

CSCI 132:

Basic Data Structures and Algorithms

Lecture 4: Intro to Java (OOP, Methods, Control Flow)

Reese Pearsall
Fall 2023

Announcements

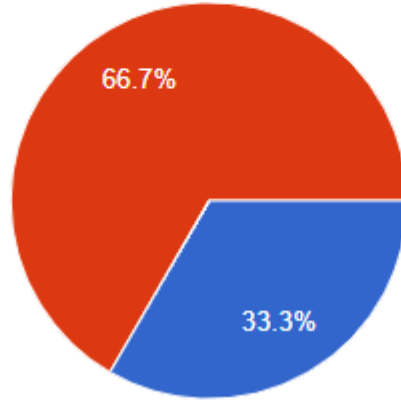
- Lab 1 will be posted later this evening. We will discuss it on Wednesday
- Lab 1 is due this Thursday at 11:59 PM
- Do not rename your .java files
- Course Questionnaire Results



Course Questionnaire

Have you programmed in Java before?

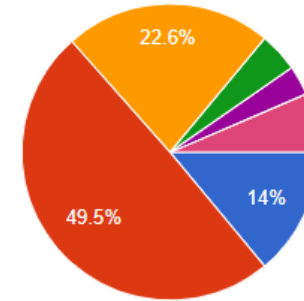
93 responses



● Yes
● No

Class:

93 responses



● Freshman
● Sophomore
● Junior
● Senior
● Graduate Student
● High School Student
● Other

About 50% of the class are CS majors. Lots of other majors in this class!

Very Hardcore
Procrastinator

Can't keep up
with CS majors

I work full
time

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

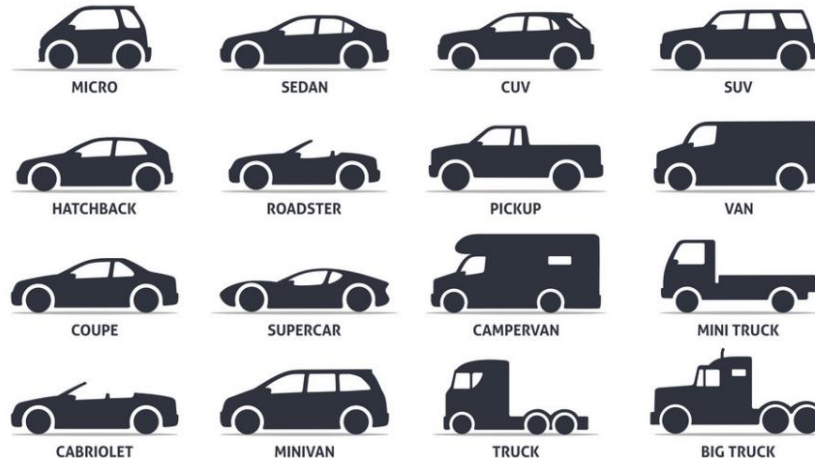


In this class, we will use **Java** as our programming language

Why do we need more than one programming language?

Different programming languages are better for different things

```
public void processData() {  
    do {  
        int data = getData();  
  
        if (data < 0)  
            performOperation1(data);  
        else  
            performOperation2(data);  
    } while (hasMoreData());  
}
```



Java vs Python



Good for developing large, commercial, distributable software

Faster than Python

OOP Language

Verbose (sigh)

Static Typed

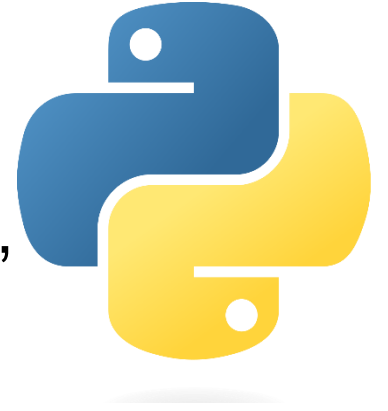
Very flexible. Good for shorter jobs, data analysis, Web development,

Slower than Java

Functional programming language

Simple (but requires whitespace)

Dynamic Typed



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
```

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
```

student1 and student2 are instances of the Student class.

```
    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

```
class Student():
```

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

student1 and student2 are instances of the Student class.

```
    def getName(self):  
        return self.name
```

```
    def getGPA(self):  
        return self.gpa
```

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

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

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

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

```
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")
```

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...

Object Oriented Programming

student1

```
class Student():
```

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

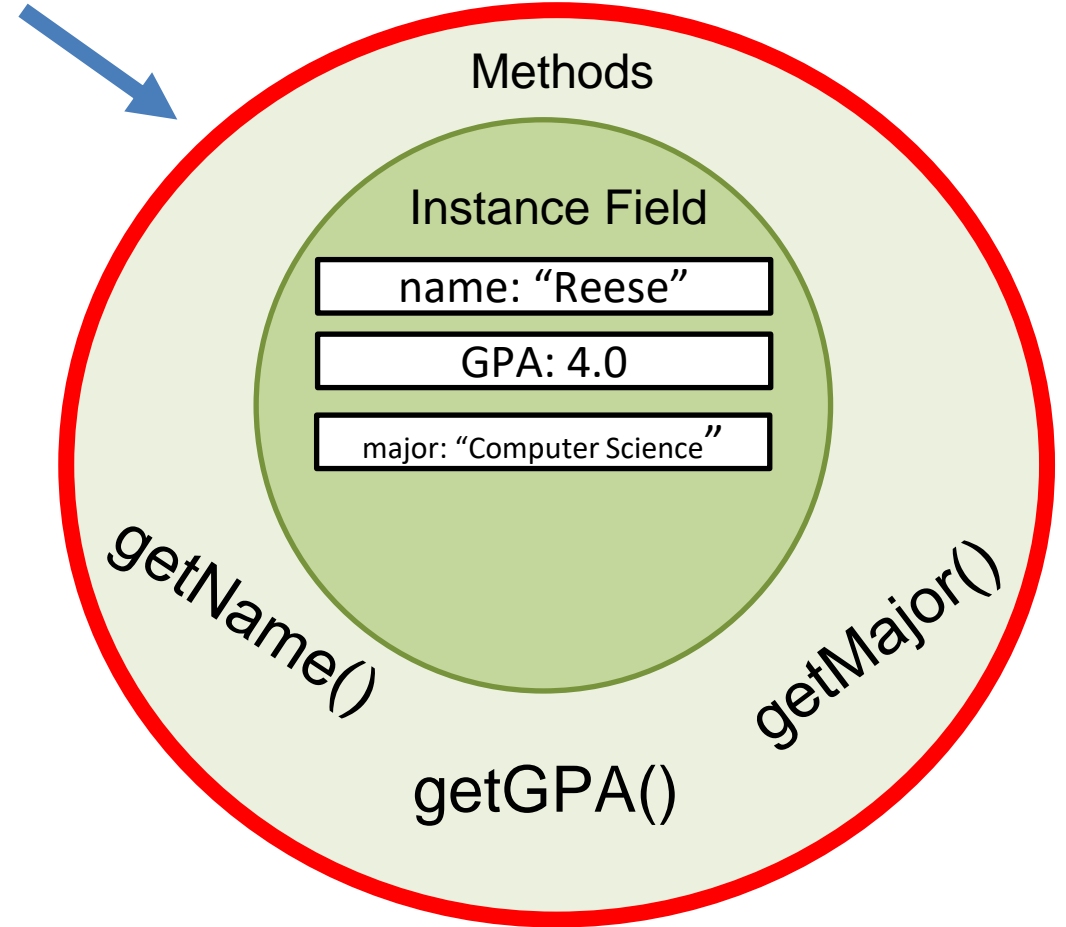
```
    def getName(self):  
        return self.name
```

```
    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

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

```
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;  
    }
```

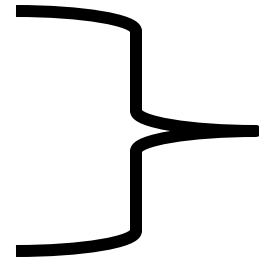
```
    public String getMajor(){  
        return this.major;  
    }
```

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

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

```
public class Student {
```

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



Instance fields of our Student Class

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

```
    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

```
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;
```

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

```
    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

```
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";  
    }  
}
```

Student.Java

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

```

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

```
        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";  
    }
```

The constructor has 4 arguments

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

Student.Java

```
public class StudentDemo {
```

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

```
    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";  
    }  
}
```

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

```
public class StudentDemo {
```

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

```
    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";  
    }  
}
```

Student.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";  
    }  
}
```


Student.Java

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

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

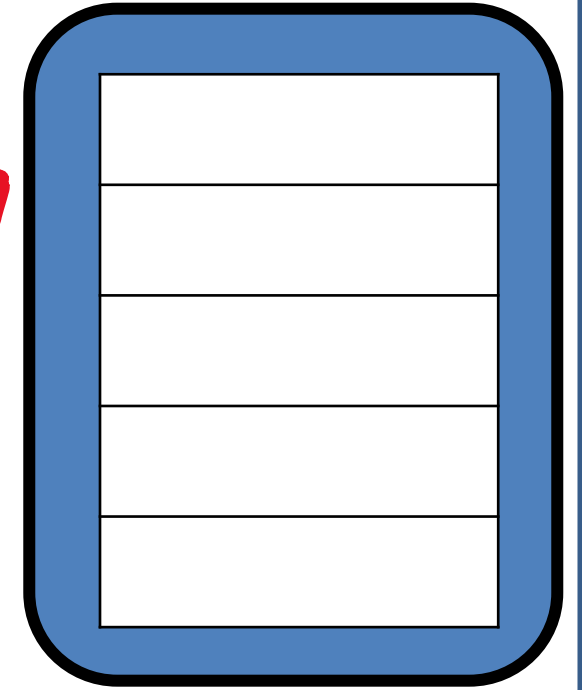
StudentDemo.Java

```
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";  
    }  
}
```

student1



Student.Java

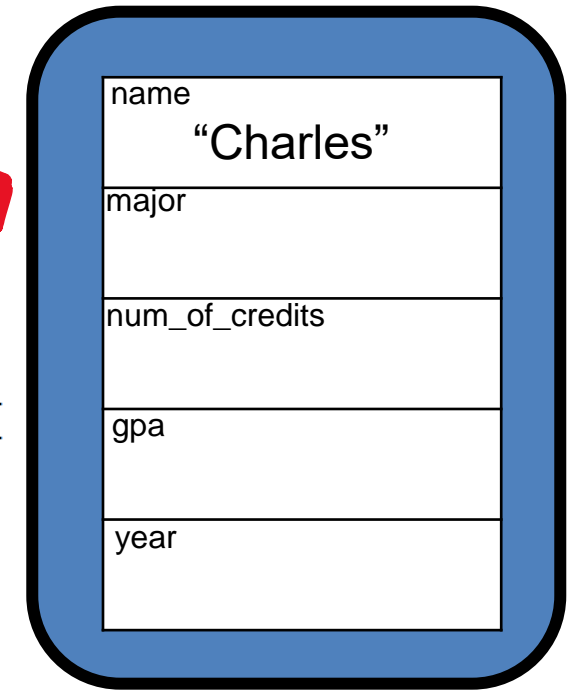
```
public class StudentDemo {  
  
    public static void main(String[] args) {
```

```
        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";  
    }  
}
```

student1



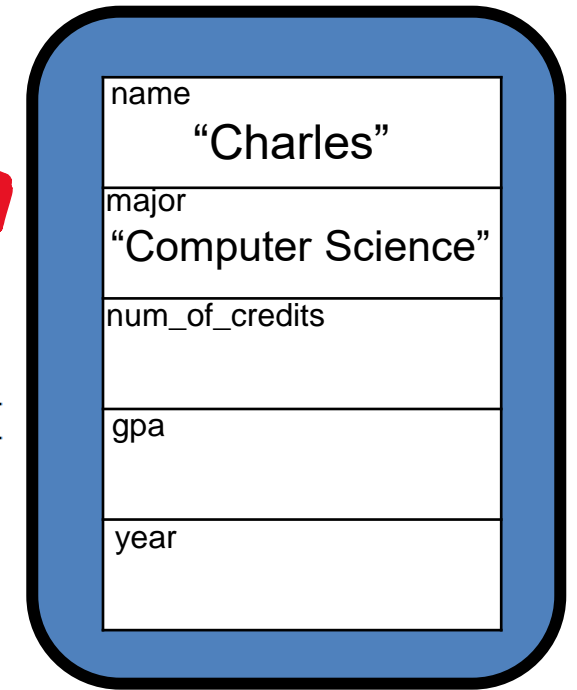
Student.Java

```
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";  
    }  
}
```

student1



Student.Java

```
public class StudentDemo {  
  
    public static void main(String[] args) {  
  
        Student student1 = new Student("Charles", "Computer Science", 75, 3.5);  
    }  
}
```

StudentDemo.Java

```

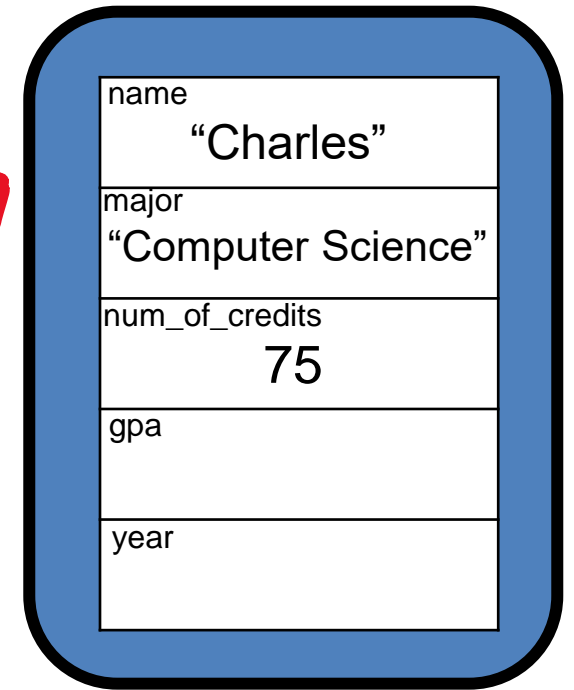
public class Student {

    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";
    }
}

```

student1



Student.Java

```

public class StudentDemo {

    public static void main(String[] args) {

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

```

StudentDemo.Java

```

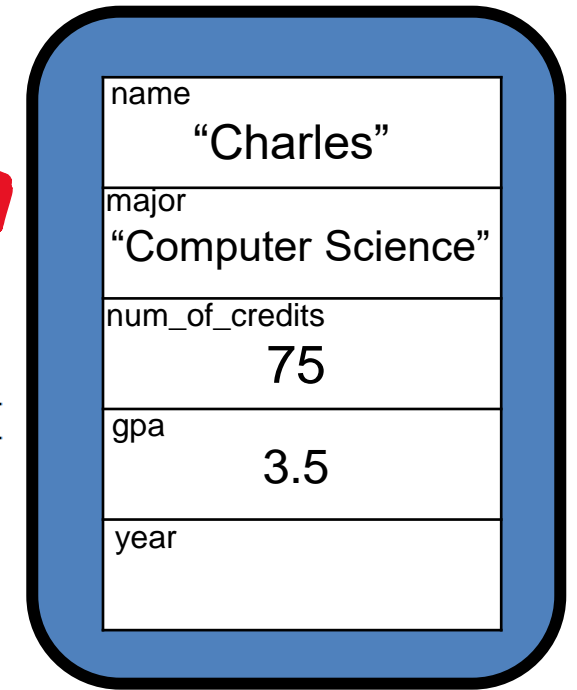
public class Student {

    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";
    }
}

```

student1



Student.Java

```

public class StudentDemo {

    public static void main(String[] args) {

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

```

StudentDemo.Java

```

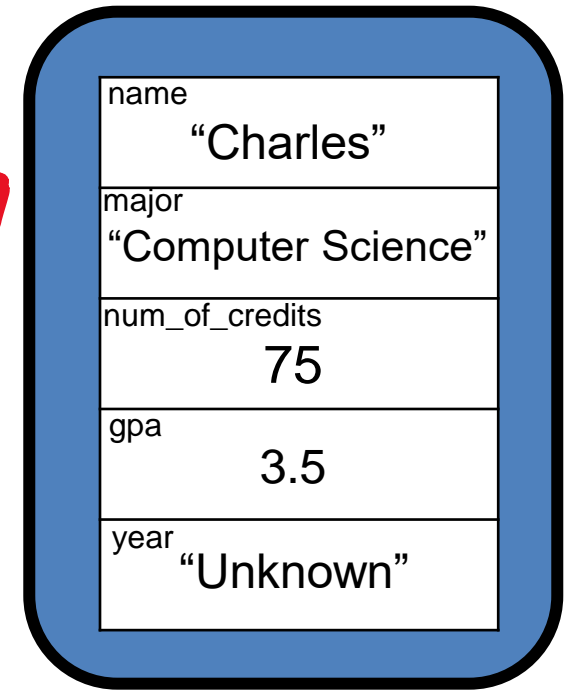
public class Student {

    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";
    }
}

```

student1



Student.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

```
public class Student {
```

(instance fields and constructor go here)

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

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

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

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

This method returns a String

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)

(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)

Name of method

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

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

```
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

```
public static void main(String[] args) {  
  
    Student student1 = new Student("Charles", "Computer Science", 75, 3.5);  
    student1.printStudentSummary();  
}
```

StudentDemo.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

```
public static void main(String[] args) {  
    Student student1 = new Student("Charles", "Computer Science", 75, 3.5);  
    student1.changeMajor("Math");  
}
```

StudentDemo.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

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

StudentDemo.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

```
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");  
        }  
    }  
}
```

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

Student.Java

(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

```
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");  
  
        }  
  
    }  
  
}
```

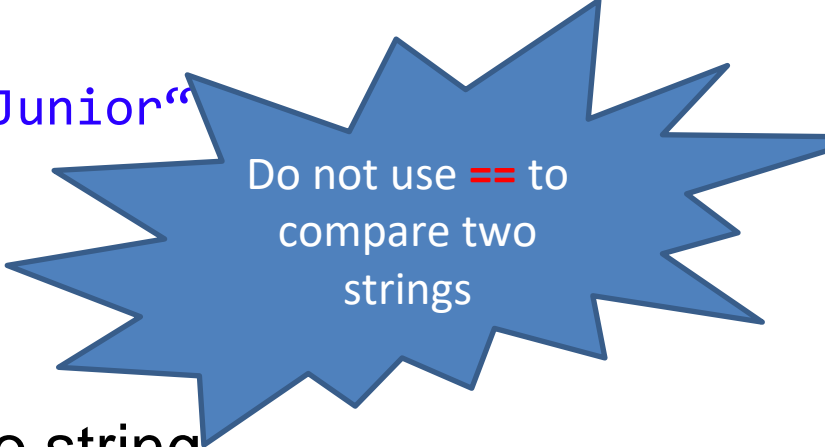
Student.Java

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 strings



Do not use `==` to
compare two
strings