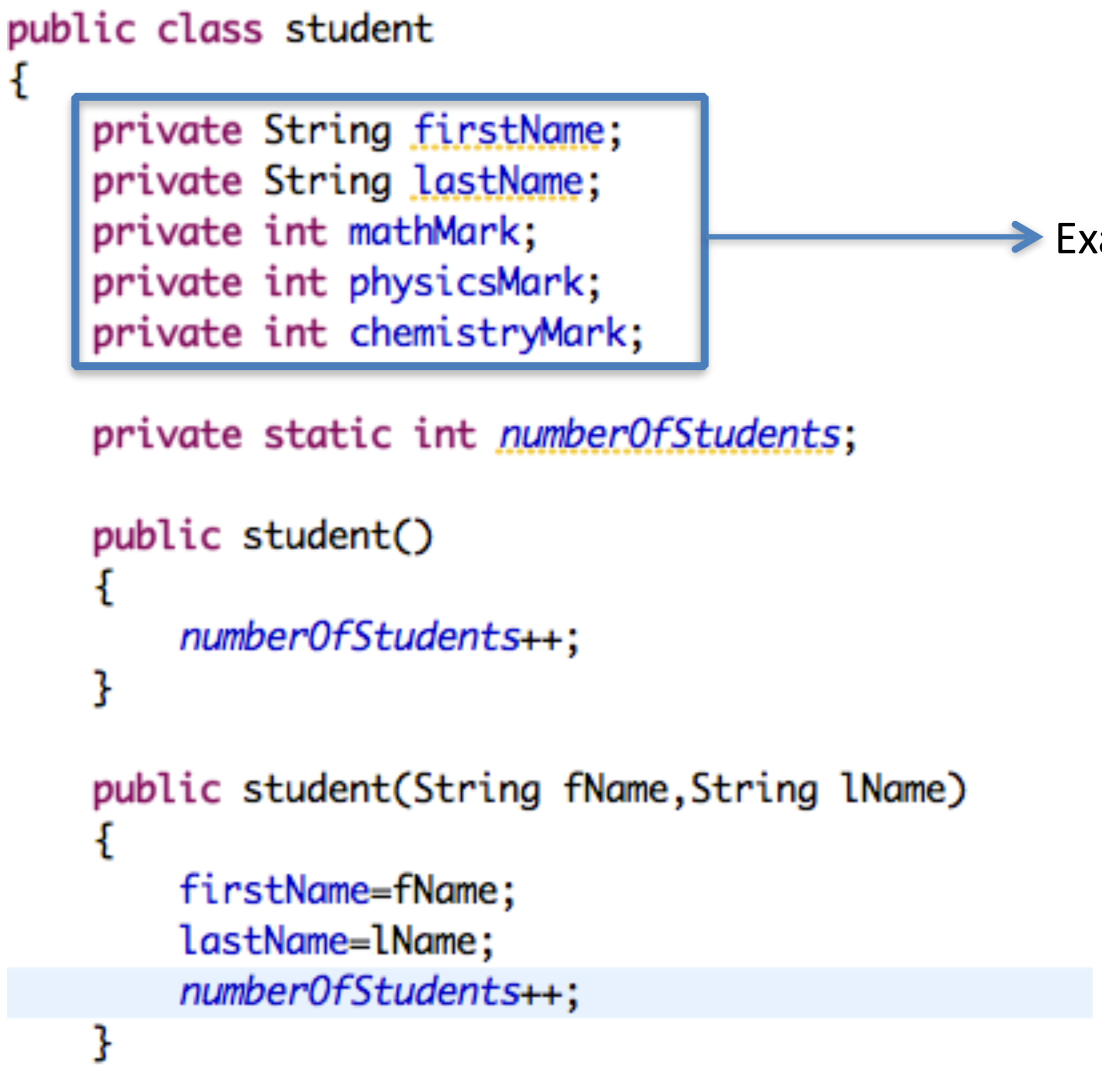
- 

```java
public class Test{
   public void pupAge(){
      int age = 0;
      age = age + 7;
      System.out.println("Puppy age is : " + age);
   }

   public static void main(String args[]){
      Test test = new Test();
      test.pupAge();
   }
}
```

age is a local variable

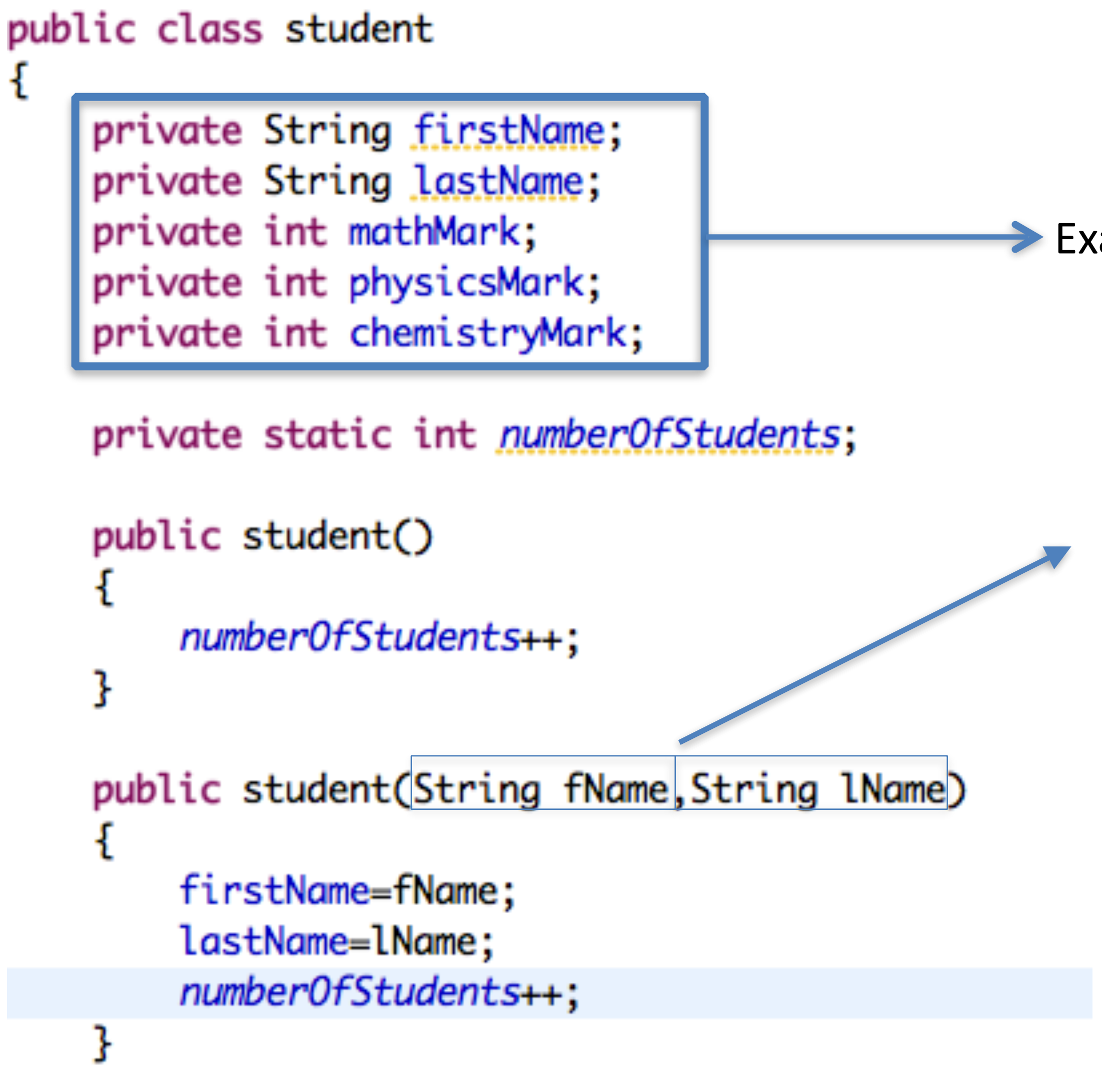


```
public class student
{
    private String firstName;
    private String lastName;
    private int mathMark;
    private int physicsMark;
    private int chemistryMark;

    private static int numberOfStudents;

    public student()
    {
        numberOfStudents++;
    }

    public student(String fName,String lName)
    {
        firstName=fName;
        lastName=lName;
        numberOfStudents++;
    }
```



```
public class student
{
    private String firstName;
    private String lastName;
    private int mathMark;
    private int physicsMark;
    private int chemistryMark;

    private static int numberOfStudents;

    public student()
    {
        numberOfStudents++;
    }

    public student(String fName,String lName)
    {
        firstName=fName;
        lastName=lName;
        numberOfStudents++;
    }
```

Example of instance variables.



```java
public class student
{
    private String firstName;
    private String lastName;
    private int mathMark;
    private int physicsMark;
    private int chemistryMark;

    private static int numberOfStudents;

    public student()
    {
        numberOfStudents++;
    }

    public student(String fName,String lName)
    {
        firstName=fName;
        lastName=lName;
        numberOfStudents++;
    }
```

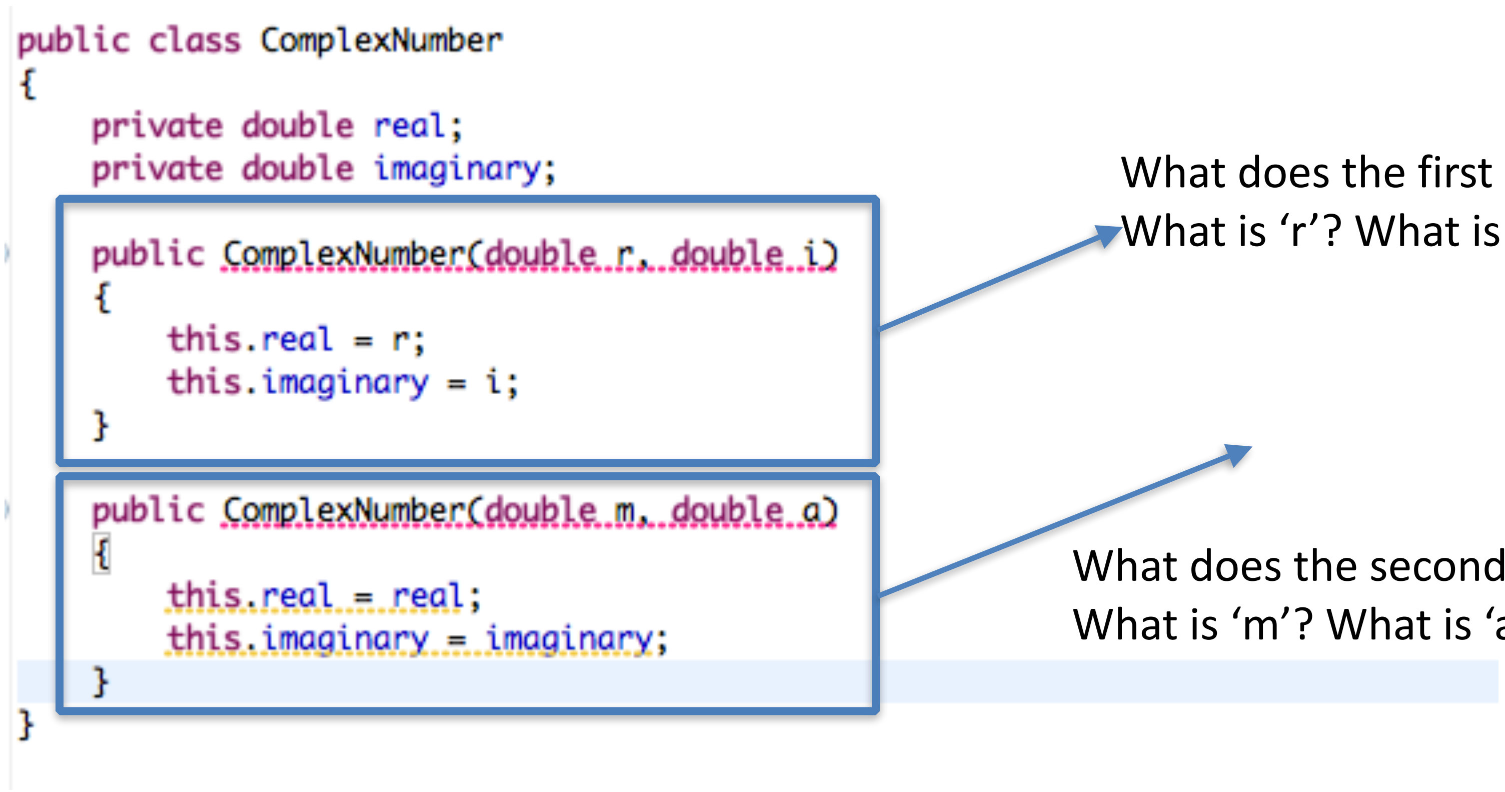
Example of instance variables.

Example of local variables.

Design Pattern 1:

- **What are factory methods?**
- Factory methods are considered as abstraction of a constructor.
- Unlike Constructors, factory methods have names and can be descriptive providing useful information on what the method does.

Not so good code.



```
public class ComplexNumber
{
    private double real;
    private double imaginary;

    public ComplexNumber(double r, double i)
    {
        this.real = r;
        this.imaginary = i;
    }

    public ComplexNumber(double m, double a)
    {
        this.real = real;
        this.imaginary = imaginary;
    }
}
```

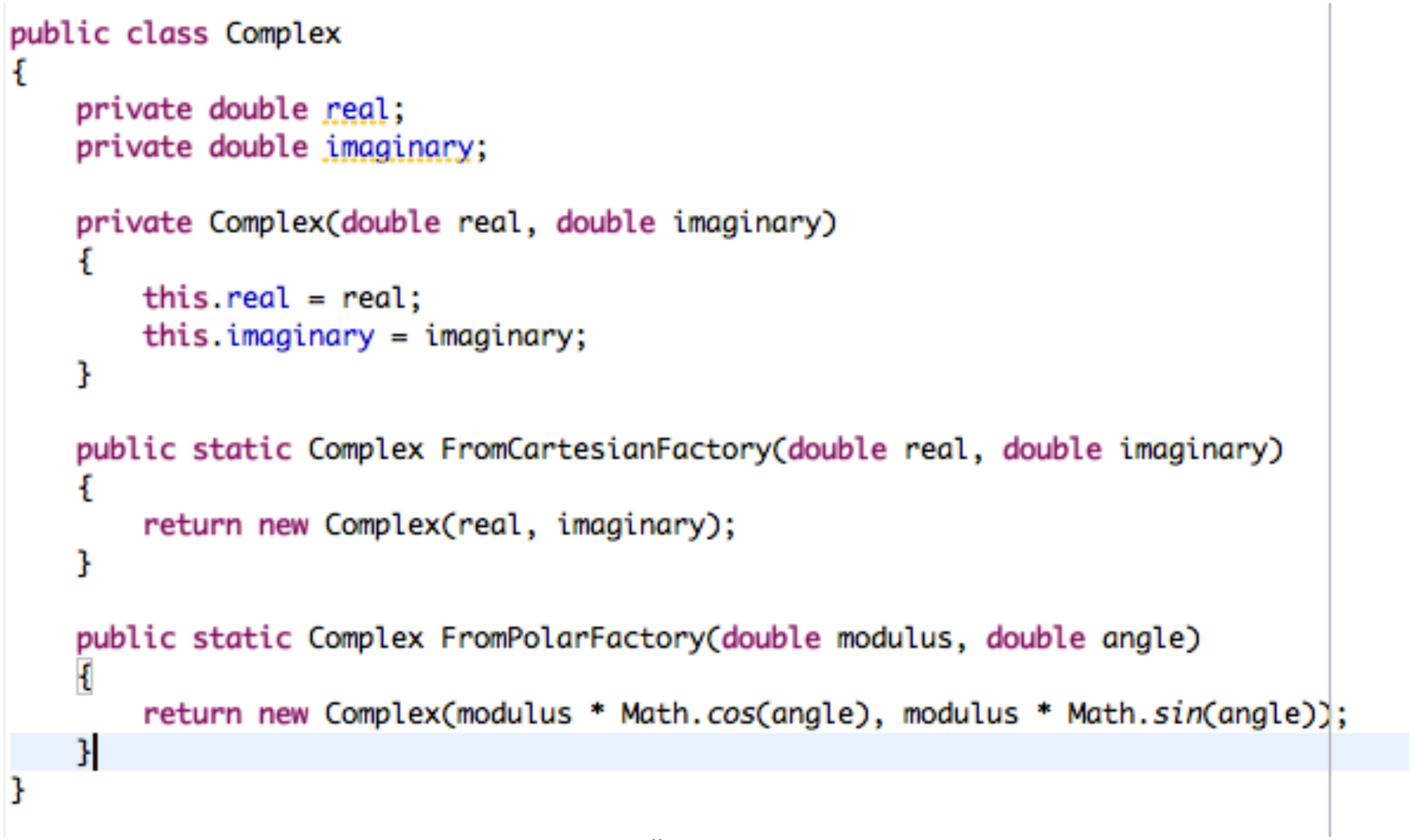
What does the first constructor do?
What is ‘r’? What is ‘i’?

What does the second constructor do?
What is ‘m’? What is ‘a’?

Why does’t this code compile? What is wrong here?

Using Factory Methods.

•

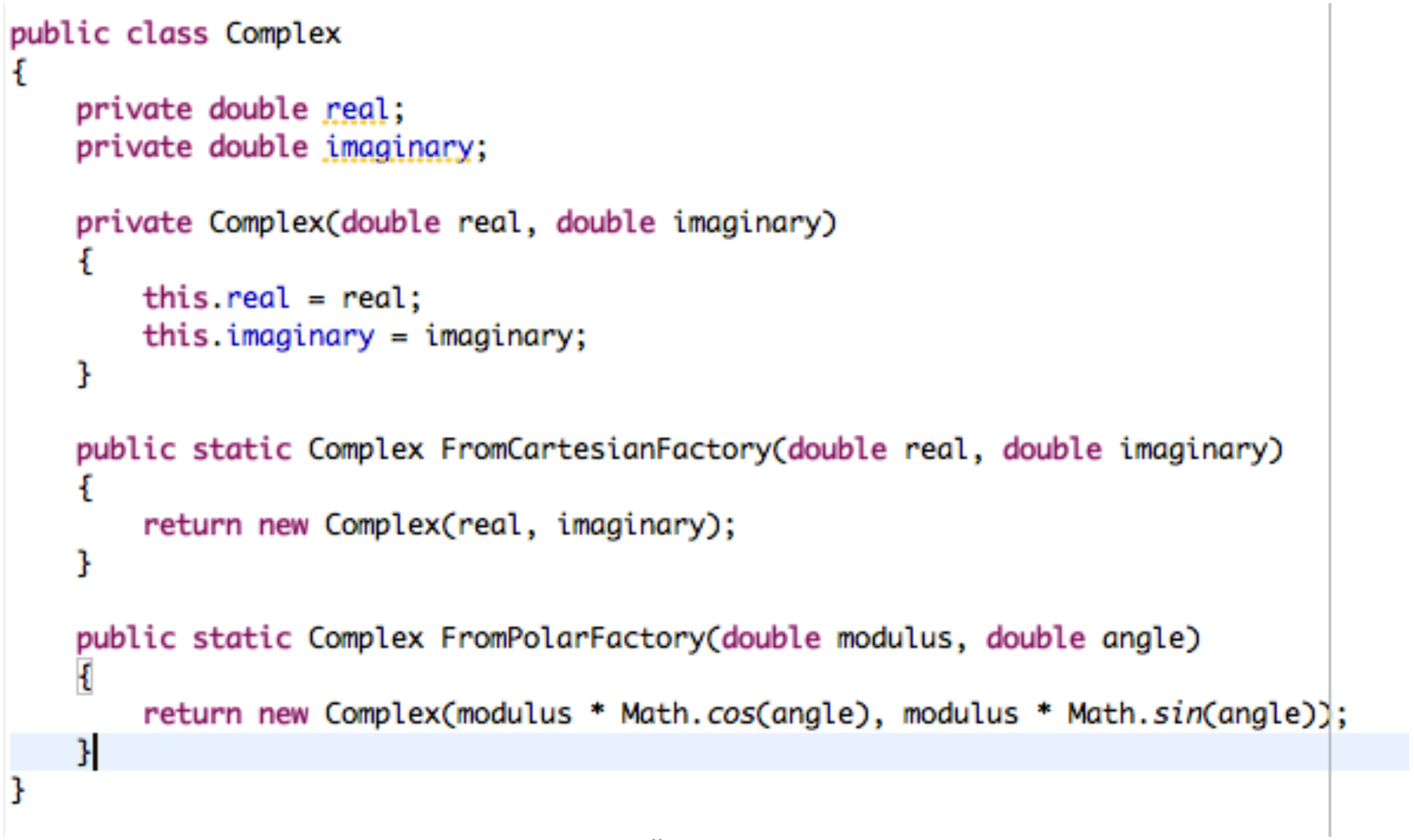


```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Using Factory Methods.

-

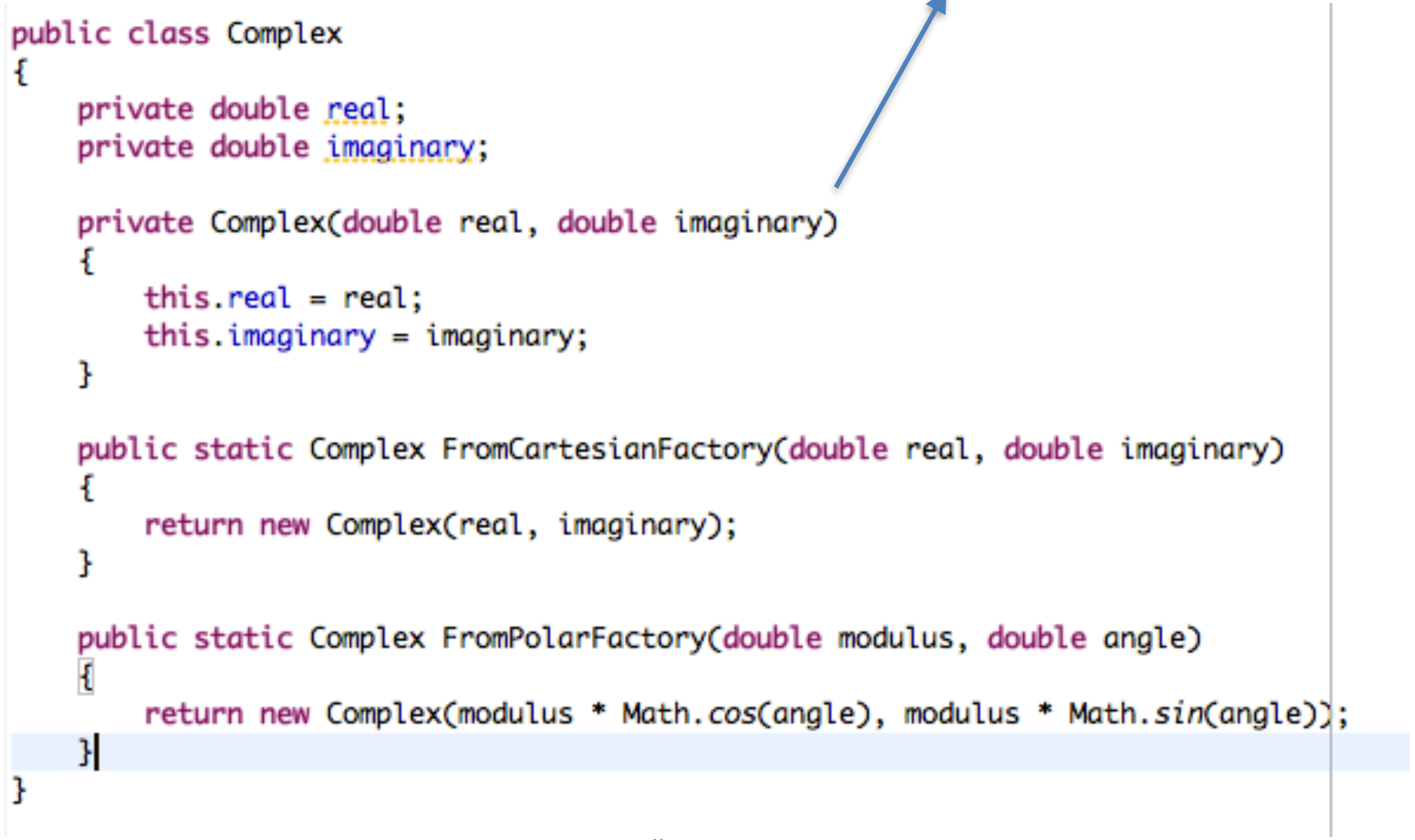
Why have this constructor as private?



```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Using Factory Methods.

Why have this constructor as private?

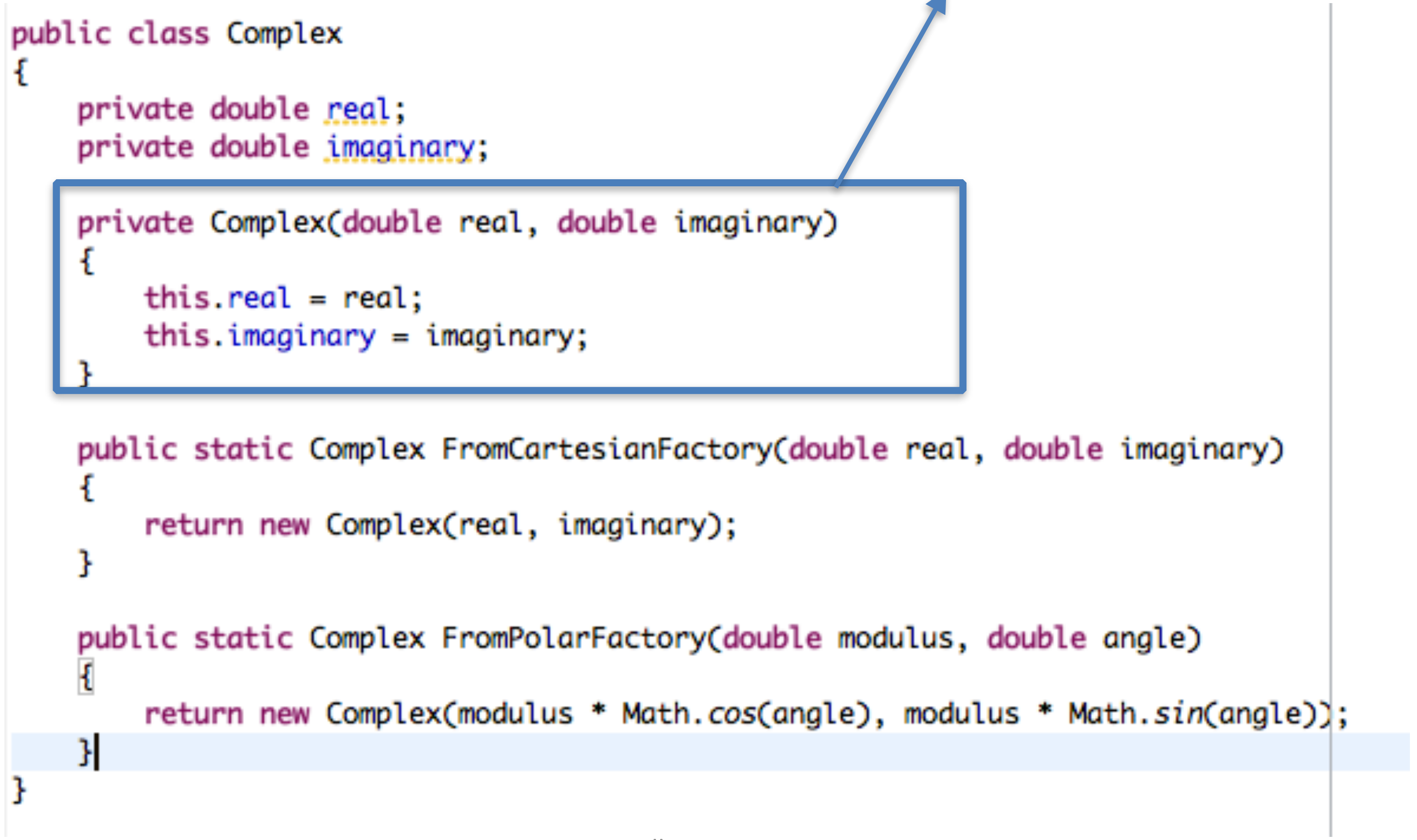
- 

```java
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Using Factory Methods.

-

Why have this constructor as private?

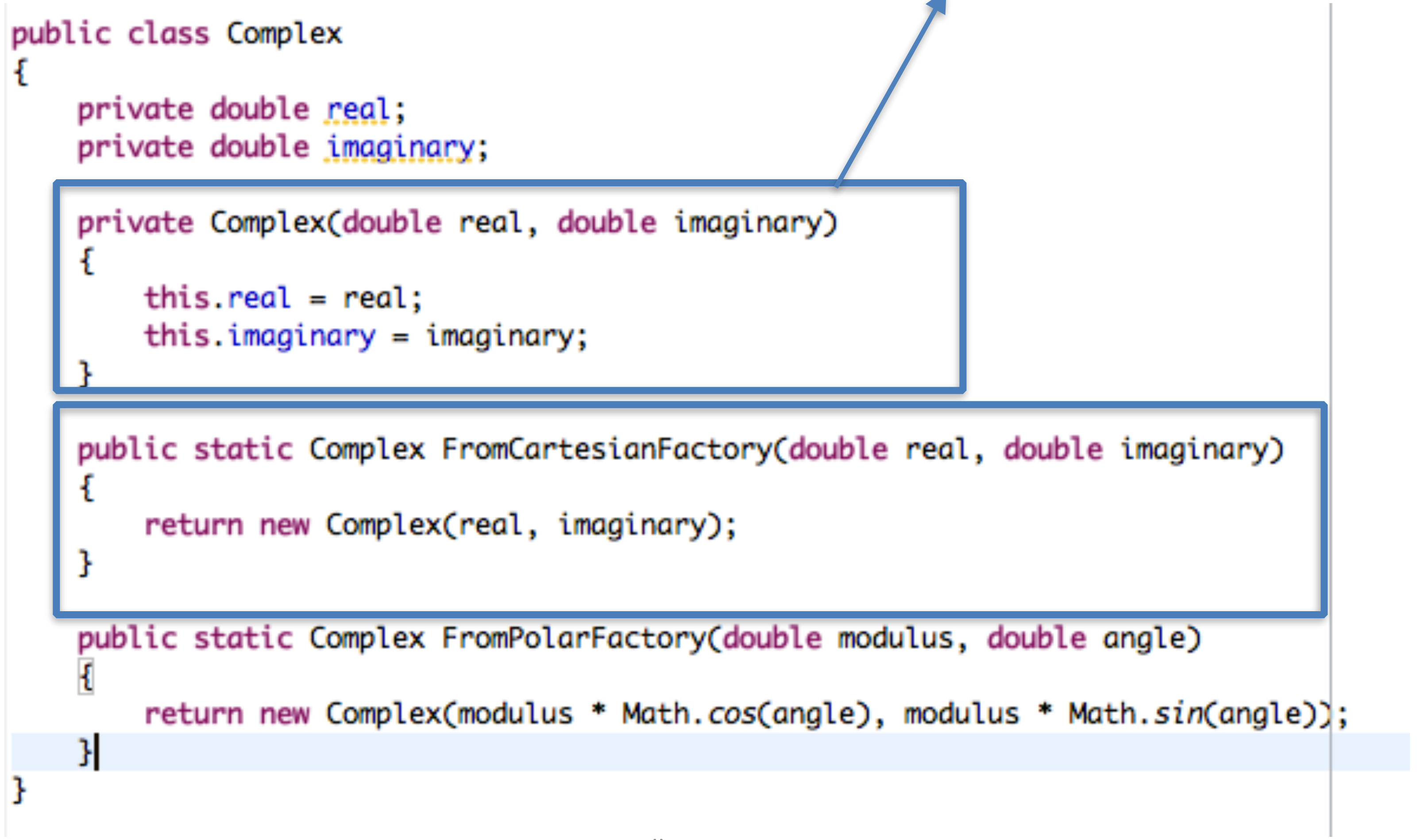


```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Using Factory Methods.

•

Why have this constructor as private?

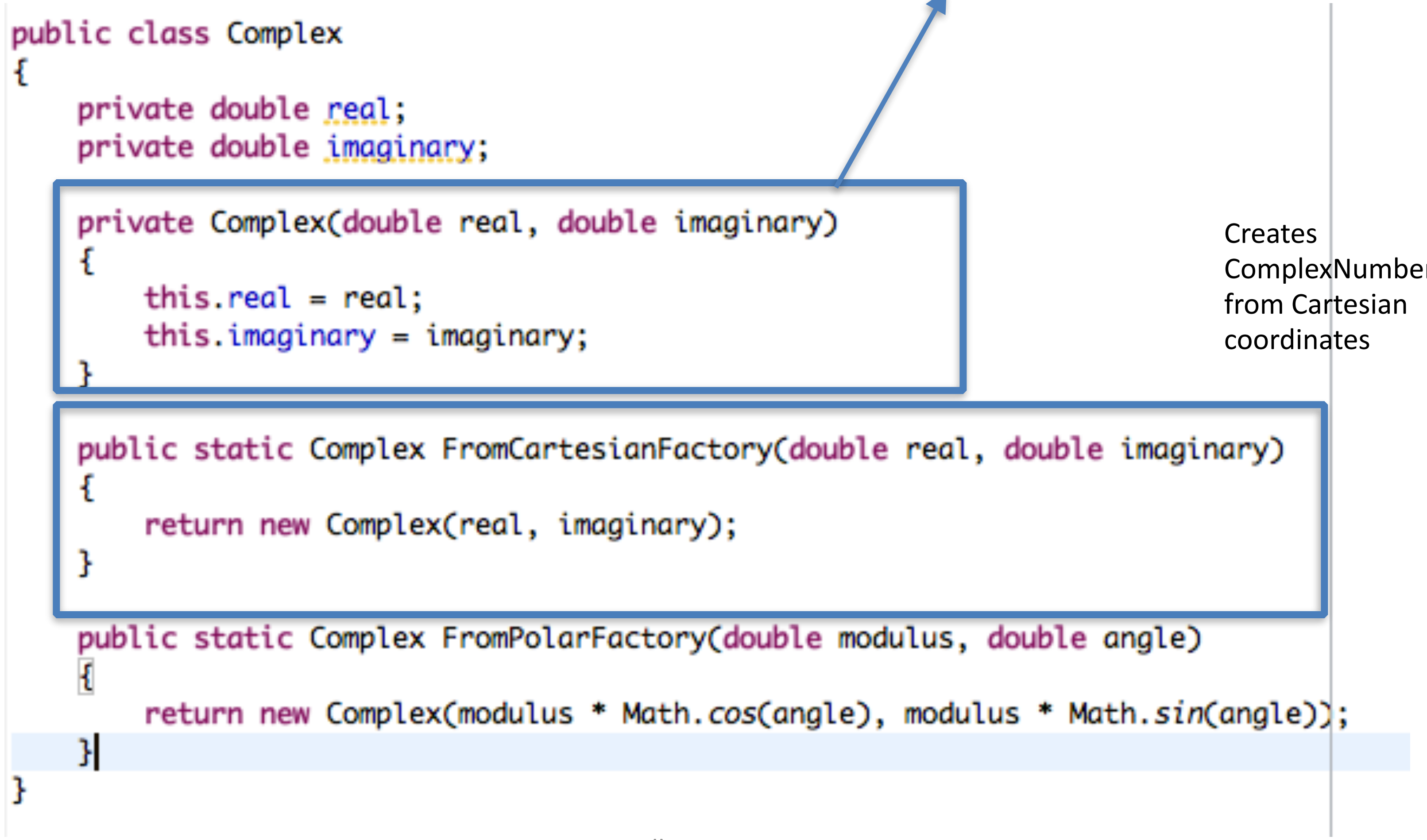


```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Using Factory Methods.

•

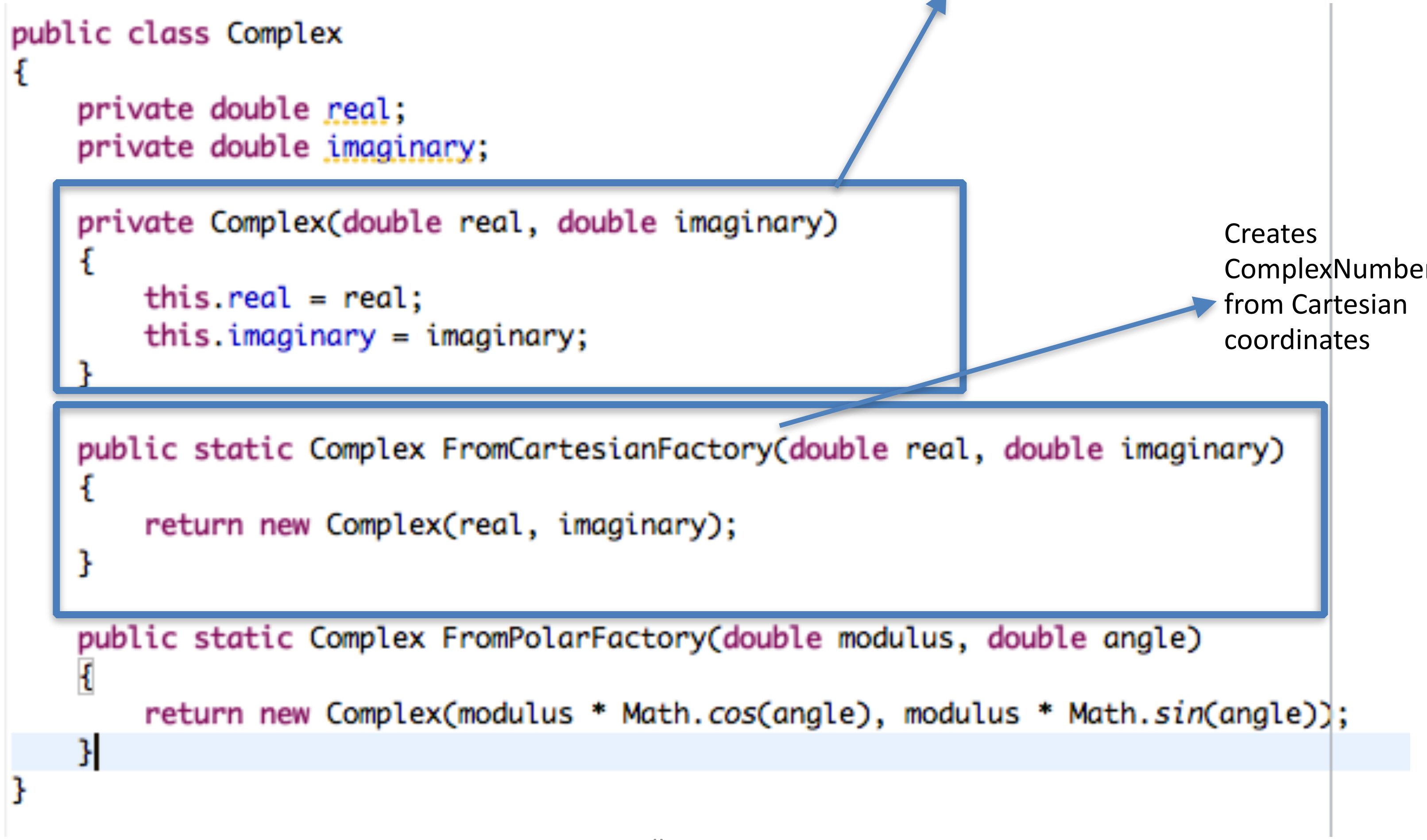
Why have this constructor as private?



```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Creates ComplexNumber from Cartesian coordinates

Using Factory Methods.

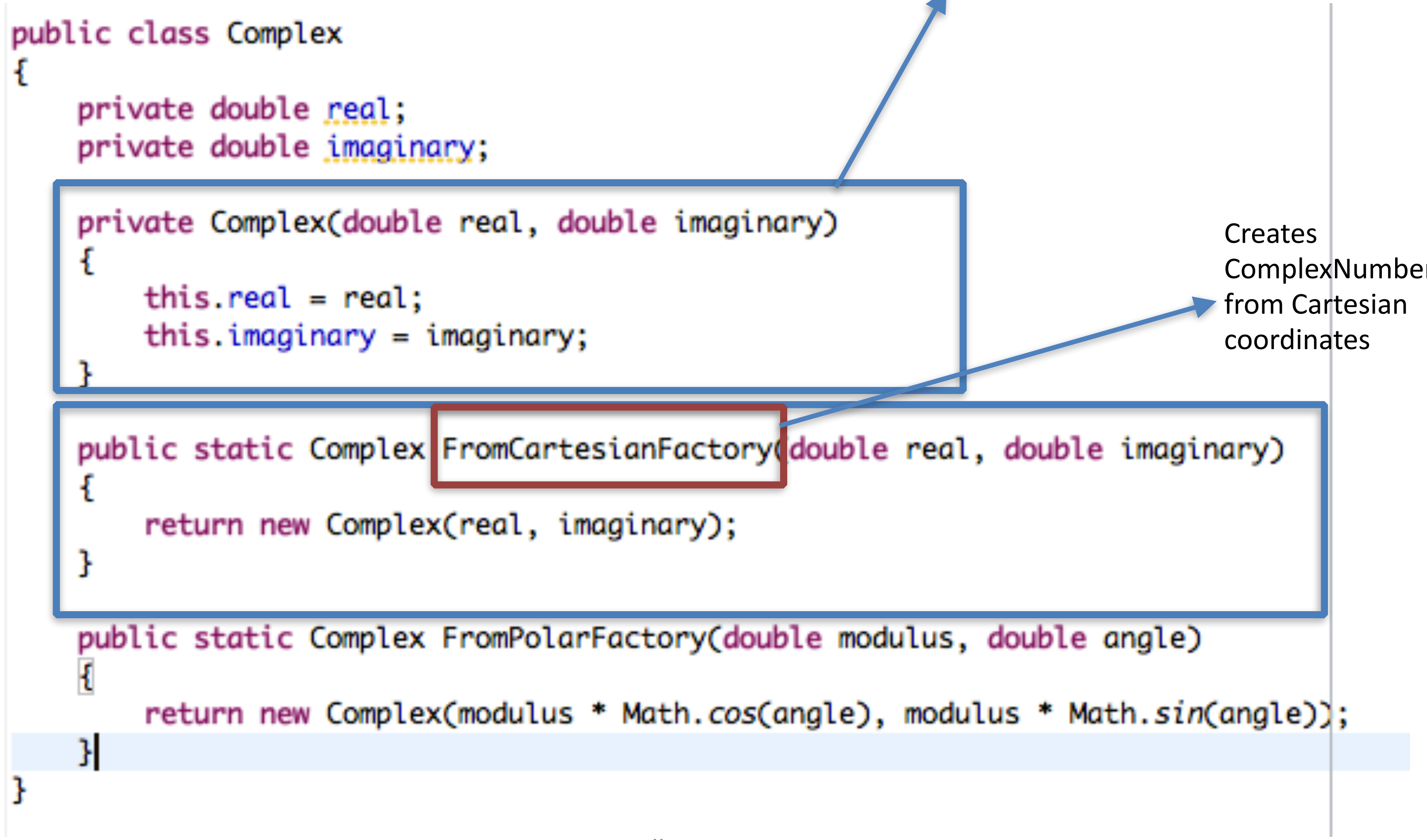
- 

```java
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Why have this constructor as private?

Creates ComplexNumber from Cartesian coordinates

Using Factory Methods.

- 

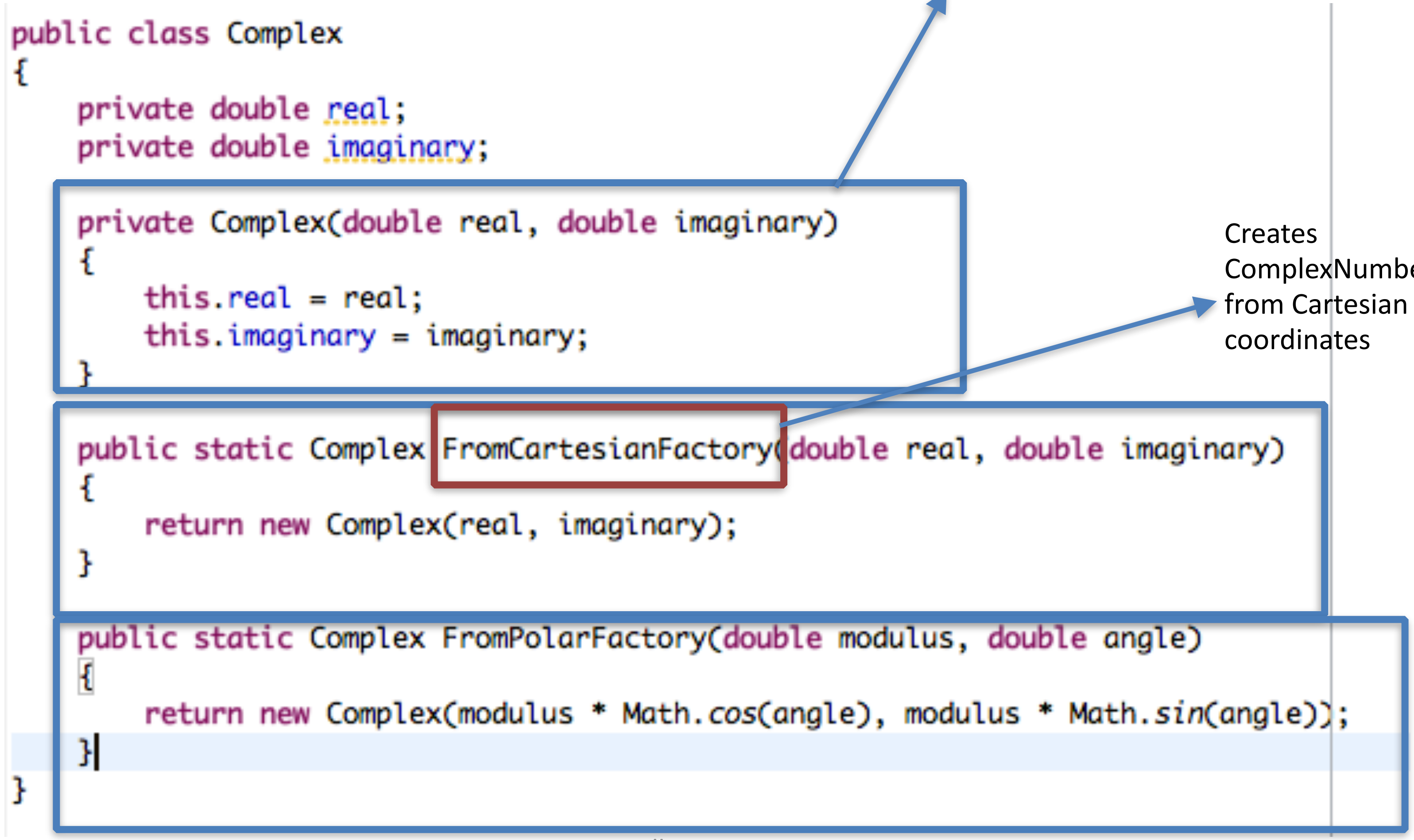
```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Why have this constructor as private?

Creates ComplexNumber from Cartesian coordinates

Using Factory Methods.

•



```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Why have this constructor as private?

Creates ComplexNumber from Cartesian coordinates

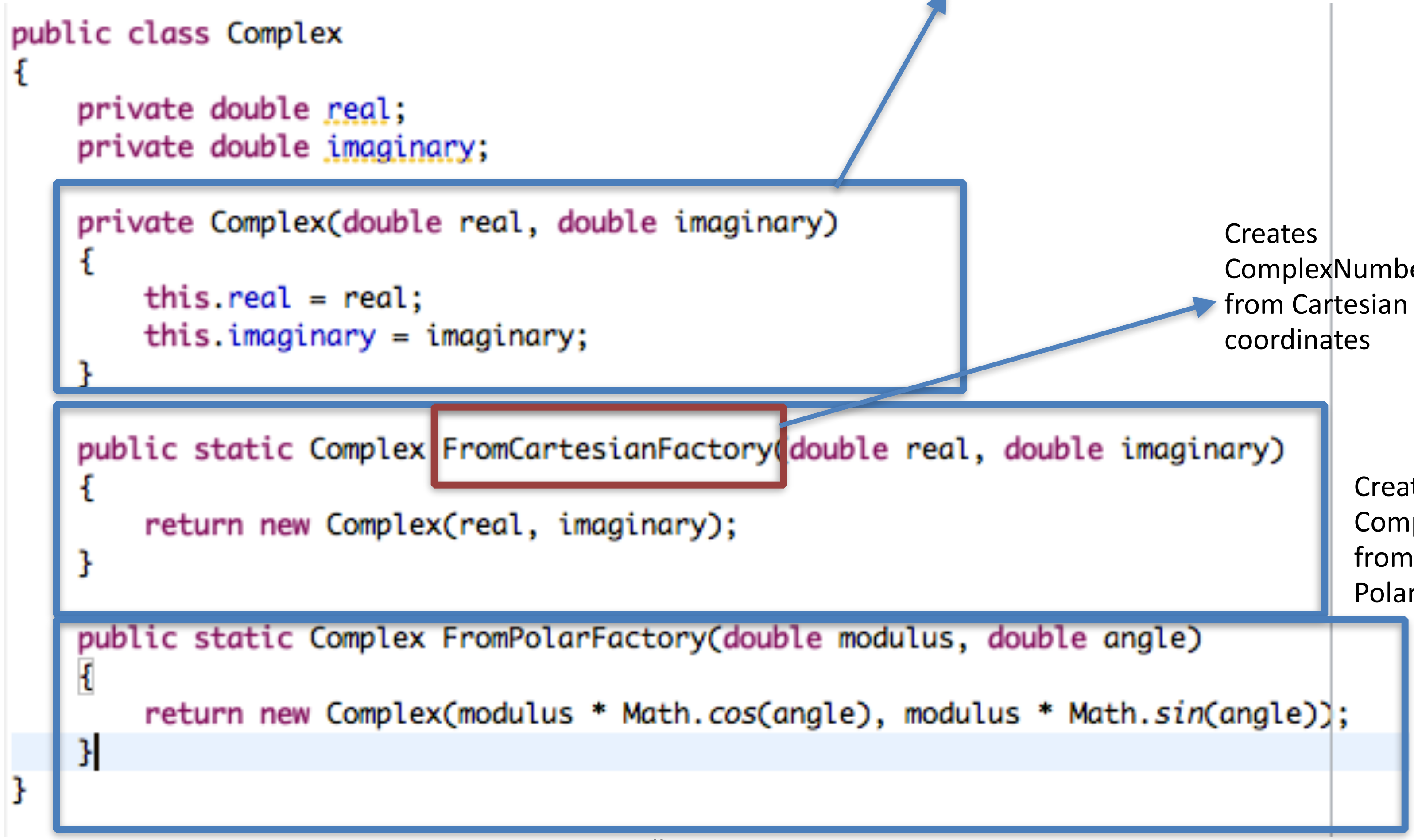
Using Factory Methods.

-

Why have this constructor as private?

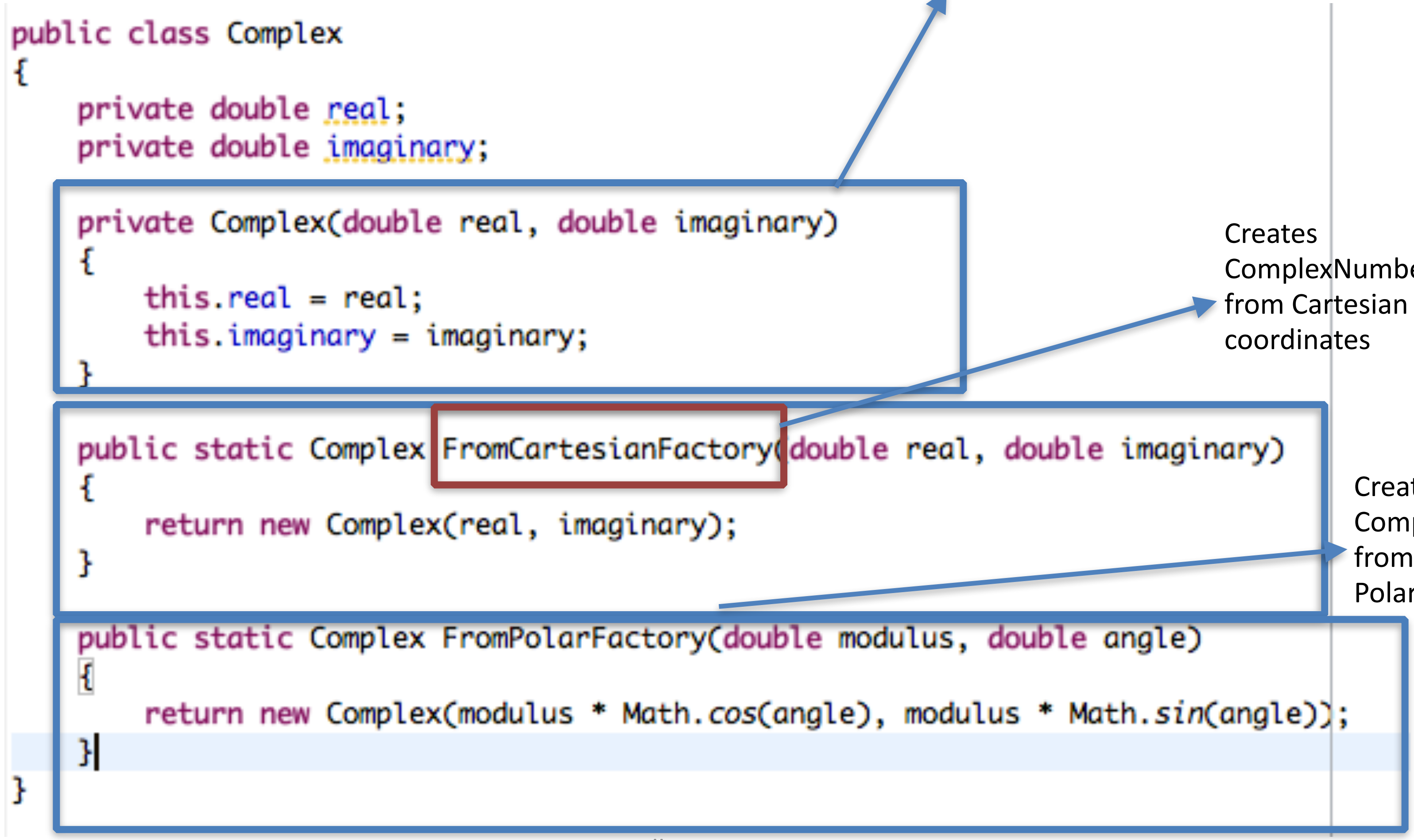
Creates ComplexNumber from Cartesian coordinates

Creates ComplexNumber from Polar coordinates



```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }
    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

Using Factory Methods.

- 

```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }

    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```

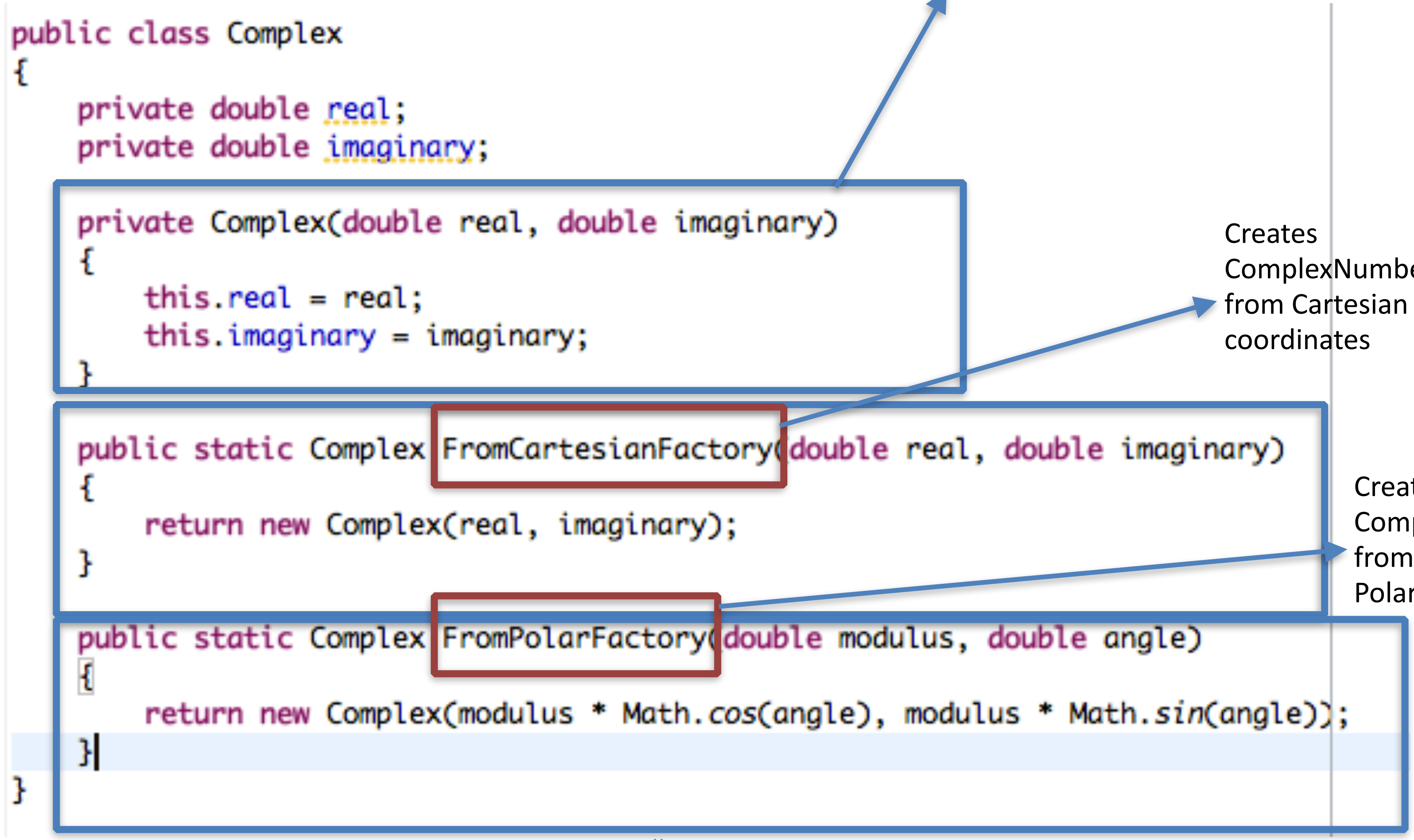
Why have this constructor as private?

Creates ComplexNumber from Cartesian coordinates

Creates ComplexNumber from Polar coordinates

Using Factory Methods.

•



Why have this constructor as private?

Creates ComplexNumber from Cartesian coordinates

Creates ComplexNumber from Polar coordinates

```
public class Complex
{
    private double real;
    private double imaginary;

    private Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex FromCartesianFactory(double real, double imaginary)
    {
        return new Complex(real, imaginary);
    }
    public static Complex FromPolarFactory(double modulus, double angle)
    {
        return new Complex(modulus * Math.cos(angle), modulus * Math.sin(angle));
    }
}
```