CSCI 132:
Basic Data Structures and Algorithms

Intro to Java (OOP, Methods, Control Flow)

Reese Pearsall & Iliana Castillon

Fall 2024

CSCI 132 TAs:

Section 1 and Section 3:
- **Sultan Yarylgassimov**
- sultanyaril@gmail.com
- Office Hours: Thursdays 12:00 – 2:00 PM in Barnard Hall 259

Section 2 and Section 4:
- **Fatima Ododo**
- fatima.ododo@student.montana.edu
- Office Hours: Tuesdays 11:00 AM – 1:00 PM in Barnard Hall 259
-

Announcements

- Lab 1 will be posted later this evening (?). We will discuss it on Wednesday
- Lab 1 is due this Thursday at 11:59 PM
- Do not rename your .java files

Adding Another Programming Language to my resume after learning how to write "Hello World" in it.

Java

In this class, we will use **Java** as our programming language

Why do we need more than one programming language?

```
public void processData() {
   do {
      int data = getData();

      if (data < 0)
         performOperation1(data);
      else
         performOperation2(data);
   } while (hasMoreData());
}
```

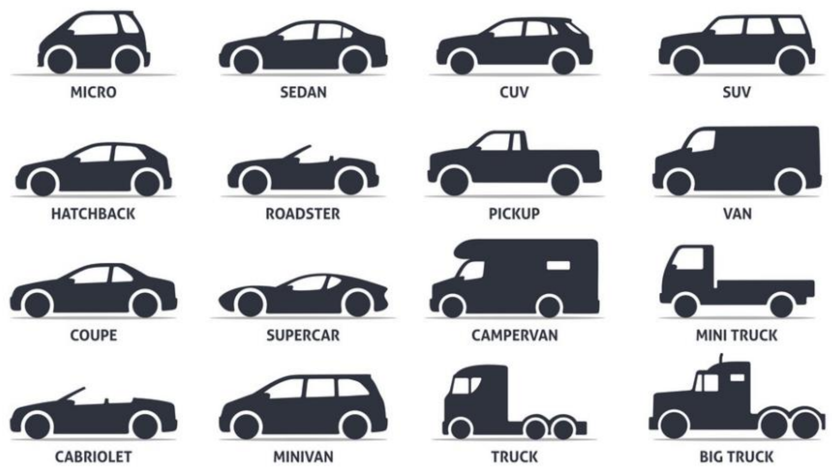
Java

In this class, we will use **Java** as our programming language

Why do we need more than one programming language?

```
public void processData() {
    do {
        int data = getData();

        if (data < 0)
            performOperation1(data);
        else
            performOperation2(data);
    } while (hasMoreData());
}
```

Different programming languages are better for different things



Java vs Python

Good for developing large, commercial, distributable software	Very flexible. Good for shorter jobs, data analysis, Web development,
Faster than Python	Slower than Java
OOP Language	Functional programming language
Verbose (sigh)	Simple (but requires whitespace)
Static Typed	Dynamic Typed

Object Oriented Programming

```python
class Student():

    def __init__(self,name,gpa,major):
        self.name = name
        self.gpa = gpa
        self.major = major

    def getName(self):
        return self.name

    def getGPA(self):
        return self.gpa

    def getMajor(self):
        return self.major
```

Object Oriented Programming

```
class Student():

    def __init__(self,name,gpa,major):
        self.name = name
        self.gpa = gpa
        self.major = major

    def getName(self):
        return self.name

    def getGPA(self):
        return self.gpa

    def getMajor(self):
        return self.major
```

We write **classes** that is a blueprint of something

Object Oriented Programming

```
class Student():

    def __init__(self,name,gpa,major):
        self.name = name
        self.gpa = gpa
        self.major = major

    def getName(self):
        return self.name

    def getGPA(self):
        return self.gpa

    def getMajor(self):
        return self.major
```

We write **classes** that is a blueprint of something

Classes consist of two important things:

1. Instance Fields/Attributes
2. Methods/Behaviors

Object Oriented Programming

```
class Student():

    def __init__(self,name,gpa,major):
        self.name = name
        self.gpa = gpa
        self.major = major

    def getName(self):
        return self.name

    def getGPA(self):
        return self.gpa

    def getMajor(self):
        return self.major
```

We write **classes** that is a blueprint of something

Classes consist of two important things:

1. Instance Fields/Attributes
2. Methods/Behaviors

This program does nothing until we start **creating objects**

Object Oriented Programming

```
class Student():

    def __init__(self,name,gpa,major):
        self.name = name
        self.gpa = gpa
        self.major = major

    def getName(self):
        return self.name

    def getGPA(self):
        return self.gpa

    def getMajor(self):
        return self.major

student1 = Student("Reese", 4.0, "Computer Science")

student2 = Student("Susan", 3.5, "Chemistry")
```

`student1` and `student2` are instances of the `Student` class.

Object Oriented Programming

```
class Student():
                       1.   2.   3.
    def __init__(self,name,gpa,major):
        self.name = name
        self.gpa = gpa
        self.major = major

    def getName(self):
        return self.name

    def getGPA(self):
        return self.gpa

    def getMajor(self):
        return self.major
                       1.     2.           3.
student1 = Student("Reese", 4.0, "Computer Science")
                       1.     2.       3.
student2 = Student("Susan", 3.5, "Chemistry")
```

`student1` and `student2` are instances of the `Student` class.

To create an object, we called the class name, and then pass the necessary **parameters/arguments**

This triggers the constructor, which will *create* our objects

Object Oriented Programming

```
class Student():
                                1.     2.     3.
    def __init__(self,name,gpa,major):
        self.name = name
        self.gpa = gpa
        self.major = major

    def getName(self):
        return self.name

    def getGPA(self):
        return self.gpa

    def getMajor(self):
        return self.major
```

`student1` and `student2` are instances of the `Student` class.

An object is an encapsulation of information…

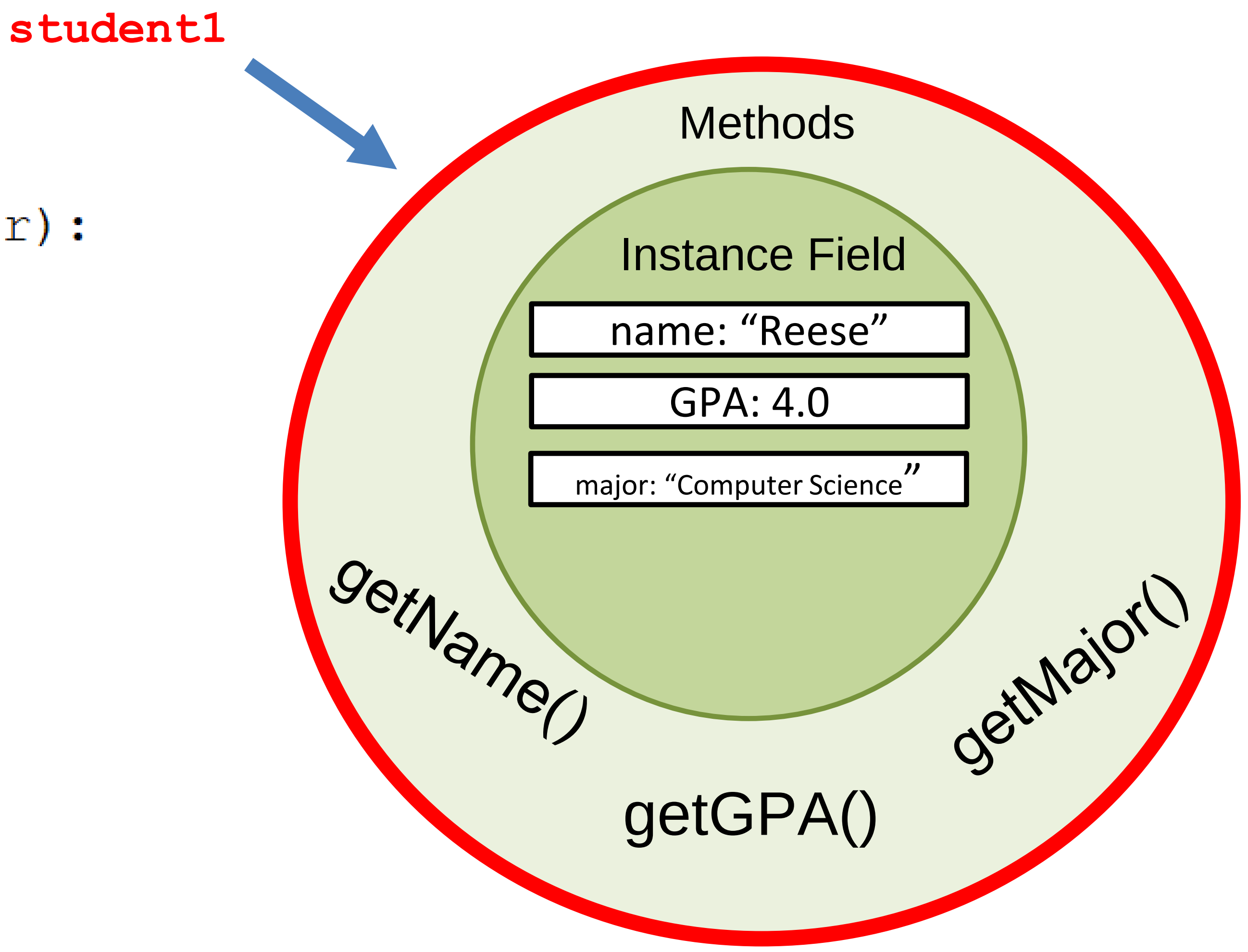
```
print(student1)
<__main__.Student object at 0x000002010BD0E0D0>
```

Printing/accessing an object doesn't do much on its own…

```
                         1.         2.                3.
student1 = Student("Reese", 4.0, "Computer Science")
                         1.         2.         3.
student2 = Student("Susan", 3.5, "Chemistry")
```

Object Oriented Programming

```
class Student():

    def __init__(self,name,gpa,major):
        self.name = name
        self.gpa = gpa
        self.major = major

    def getName(self):
        return self.name

    def getGPA(self):
        return self.gpa

    def getMajor(self):
        return self.major

 student1 = Student("Reese", 4.0, "Computer Science")

 student2 = Student("Susan", 3.5, "Chemistry")
```



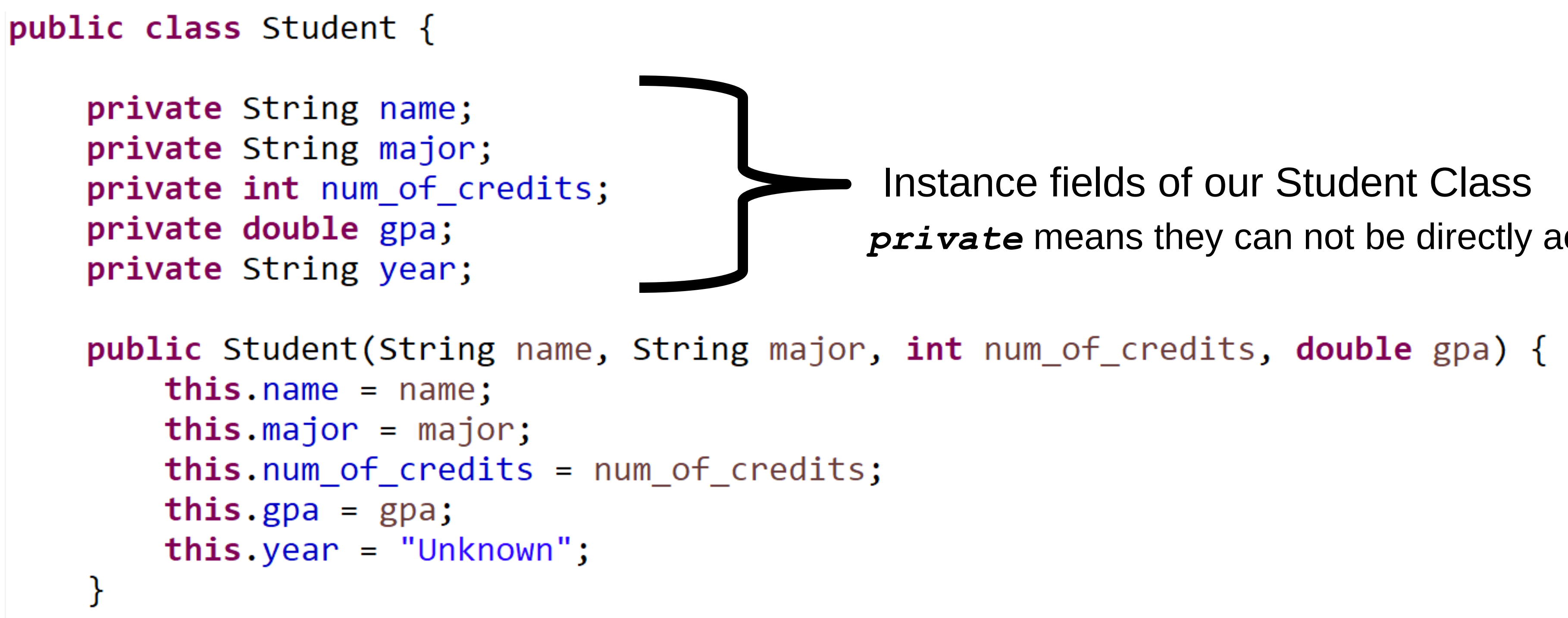
Object Oriented Programming

```python
class Student():

    def __init__(self,name,gpa,major):
        self.name = name
        self.gpa = gpa
        self.major = major

    def getName(self):
        return self.name

    def getGPA(self):
        return self.gpa

    def getMajor(self):
        return self.major


student1 = Student("Reese", 4.0, "Computer Science")

student2 = Student("Susan", 3.5, "Chemistry")
```

```java
public class Student {

    String name;
    double GPA;
    String major;

    public Student(String name, double GPA, String major){
        this.name = name;
        this.GPA = GPA;
        this.major = major;
    }

    public String getName(){
        return this.name;
    }

    public double getGPA(){
        return this.GPA;
    }

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

```java
Student student1 = new Student("Reese",4.0,"Computer Science");

Student student2 = new Student("Susan",3.5,"Chemistry");
```

Java is only OOP,
all our code will be going inside of a class

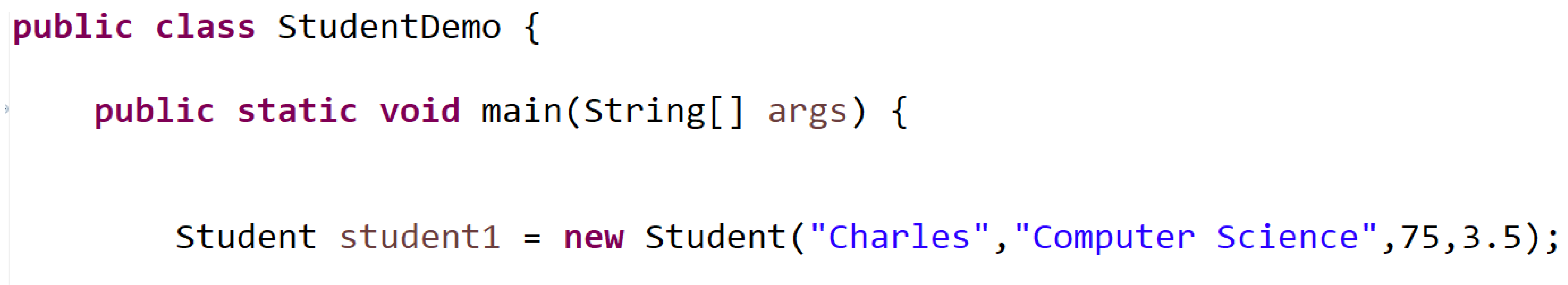


```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

Instance fields of our Student Class

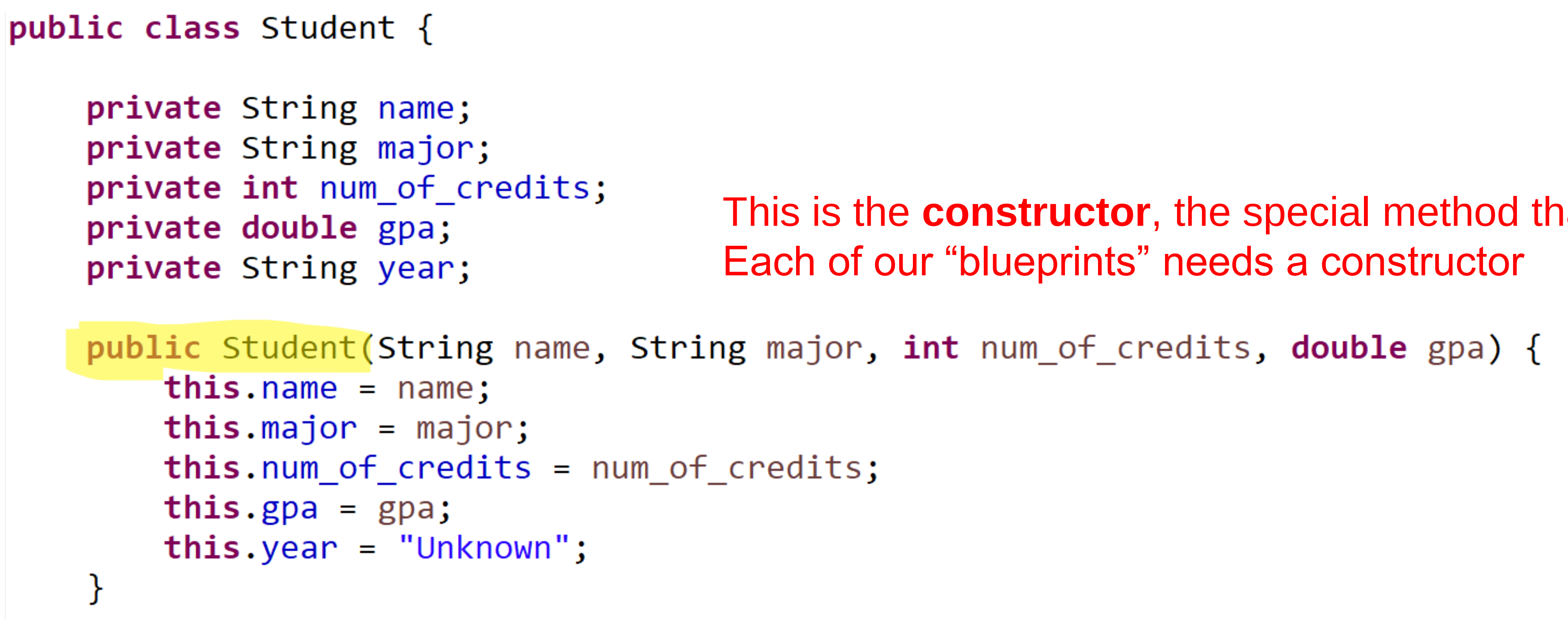
private means they can not be directly accessed outside of the class

Student.Java



```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

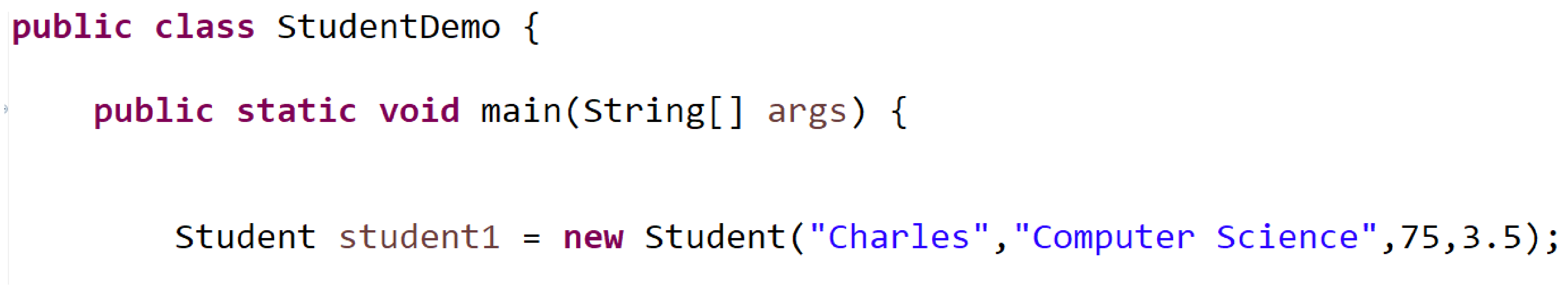
StudentDemo.Java



```
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

This is the **constructor**, the special method that creates our objects
Each of our “blueprints” needs a constructor

Student.Java



```
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

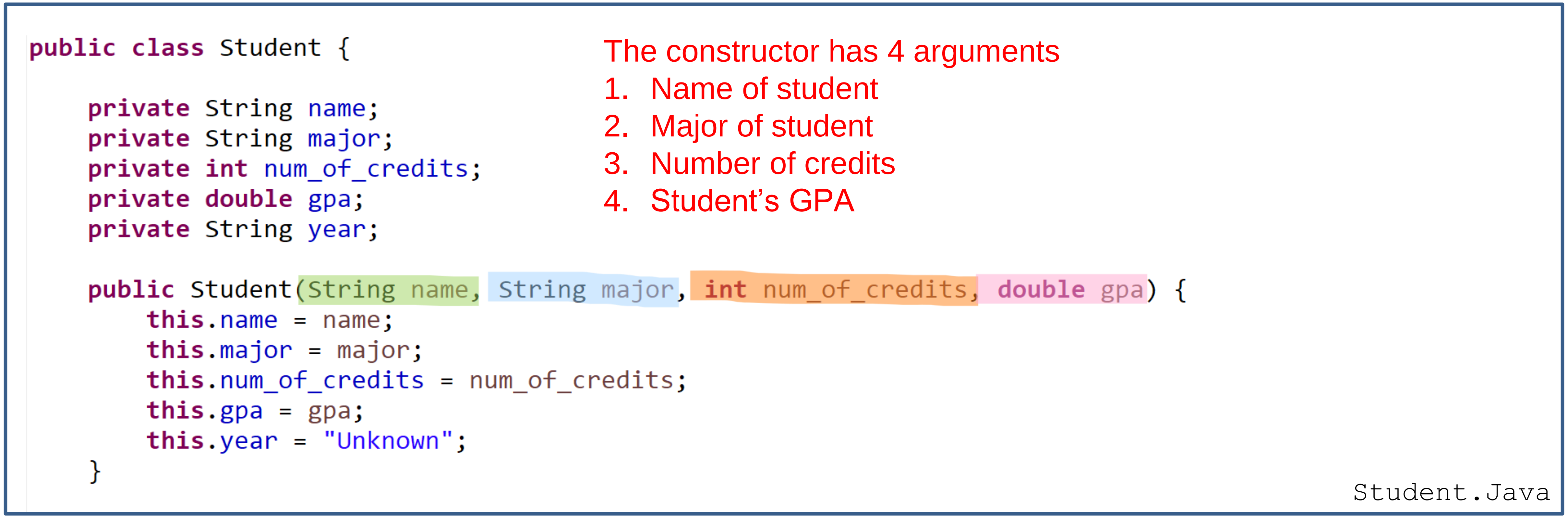
```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

When we use the **new** keyword, it will invoke our constructor

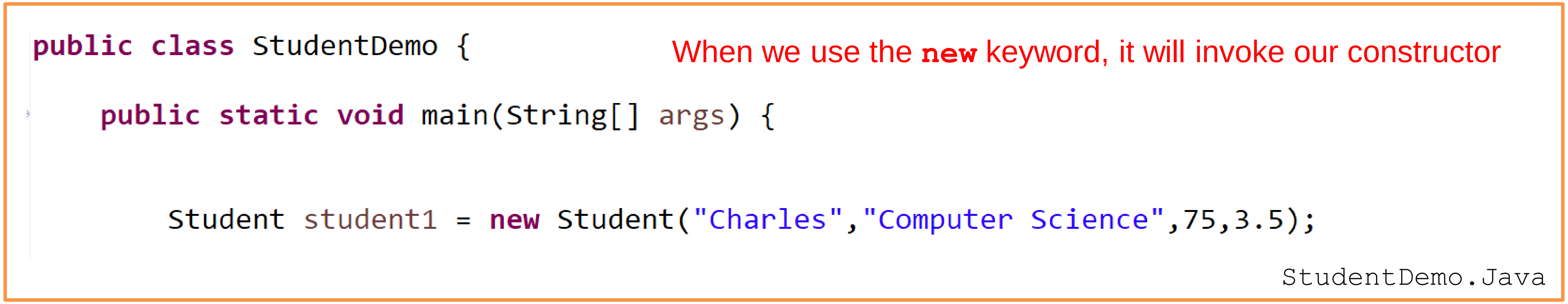


```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

Student.Java

The constructor has 4 arguments

1. Name of student
2. Major of student
3. Number of credits
4. Student’s GPA



```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

When we use the **`new`** keyword, it will invoke our constructor

```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

The constructor has 4 arguments

1. Name of student
2. Major of student
3. Number of credits
4. Student's GPA

Whenever we create a new Student object with **new**, we must make sure we pass in these 4 values

Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

When we use the **new** keyword, it will invoke our constructor

StudentDemo.Java

```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

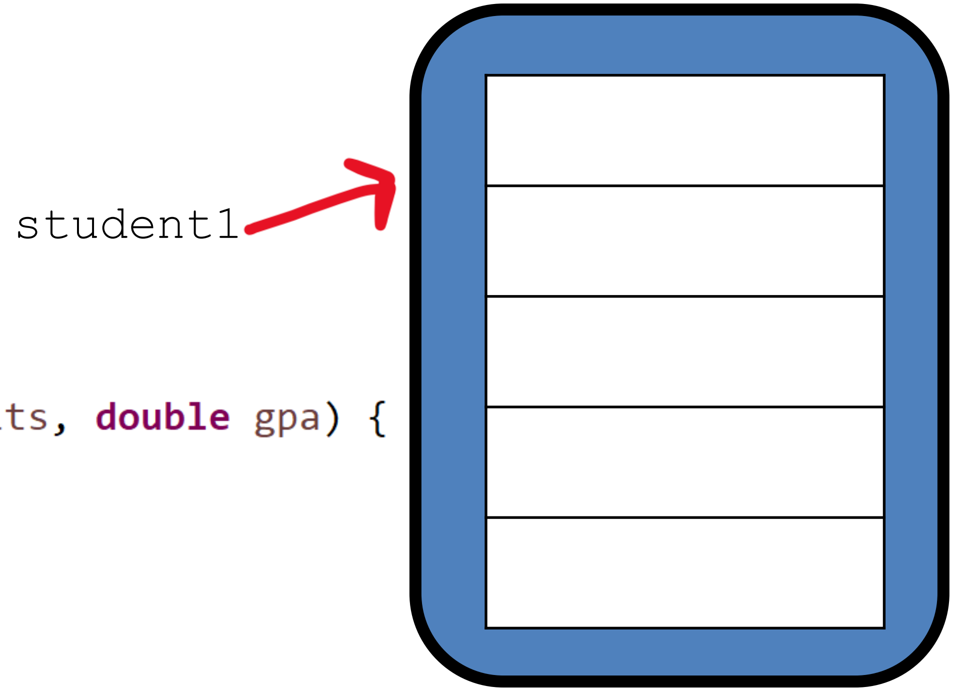
```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

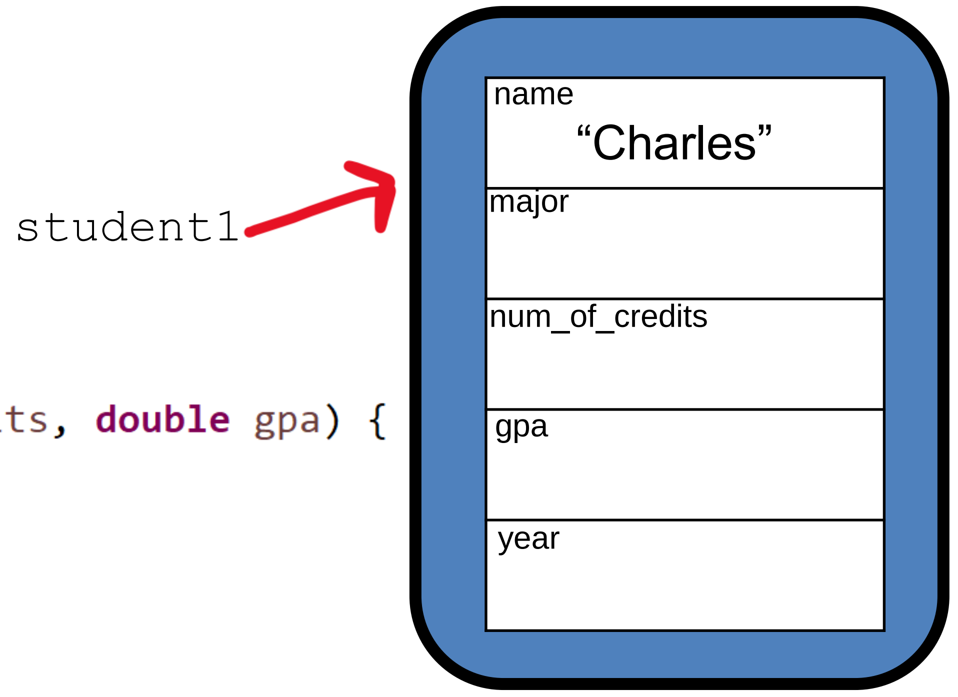


Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

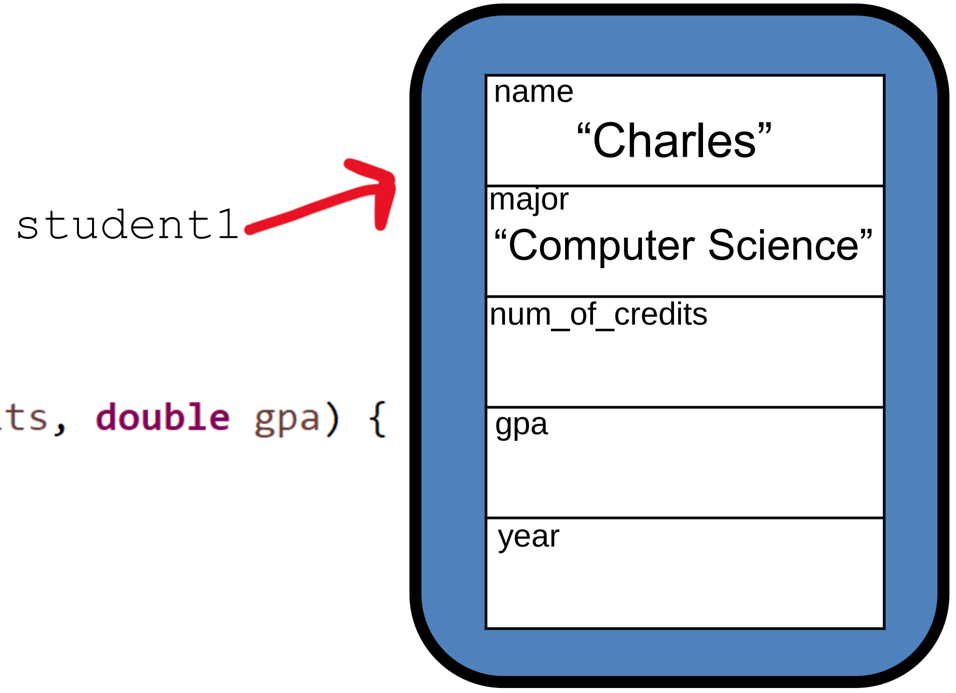


Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

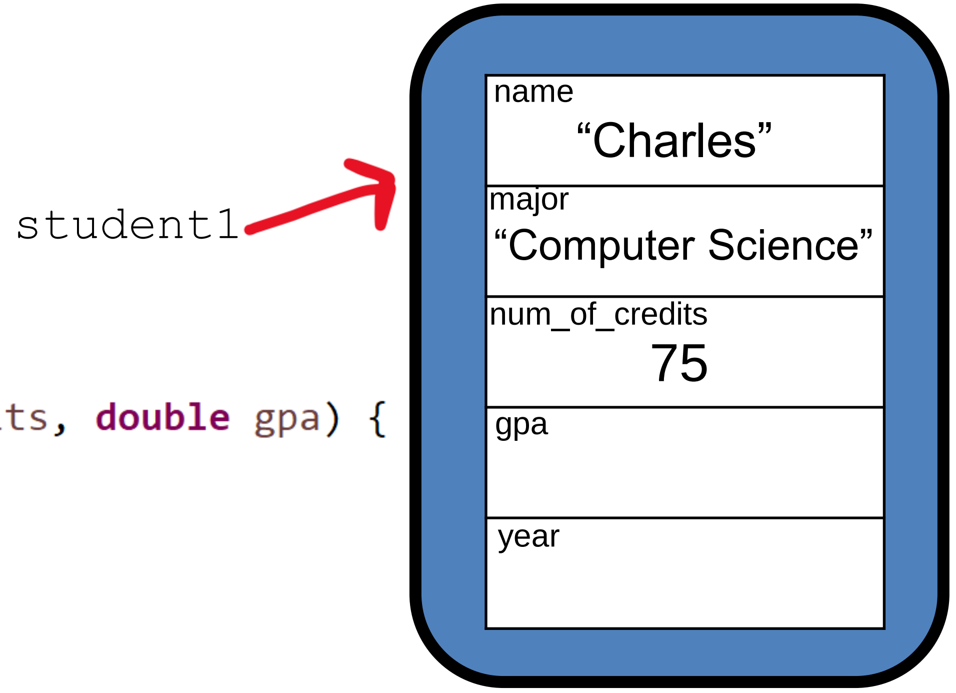


Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

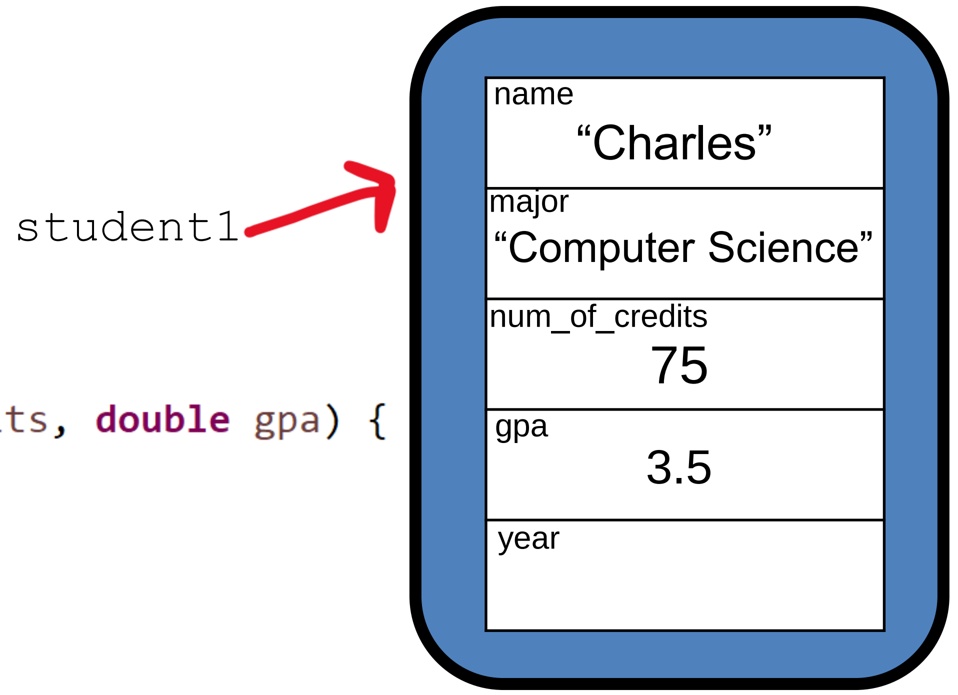


Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

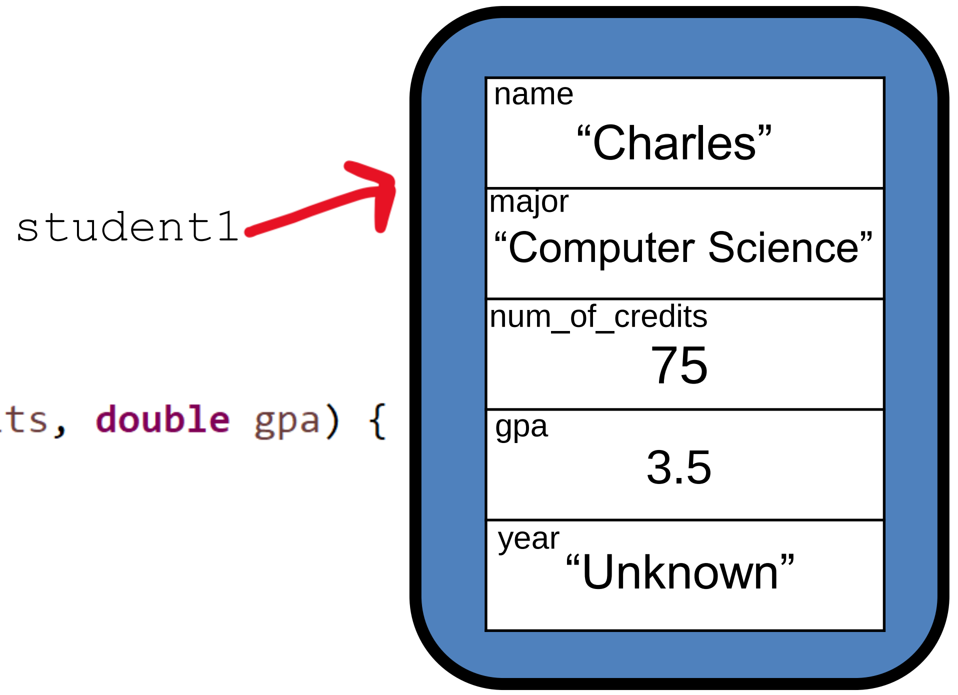


Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

```java
public class Student {

    private String name;
    private String major;
    private int num_of_credits;
    private double gpa;
    private String year;

    public Student(String name, String major, int num_of_credits, double gpa) {
        this.name = name;
        this.major = major;
        this.num_of_credits = num_of_credits;
        this.gpa = gpa;
        this.year = "Unknown";
    }
```

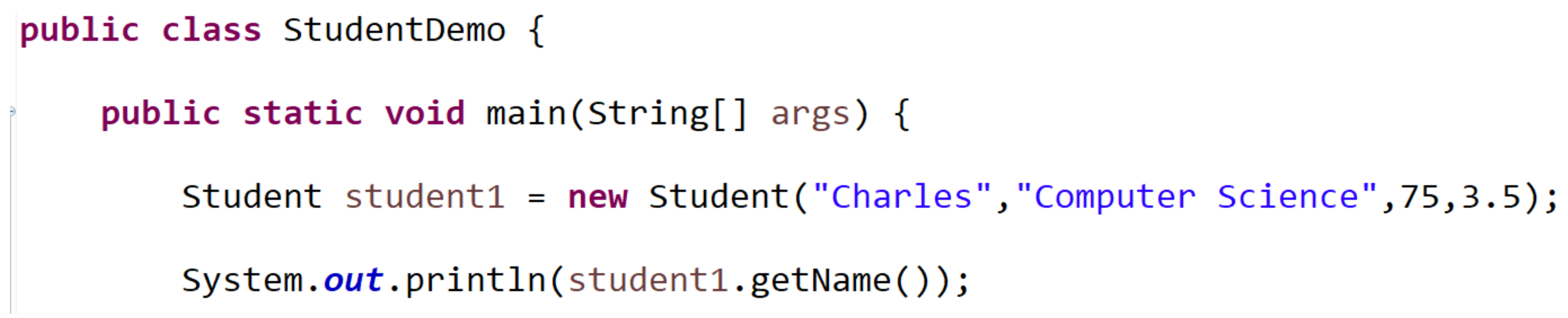


Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {


        Student student1 = new Student("Charles","Computer Science",75,3.5);
```

StudentDemo.Java

Let’s add a function (a **method**) that will get a Student’s name

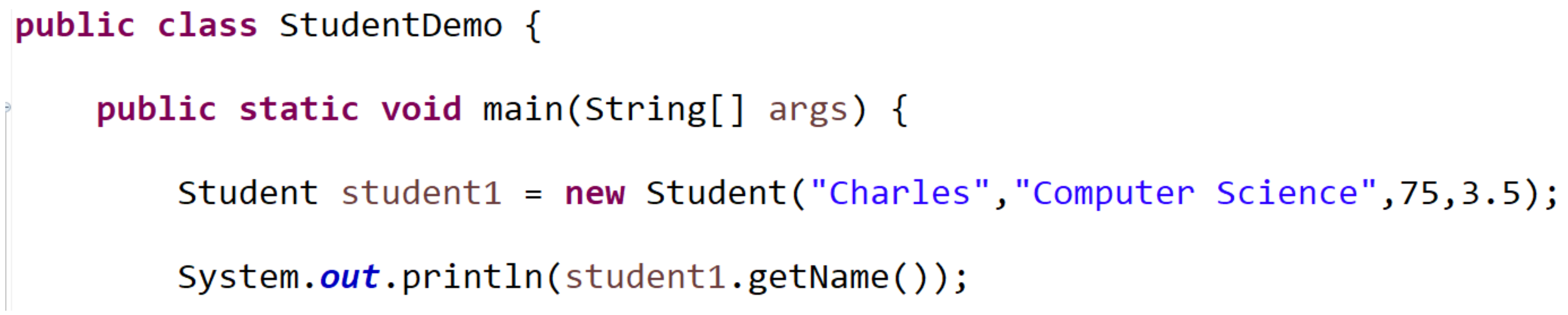


```
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);

        System.out.println(student1.getName());
```

StudentDemo.Java

Let’s add a function (a **method**) that will get a Student’s name

- We called this method on a Student object (student1.getName())
- So, our function needs to belong in our Student class (Student.Java)



```
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);

        System.out.println(student1.getName());
```

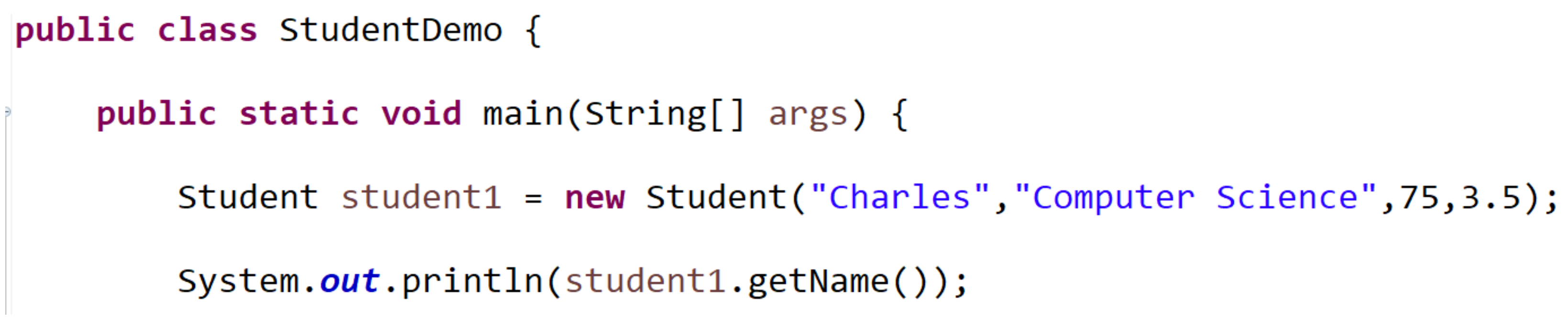
StudentDemo.Java

Let’s add a function (a **method**) that will get a Student’s name

- We called this method on a Student object (student1.getName())
- So, our function needs to belong in our Student class (Student.Java)

What should this function take as input? What should this function output?

- Input: a Student object
- Output: the name of a student (String)



```
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);

        System.out.println(student1.getName());
```

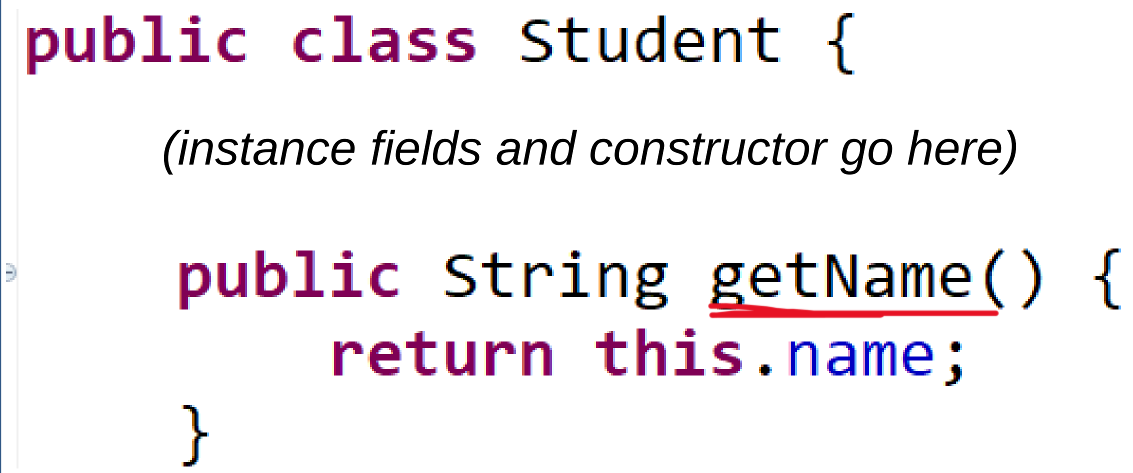
StudentDemo.Java

```java
public class Student {

    (instance fields and constructor go here)

    public String getName() {
        return this.name;
    }
```

Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);

        System.out.println(student1.getName());
```

StudentDemo.Java



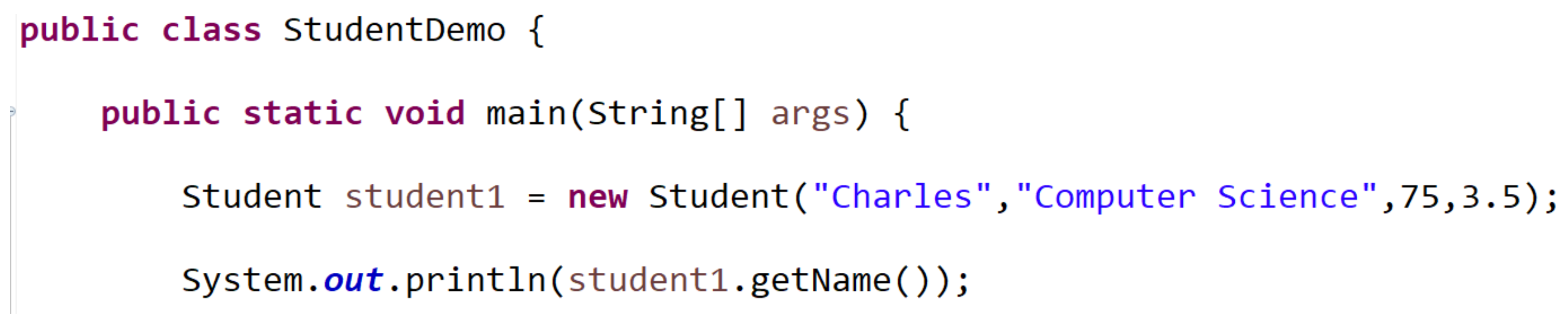
```
public class Student {
```

(instance fields and constructor go here)

```
    public String getName() {
        return this.name;
    }
```

Name of method

Student.Java



```
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);

        System.out.println(student1.getName());
```

StudentDemo.Java

```java
public class Student {

    (instance fields and constructor go here)

    public String getName() {
        return this.name;
    }
```

Name of method

When we define methods in Java, we must declare the *data type* that the method will return

This method returns a String

Student.Java

```java
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);

        System.out.println(student1.getName());
```

StudentDemo.Java

```
public class Student {
```

(instance fields and constructor go here)

```
    public String getName() {
        return this.name;
    }
```

Name of method

This method returns a String

This method is public (other classes can use it)

(Generally, all methods will be public ☺)

Student.Java

```
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);

        System.out.println(student1.getName());
```

StudentDemo.Java

```
public class Student {

        (instance fields and constructor go here)

    public String getName() {
        return this.name;
    }
```

Name of method

This method returns a String

This method is public (other classes can use it)

The **this** keyword refers to the *object* that this method was called on (student1)

(return student1's name attribute)

Student.Java

```
public class StudentDemo {

    public static void main(String[] args) {

        Student student1 = new Student("Charles","Computer Science",75,3.5);

        System.out.println(student1.getName());
```

StudentDemo.Java

```java
public void printStudentSummary() {
    System.out.println("Name: " + this.name);
    System.out.println("Major: " + this.major);
    System.out.println("Name: " + this.num_of_credits);
    System.out.println("GPA: " + this.gpa);
    System.out.println("Year: " + this.year);
}
```

Here is a method that doesn't return anything

`void` is used to indicate that a method will not return anything

Student.Java

```java
public static void main(String[] args) {

    Student student1 = new Student("Charles","Computer Science",75,3.5);
    student1.printStudentSummary();
```

StudentDemo.Java

```java
public void changeMajor(String newMajor) {
    this.major = newMajor;
}
```

Here is method to change a Student's major. When we call this method, we pass in the Student's new major as an argument

So when we define this method, we need to make sure it accepts one argument

Student.Java

```java
public static void main(String[] args) {

    Student student1 = new Student("Charles","Computer Science",75,3.5);

    student1.changeMajor("Math");
```

StudentDemo.Java

```java
public void checkForProbation() {

    if(this.gpa >= 2.0){
        System.out.print("student is in good standing");
    }
    else {
        System.out.println("Student: "+ this.name + " needs to go on academic probation");
    }

}
```

If statements can be used to check a condition.

- If the condition is true, execute the code in the body of the if statement
- If it is false, proceed to the `else` statement

Student.Java

```java
student1.checkForProbation();
```

StudentDemo.Java

```java
public void determineYear() {
    if(this.num_of_credits <= 30) {
        this.year = "Freshman";
    }
    else if(this.num_of_credits > 30 && this.num_of_credits <= 60) {
        this.year = "Sophomore";
    }
    else if(this.num_of_credits > 60 && this.num_of_credits <= 90) {
        this.year = "Junior";
    }
    else if(this.num_of_credits > 90 && this.num_of_credits <= 120) {
        this.year = "Senior";
    }
    else {
        this.year = "???";
    }
}
```

We can check multiple conditions using the and operator (**&&**)

(we do not have the `and` keyword in Java)

Student.Java

```java
student1.determineYear();
```

StudentDemo.Java

Example: A student is allowed to register for CSCI 476 if they have a GPA greater than 2.0, **and** if they are a Junior **or** Senior

```
public void allowToRegister() {

    if (this.gpa > 2.0) { // check the first condition (Alternatively, we could use an && here)

        if (this.year.equals("Junior") || this.year.equals("Senior")){

            System.out.println("Student is allowed to register for CSCI 476");

        }
    }
}
```

Student.Java

We can check one of two conditions is true using the or operator (**||**)

(we do not have the `or` keyword in Java)

```
student1.determineYear();
```

StudentDemo.Java

Example: A student is allowed to register for CSCI 476 if they have a GPA greater than 2.0, **and** if they are a Junior **or** Senior

```
public void allowToRegister() {

    if (this.gpa > 2.0) { // check the first condition (Alternatively, we could use an && here)

        if (this.year.equals("Junior") || this.year.equals("Senior")){

            System.out.println("Student is allowed to register for CSCI 476");

        }
    }
}
```

Student.Java

Why do `this.year.equals("Junior")` and not `this.year == "Junior“`

Checking for string equality in Java is a little bit funky…

Using == does **not** check for equivalence of values between two strings…

Example: A student is allowed to register for CSCI 476 if they have a GPA greater than 2.0, **and** if they are a Junior **or** Senior

```java
public void allowToRegister() {

    if (this.gpa > 2.0) { // check the first condition (Alternatively, we could use an && here)

        if (this.year.equals("Junior") || this.year.equals("Senior")){

            System.out.println("Student is allowed to register for CSCI 476");

        }
    }
}
```

Student.Java

Why do `this.year.equals("Junior")` and not `this.year == "Junior“`

Checking for string equality in Java is a little bit funky…

Using == does **not** check for equivalence of values between two strings…

Instead, we need to use the **.equals()** method between two string

Do not use **==** to compare two strings