# Classes

Brahm Capoor

# Announcements

3 handouts: Assignment #3, Section #3, Bouncing Ball

# Announcements

3 handouts: Assignment #3, Section #3, Bouncing Ball

Assignment #3 YEAH Hours: Tuesday at 6:30 in Bishop Auditorium

# Announcements

3 handouts: Assignment #3, Section #3, Bouncing Ball

Assignment #3 YEAH Hours: Tuesday at 6:30 in Bishop Auditorium

Section handouts

# Announcements

3 handouts: Assignment #3, Section #3, Bouncing Ball

Assignment #3 YEAH Hours: Tuesday at 6:30 in Bishop Auditorium

Section handouts

Assignment 2 Pain Poll

# Variables

Variables allow us to store information about the state of our program

- Local variables: Temporary state
- Instance variables: Persistent State

# Variables

Variables allow us to store information about the state of our program

- Local variables: Temporary state
- Instance variables: Persistent State

Parameters (and return values) allow us to control the flow of this information

# Variables

Variables allow us to store information about the state of our program

- Local variables: Temporary state
- Instance variables: Persistent State

Parameters (and return values) allow us to control the flow of this information

We mostly didn't need parameters in Karel. Why?

```
if (frontIsClear())
```

# Implementing Classes

# Imagine a program that manages a single student

```
public void run() {
    String studentName = “Brahm”;
    int studentId = 31415926;
    String email = “brahm@stanford.edu”;
    int numUnits = 180;
    boolean isInternational = true;



}
```

# Imagine a program that manages a single student

```
public void run() {
    String studentName = “Brahm”;
    int studentId = 31415926;
    String email = “brahm@stanford.edu”;
    int numUnits = 180;
    boolean isInternational = true;


    printDetails(studentName, studentId, email, numUnits, isInternational);
}
```

# What if we needed multiple students?

```
public void run() {
    String studentName1 = "Brahm";
    int studentId1 = 31415926;
    String email1 = "brahm@stanford.edu";
    int numUnits1 = 180;
    boolean isInternational1 = true;

    String studentName2 = "Trillian";
    int studentId2 = 27182818;
    String email2 = "hooloovoo@stanford.edu";
    int numUnits2 = 143;
    boolean isInternational2 = false;

    printDetails(studentName1, studentId1, email1, numUnits1, isInternational1);
    printDetails(studentName2, studentId2, email2, numUnits2, isInternational2);
}
```

# What if we needed multiple students?

```
public void run() {
    String studentName1 = "Brahm";
    int studentId1 = 31415926;
    String email1 = "brahm@stanford.edu";
    int numUnits1 = 180;
    boolean isInternational1 = true;

    String studentName2 = "Trillian";
    int studentId2 = 27182818;
    String email2 = "hooloovoo@stanford.edu";
    int numUnits2 = 143;
    boolean isInternational2 = false;

    printDetails(studentName1, studentId1, email1, numUnits1, isInternational1);
    printDetails(studentName2, studentId2, email2, numUnits2, isInternational2);
}
```

Style: ✓-

# Wouldn’t this be nice?

```
public void run() {
     Student s1 = new Student(“Brahm”, 31415926, “brahm@stanford.edu”, 180, true);
     Student s2 = new Student(“Trillian”, 27182818, “hooloovoo@stanford.edu”, 143, false);

     s1.printDetails();
     s2.printDetails();
}
```

# A brief history lesson

The first programming languages in the 1960s were called procedural languages

- Comprised of sequences of statements (kind of like Karel)
- Emphasized strategic top down design
- Examples: C, BASIC, COBOL, Fortran

# A brief history lesson

In the late 1960s, a language called Smalltalk was made to facilitate object-oriented programming

> “The design of a language for using computers must deal with internal models”
>
> Translation: programming involves things which have properties and behaviours

# A brief history lesson

Object Oriented Programming began to gain popularity in the 1970s

- Bjorn Stroustrup added it to C to make C++, the first commercially-used language with OOP

# A brief history lesson

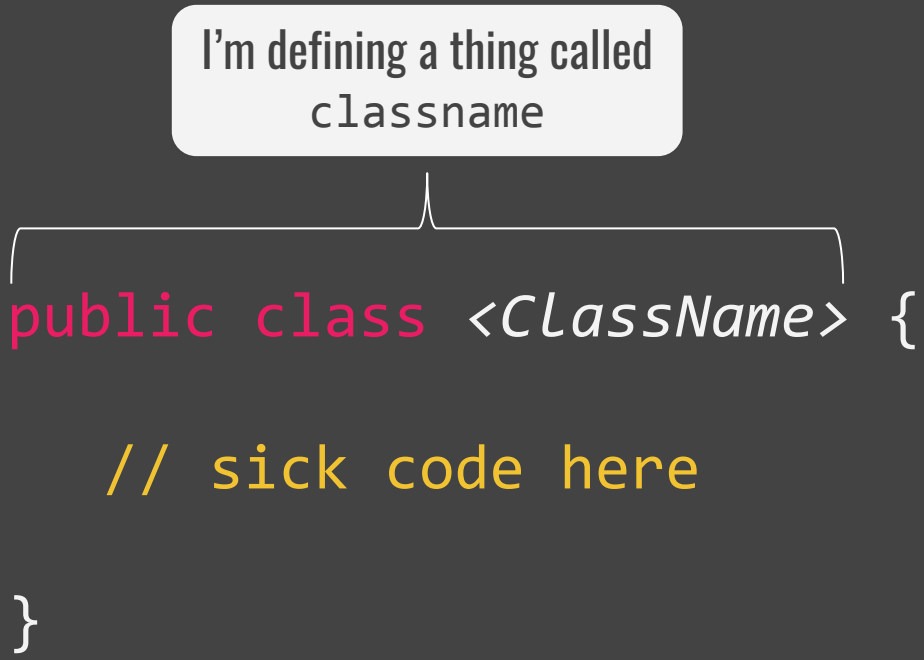
The first major Object-Oriented Programming language to be designed from the ground up was Java

# A brief history lesson

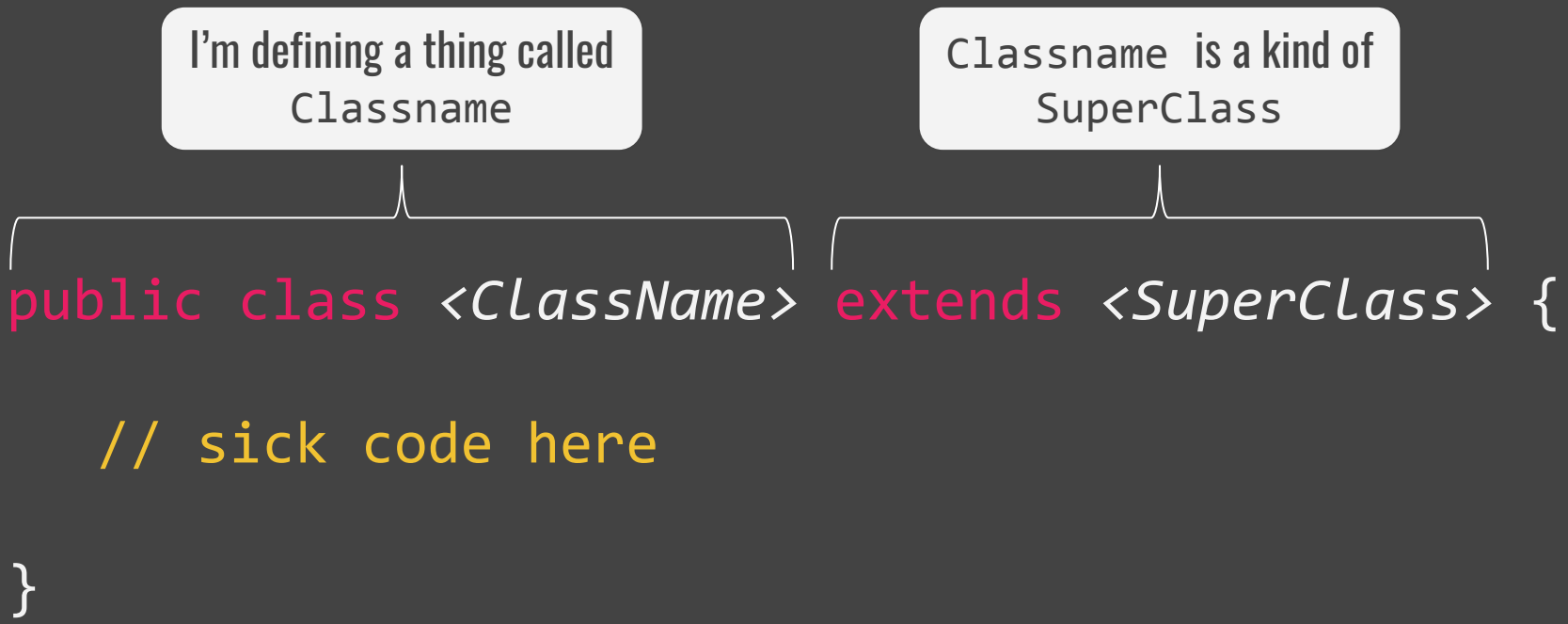
The first major Object-Oriented Programming language to be designed from the ground up was Java

Which brings us to 2018...

```
public class <ClassName> {

   // sick code here

}
```


I'm defining a thing called
classname
public class <ClassName> {
// sick code here
}

```
public class <ClassName> extends <SuperClass> {

   // sick code here

}
```


I'm defining a thing called Classname
Classname is a kind of SuperClass
public class <ClassName> extends <SuperClass> {
// sick code here
}

```
public class Student {

    // sick code here

}
```

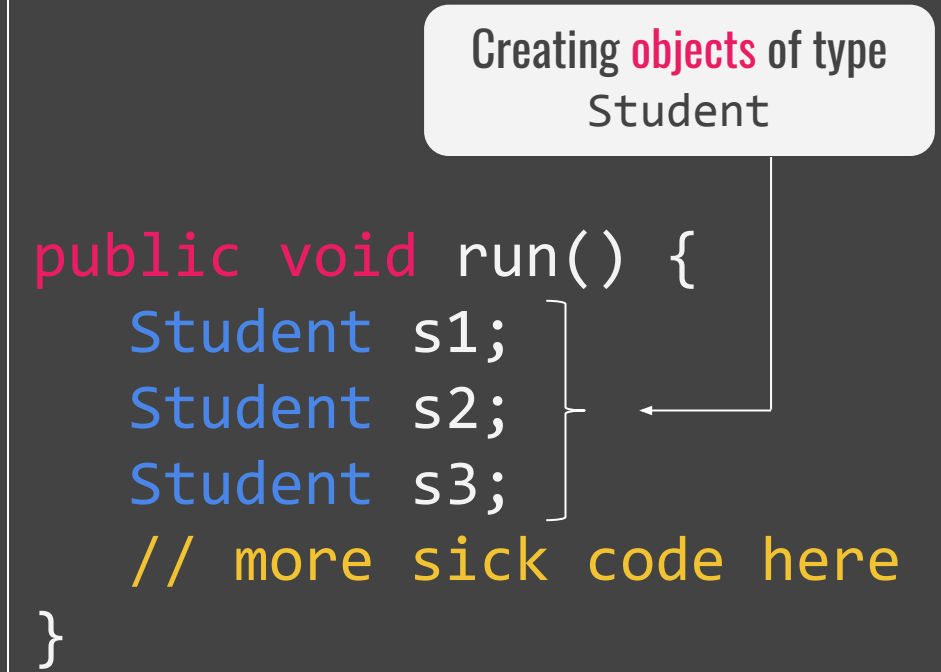
Student.java


```
public class Student {

    // sick code here

}
```

Student.java

```java
public class Student {

    // sick code here

}
```

Stanford.java



```
public class Student {

    // sick code here

}
```

1) ~~Programming involves things~~
2) ...which have properties
3) ...and behaviour

# Instance variables

Defined as part of a class, but not within any particular method

# Instance variables

Defined as part of a class, but not within any particular method

```
public void run() {

    String studentName = “Brahm”;
    int studentId = 31415926;
    String email = “brahm@stanford.edu”;
    int numUnits = 180;
    boolean isInternational = true;
}
```

# Instance variables

Defined as part of a class, but not within any particular method

```
public class Student {

    private String studentName;
    private int studentId;
    private String email;
    private int numUnits;
    private boolean isInternational;
}
```

# Instance variables

Defined as part of a class, but not within any particular method

```
public class Student {

    private String studentName;
    private int studentId;
    private String email;
    private int numUnits;
    private boolean isInternational;
}
```

`s1`, `s2` and `s3` all have their own independent properties

```
public void run() {

    Student s1;
    Student s2;
    Student s3;

}
```

# What does `private` mean?

`Private` (and `public`) are called visibility specifiers

If a method or variable is private, it is only accessible in the class in which it is defined

If a method or variable is public, it is accessible in any class

# Initializing your instance variables in the constructor

```
public class Student {

    public Student(String name, int id, String email,
                   int numUnits, boolean isInternational) {
        studentName = name;
        studentId = id;
        email = email;
        numUnits = numUnits;
        isInternational = isInternational;
    }

    /* instance variables go down here */
}
```

# Initializing your instance variables in the constructor

```
public class Student {

    public Student(String name, int id, String email,
                   int numUnits, boolean isInternational) {
        studentName = name;
        studentId = id;
        email = email; // BUGGY: Setting a variable to itself!
        numUnits = numUnits;
        isInternational = isInternational;
    }

    /* instance variables go down here */
}
```

# Initializing your instance variables in the constructor

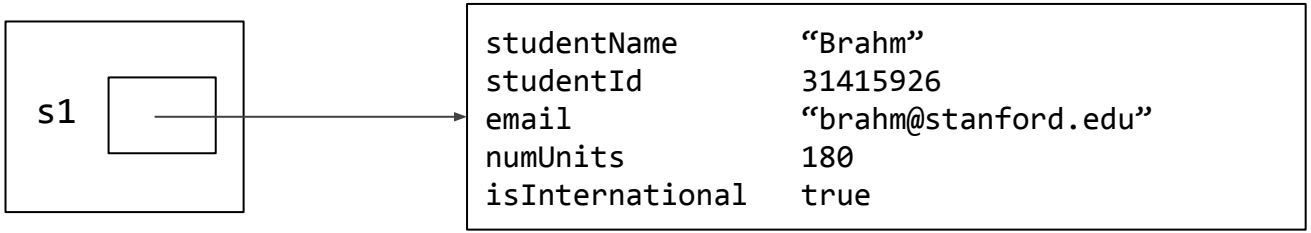
```
public class Student {

    public Student(String name, int id, String email,
                   int numUnits, boolean isInternational) {
        studentName = name;
        studentId = id;
        this.email = email; // to disambiguate between variables
        this.numUnits = numUnits;
        this.isInternational = isInternational;
    }

    /* instance variables go down here */
}
```

# Now we can make students!

```
public class Student {

    public Student(String name, int id, String email,
                   int numUnits, boolean isInternational) {...}

}
```

```
public void run() {

    Student s1 = new Student("Brahm", 31415926, "brahm@stanford.edu",
                             180, true);
}
```

# Now we can make students!



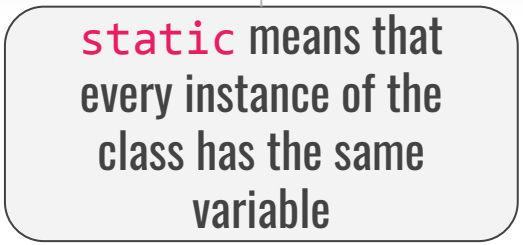

```
public void run() {

    Student s1 = new Student(“Brahm”, 31415926, “brahm@stanford.edu”,
                             180, true);
}
```

# It’s also possible for objects to share the same variable

```
public class Student {

    public static final double UNITS_TO_GRADUATE = 180;
    /*etc*/
}
```

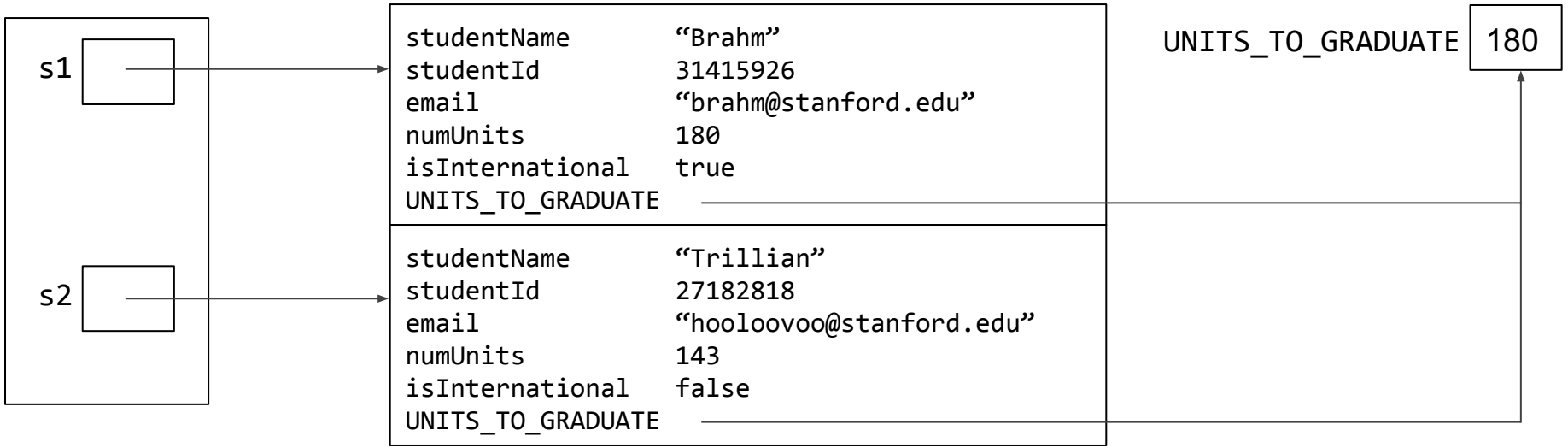
# It's also possible for objects to share the same variable

```
public class Student {

    public static final double UNITS_TO_GRADUATE = 180;
    /*etc*/
}
```



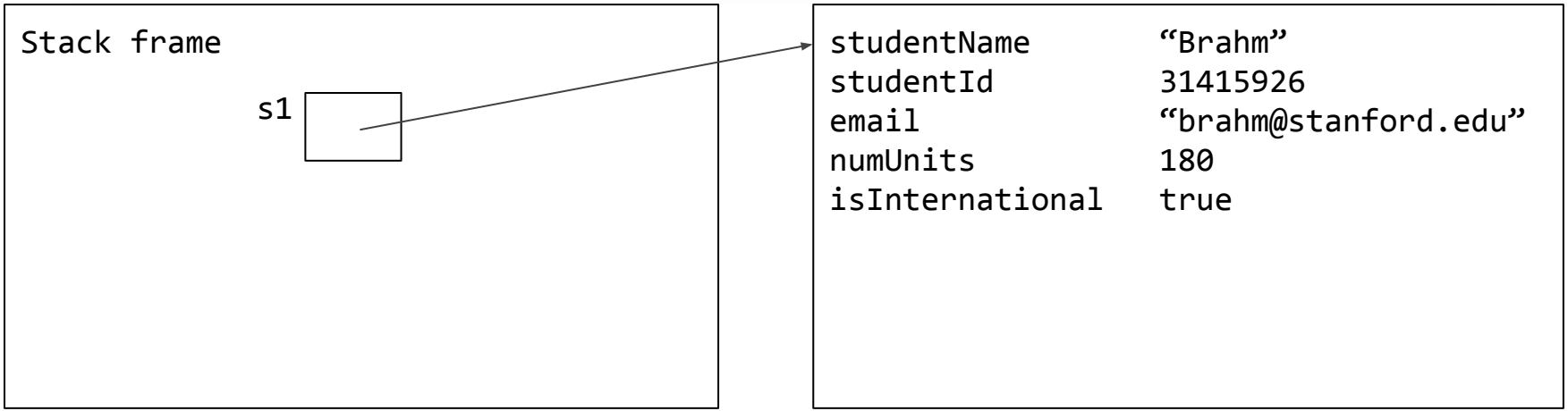
# How does this look?

```
Student s1 = new Student("Brahm", 31415926, "brahm@stanford.edu",
                         180, true);
Student s2 = new Student("Sophia", 27182818, "hooloovoo@stanford.edu",
                         143, false);
```



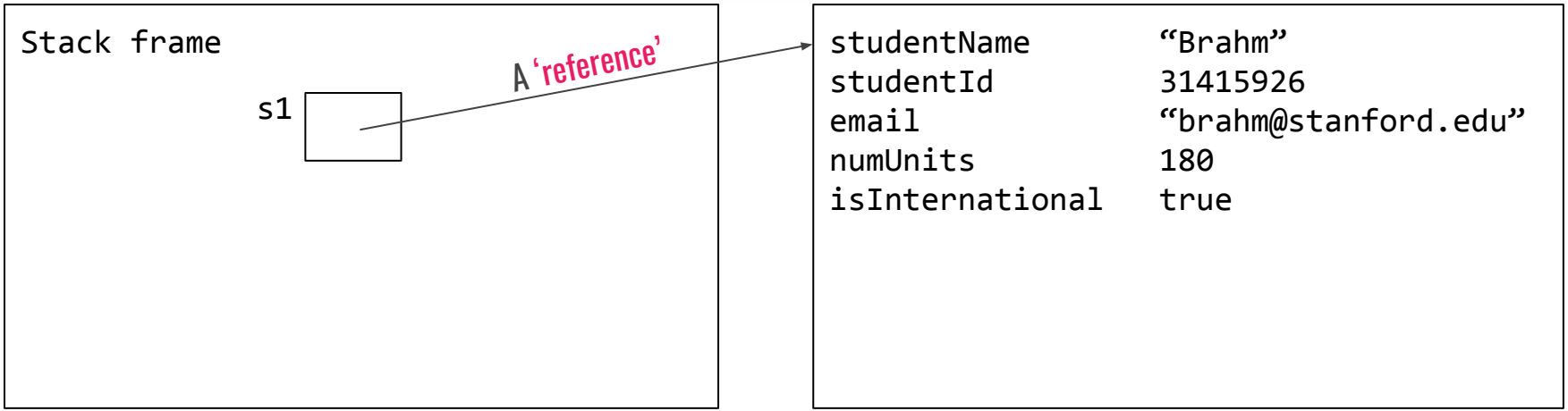
# Under the hood

```
Student s1 = new Student("Brahm", 31415926, "brahm@stanford.edu", 180, true);
```



# Under the hood

```
Student s1 = new Student("Brahm", 31415926, "brahm@stanford.edu", 180, true);
```



# s1 and s2 store references to Student objects

```
public class Student {

    public Student(String name, int id, String email,
                   int numUnits, boolean isInternational) {...}

}
```

```
public void run() {

    Student s1 = new Student("Brahm", 31415926, "brahm@stanford.edu",
                             180, true);
    Student s2 = new Student("Trillian", 27182818,
                             "hooloovoo@stanford.edu", 143, false);
}
```

# s1 and s2 store references to Student objects

```
public class Student {

    public Student(String name, int
                   int numUnits, bo

}
```

```
public void run() {

    Student s1 = new Student("Brahm", 3
                             180, true);
    Student s2 = new Student("Trillian", 27182818,
                             "hooloovoo@stanford.edu", 143, false);
}
```

1) ~~Programming involves things~~
2) ~~...which have properties~~
3) ...and behaviour

# Using an object's properties

Variables are only useful because we can get and set their values

- This is also true for instance variables

# Using an object's properties

Variables are only useful because we can get and set their values

Instance variables are private

- We can get and set their values in the same class
- It would be nice to be able to do so in other classes as well
- A `Stanford` program should be able to change a student's number of units, for example

# Our first Getters and Setters

```
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }









    private int numUnits;

}
```

```
public void run() {

    Student s1 = new Student(42);









}
```

# Our first Getters and Setters

```
public class Student {

     public Student(int unitCount) {
          numUnits = unitCount;
     }

     public int getUnits() {
          return numUnits;
     }




     private int numUnits;

}
```

```
public void run() {

     Student s1 = new Student(42);



     println(“Curr:” + s1.getUnits());









}
```

# Our first Getters and Setters

```
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }

    public int getUnits() {
        return numUnits;
    }

    public void setUnits(int newUnits) {
        numUnits = newUnits;
    }

    private int numUnits;

}
```

```
public void run() {

    Student s1 = new Student(42);



    println("Curr:" + s1.getUnits());



    s1.setUnits(60);









}
```

# Our first Getters and Setters: some notes

```
public class Student {

     public Student(int unitCount) {
          numUnits = unitCount;
     }

     public int getUnits() {
          return numUnits;
     }

     public void setUnits(int newUnits) {
          numUnits = newUnits;
     }

     private int numUnits;

}
```

Getter and Setter methods are public (exported) so we can call them in other classes and programs

# Our first Getters and Setters: some notes

```
public class Student {

     public Student(int unitCount) {
          numUnits = unitCount;
     }

     public int getUnits() {
          return numUnits;
     }

     public void setUnits(int newUnits) {
          numUnits = newUnits;
     }

     private int numUnits;

}
```

Getter and Setter methods are public (exported) so we can call them in other classes and programs

Define Getters and Setters whenever you want to grant a client access to or control over an instance variable

# Our first Getters and Setters: some notes

```
public class Student {

     public Student(int unitCount) {
          numUnits = unitCount;
     }

     public int getUnits() {
          return numUnits;
     }

     public void setUnits(int newUnits) {
          numUnits = newUnits;
     }

     private int numUnits;

}
```

Getter and Setter methods are public (exported) so we can call them in other classes and programs

Define Getters and Setters whenever you want to grant a client access to or control over an instance variable

These methods are typically very short

# Our first Getters and Setters: some notes

```
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }

    public int getUnits() {
        return numUnits;
    }

    public void setUnits(int newUnits) {
        numUnits = newUnits;
    }

    private int numUnits;

}
```

Getter and Setter methods are public (exported) so we can call them in other classes and programs

Define Getters and Setters whenever you want to grant a client access to or control over an instance variable

These methods are typically very short

Why not just make these variables public?

# Our first Getters and Setters: some notes

```
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }

    public int getUnits() {
        return numUnits;
    }

    public void setUnits(int newUnits) {
        numUnits = newUnits;
    }

    private int numUnits;

}
```

Getter and Setter methods are public (exported) so we can call them in other classes and programs

Define Getters and Setters whenever you want to grant a client access to or control over an instance variable

These methods are typically very short

Why not just make these variables public?

```
Student s1 = new Student(42);
s1.numUnits = 0; // why is this problematic?
```

# Our first Getters and Setters: some notes

```
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }

    public int getUnits() {
        return numUnits;
    }

    public void setUnits(int newUnits) {
        numUnits = newUnits;
    }

    private int numUnits;

}
```

Getter and Setter methods are public (exported) so we can call them in other classes and programs

Define Getters and Setters whenever you want to grant a client access to or control over an instance variable

These methods are typically very short

They allow more precise control over the value of a variable:

```
public void setUnits(int newUnits) {
    if (newUnits >= numUnits) {
        numUnits = newUnits;
    }
}
```

# Why stop there?

Now that we know how to use instance variables, we can do even cooler things

```
public boolean canGraduate() {
     return numUnits >= 180;
}
```

```
public void dropClass (int classUnits) {
     if (classUnits <= 5) {
          numUnits -= classUnits;
     }
}
```

# Why stop there?

Now that we know how to use instance variables, we can do even cooler things

```
public boolean canGraduate() {
     return numUnits >= 180;
}
```

```
public void dropClass (int classUnits) {
     if (classUnits <= 5) {
          numUnits -= classUnits;
     }
}
```

# Methods allow us to define behaviours for our classes

# One special method

How could we do this?

```
Student s1 = new Student(...);
println(s1); //s1 isn't a string!
```

# One special method

How could we do this?

```
Student s1 = new Student(...);
println(s1); //s1 isn't a string!
```

We define a `toString()` method

```
public String toString() {
    return studentName + " (#" + studentId + ")";
}
```

# One special method

How could we do this?

```
Student s1 = new Student(...);
println(s1); //s1 isn't a string!
```

We define a `toString()` method

```
public String toString() {
    return studentName + " (#" + studentId + ")";
}
```

...and now this magically works!

```
Student s1 = new Student(...);
println(s1); //Prints "Brahm (#31415926)"
```

# One special method

How could we do this?

```
Student s1 =
println(s1);
```

We define a `toString()` method

```
public String
    return s
}
```

";

~~1) Programming involves things~~
~~2) …which have properties~~
~~3) …and behaviour~~

…and now this magically works!

```
Student s1 = new Student(...);
println(s1); //Prints “Brahm (#31415926)”
```


Demo

# Passing an object as a parameter

```
public void run() {
     int x = 7;
     doSomething(x);
     println(x); // prints 7
}

private void doSomething(int n) {
     n *= 2;
}
```

# Passing an object as a parameter

```
public void run() {
     int x = 7;
     doSomething(x);
     println(x); // prints 7
}

private void doSomething(int n) {
     n *= 2;
}
```

```
public void run() {
     Student s1 = new Student(42);
     doSomething(s1);
     println(s1.getUnits());
}

private void doSomething(Student s) {
     s.setUnits(s.getUnits() * 2);
}
```

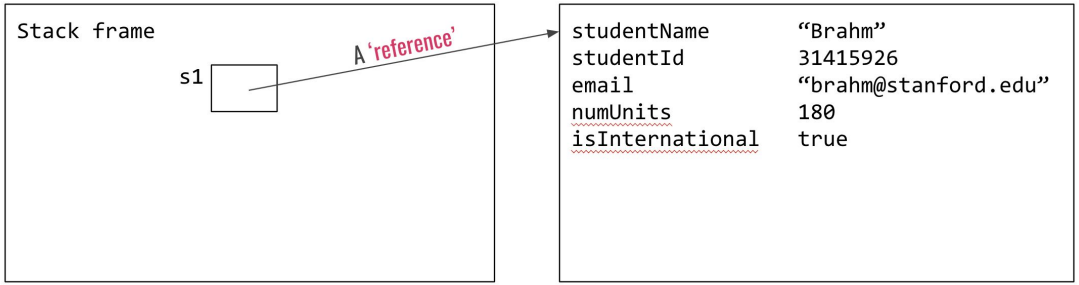
# Passing an object as a parameter

```
public void run() {
    int x = 7;
    doSomething(x);
    println(x); // prints 7
}

private void doSomething(int n) {
    n *= 2;
}
```

```
public void run() {
    Student s1 = new Student(42);
    doSomething(s1);
    println(s1.getUnits()); // prints 84
}

private void doSomething(Student s) {
    s.setUnits(s.getUnits() * 2);
}
```

# How does this work?

Object variables store references




It helps to think of a reference as the location of the object elsewhere in your computer

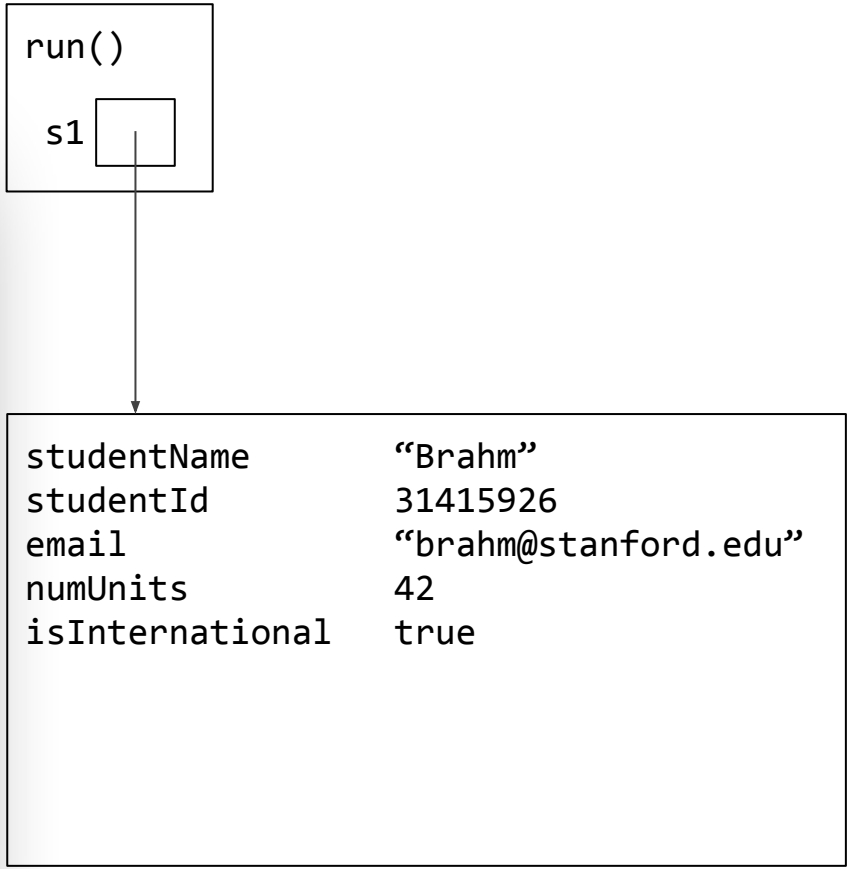
When we pass in an object to a method, we pass in a reference to that object

# How does this work?

```
public void run() {
     Student s1 = new Student(42);
     doSomething(s1);
     println(s1.getUnits());
}

private void doSomething(Student s) {
     s.setUnits(s.getUnits() * 2);
}
```

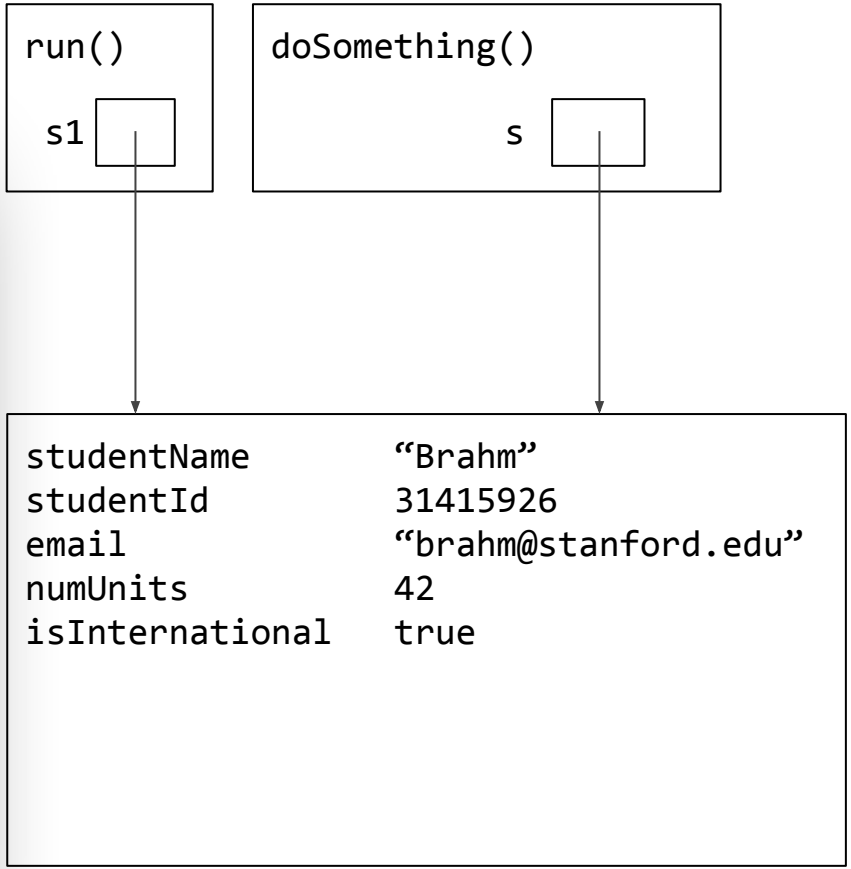
# How does this work?

```
public void run() {
    Student s1 = new Student(..., 42);
    doSomething(s1);
    println(s1.getUnits());
}

private void doSomething(Student s) {
    s.setUnits(s.getUnits() * 2);
}
```



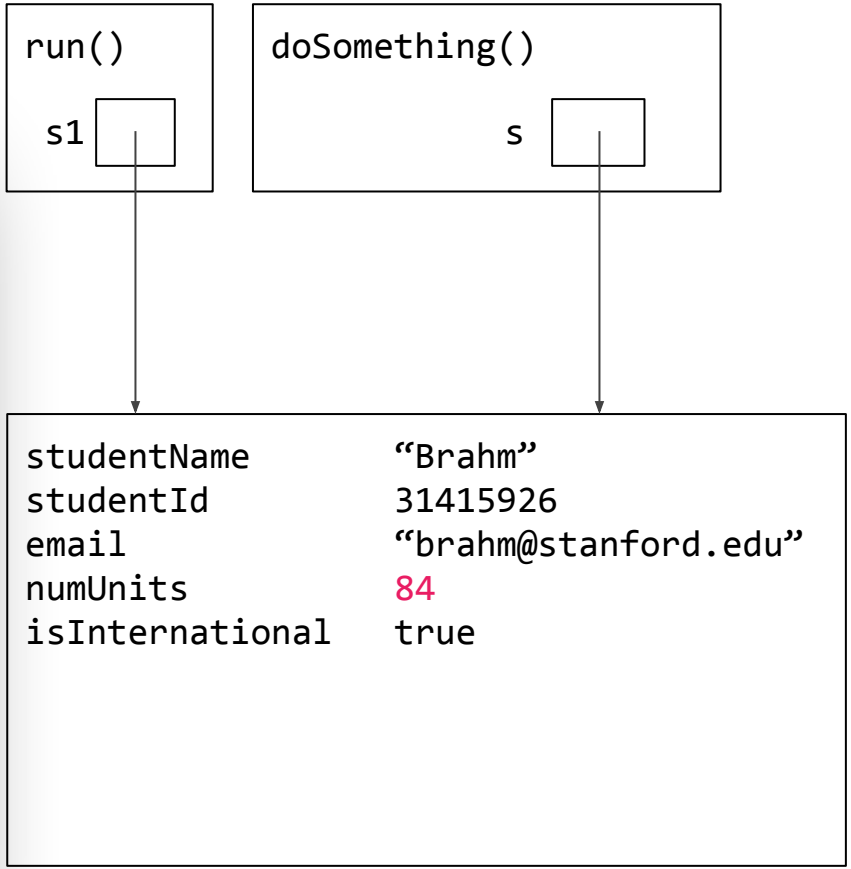
# How does this work?

```
public void run() {
    Student s1 = new Student(..., 42);
    doSomething(s1);
    println(s1.getUnits());
}

private void doSomething(Student s) {
    s.setUnits(s.getUnits() * 2);
}
```



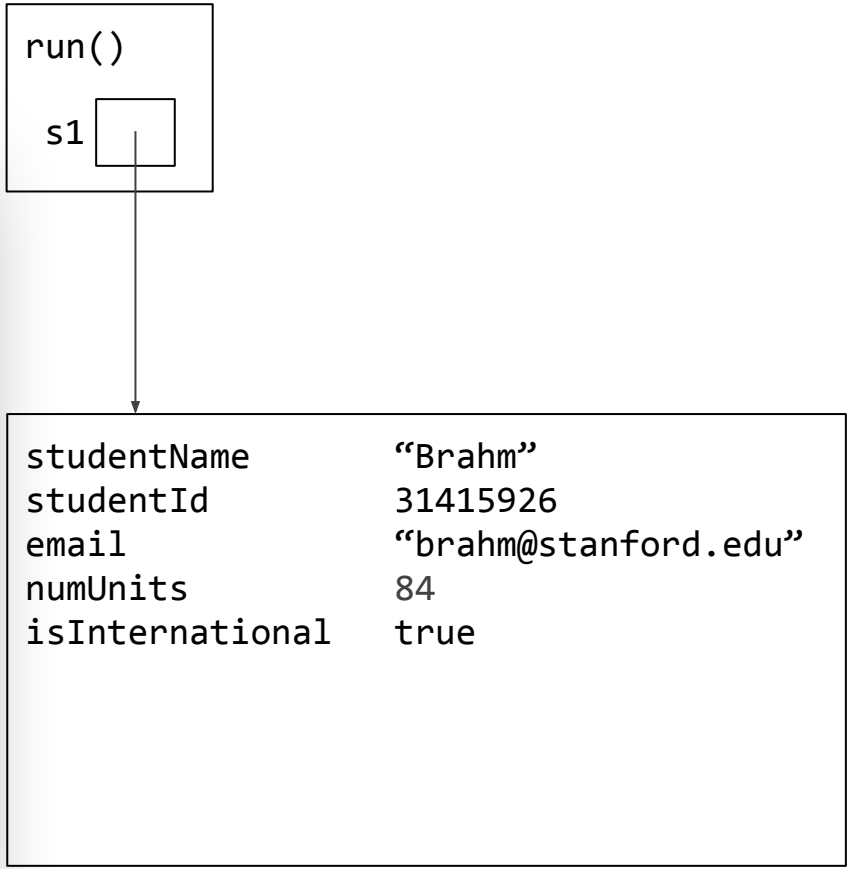
# How does this work?

```
public void run() {
     Student s1 = new Student(..., 42);
     doSomething(s1);
     println(s1.getUnits());
}

private void doSomething(Student s) {
     s.setUnits(s.getUnits() * 2);
}
```



# How does this work?

```
public void run() {
     Student s1 = new Student(..., 42);
     doSomething(s1);
     println(s1.getUnits());
}

private void doSomething(Student s) {
     s.setUnits(s.getUnits() * 2);
}
```



# A summary

```
public void run() {
     int x = 7;
     doSomething(x);
     println(x); // prints 7
}

private void doSomething(int n) {
     n *= 2;
}
```

Passing by copy/value

```
public void run() {
     Student s1 = new Student(42);
     doSomething(s1);
     println(s1.getUnits()); // prints 84
}

private void doSomething(Student s) {
     s.setUnits(s.getUnits() * 2);
}
```

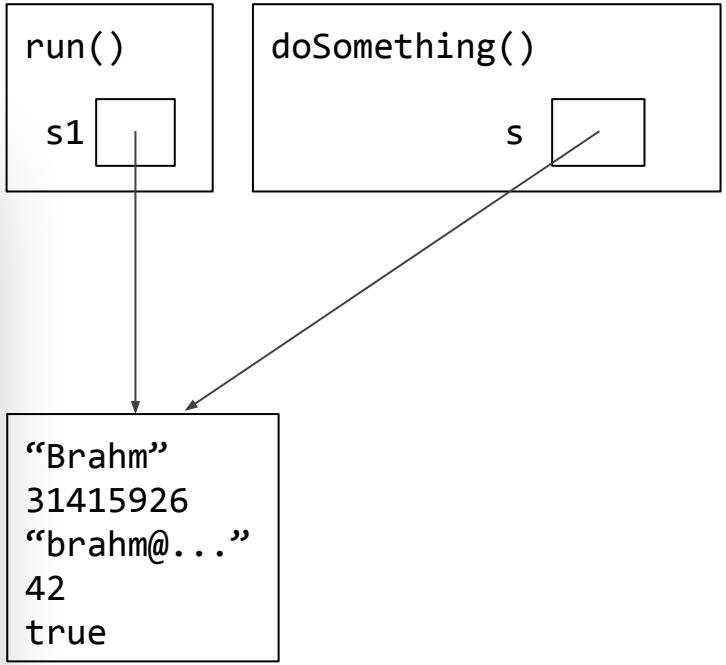
Passing by reference

# A nuance: we're passing copies of references

# A nuance: we're passing copies of references

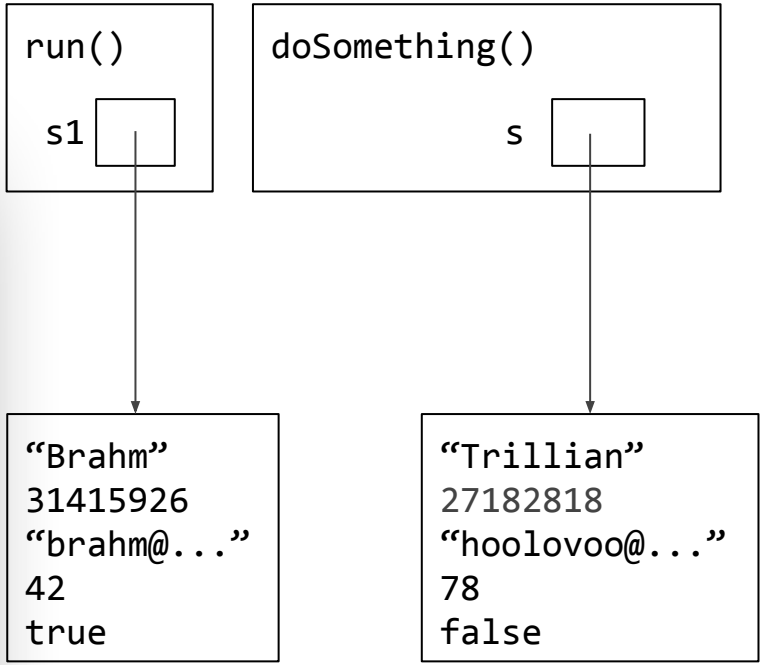
# What does that mean?

```
public void run() {
     Student s1 = new Student(...);
     doSomething(s1);
     println(s1.getUnits());
}

private void doSomething(Student s) {
     s = new Student(...);
}
```



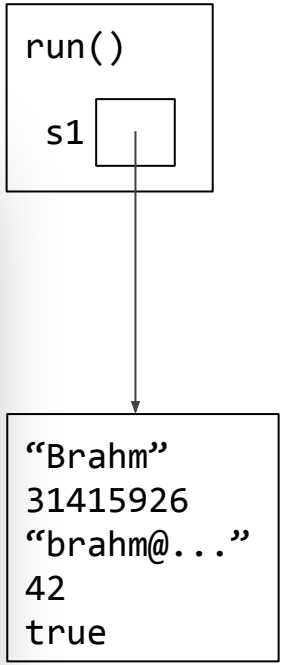
# What does that mean?

```
public void run() {
     Student s1 = new Student(...);
     doSomething(s1);
     println(s1.getUnits());
}

private void doSomething(Student s) {
     s = new Student(...);
}
```



# What does that mean?

```
public void run() {
     Student s1 = new Student(...);
     doSomething(s1);
     println(s1.getUnits());
}

private void doSomething(Student s) {
     s = new Student(...);
}
```



# What have we done?

We identified a single kind of entity our program interacted with

# What have we done?

We identified a single kind of entity our program interacted with

We modelled that entity in Java by specifying its properties and behaviour

# What have we done?

We identified a single kind of entity our program interacted with

We modelled that entity in Java by specifying its properties and behaviour

We used that model as a new custom-defined type

# How can we go further?

We've defined a `Student` class, but there are different kinds of students!

# How can we go further?

We've defined a `Student` class, but there are different kinds of students!

These type of students are mostly similar but with some unique properties and behaviour

# How can we go further?

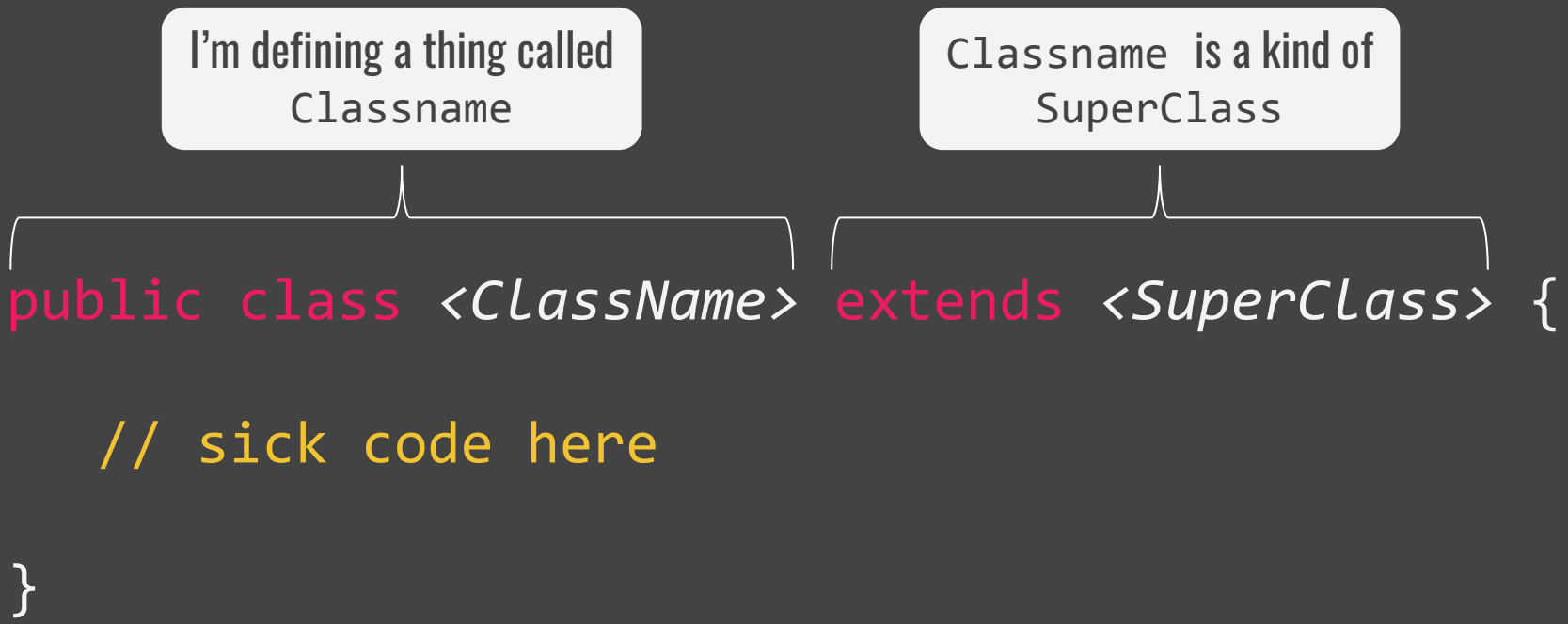
We've defined a `Student` class, but there are different kinds of students!

These type of students are mostly similar but with some unique properties and behaviour

How can we leverage our existing `Student` class to implement these new student types?

```
public class Frosh {

   // sick code here

}
```

```
public class Frosh extends Student {

   // sick code here

}
```


I'm defining a thing called Classname
Classname is a kind of SuperClass
public class <ClassName> extends <SuperClass> {
// sick code here
}

Frosh.java

```java
public class Frosh extends Student {

    // sick code here

}
```

Frosh.java

```
public class Frosh extends Student {

    // sick code here

}
```

We're subclassing, or making a subclass of, the `Student` class

```
public Student (String name, int id){...}
public void setId(int id) { this.id = id; }
public void setNUnits(int units) { nUnits = units; }
private int id, nUnits;
```

```java
public Student (String name, int id){...}
public void setId(int id) { this.id = id; }
public void setNUnits(int units) { nUnits = units; }
private int id, nUnits;
```

---

```java
public class Frosh extends Student {

   public Frosh (String name, int id) {


   }

}
```

```java
public Student (String name, int id){...}
public void setId(int id) { this.id = id; }
public void setNUnits(int units) { nUnits = units; }
private int id, nUnits;
```

---

```java
public class Frosh extends Student {

   public Frosh (String name, int id) {
      super(name, id); // call the Student constructor

   }

}
```

```java
public Student (String name, int id){...}
public void setId(int id) { this.id = id; }
public void setNUnits(int units) { nUnits = units; }
private int id, nUnits;
```

---

```java
public class Frosh extends Student {

   public Frosh (String name, int id) {
      super(name, id); // call the Student constructor
      setNUnits(0);
   }

}
```

```java
public Student (String name, int id){...}
public void setId(int id) { this.id = id; }
public void setNUnits(int units) { nUnits = units; }
private int id, nUnits;
```

---

```java
public class Frosh extends Student {

   public Frosh (String name, int id) {
      super(name, id); // call the Student constructor
      setNUnits(0); // call an inherited method
   }

}
```

```
public Student (String name, int id){...}
public void setId(int id) { this.id = id; }
public void setNUnits(int units) { nUnits = units; }
private int id, nUnits;
```

---

```
public class Frosh extends Student {
   public Frosh (String name, int id) {...}

   public String toString() {
      // overriding superclass method
      return getName() + " is a Frosh!";
   }
}
```

```
public Student (String name, int id){...}
public void setId(int id) { this.id = id; }
public void setNUnits(int units) { nUnits = units; }
private int id, nUnits;
```

---

```
public class Frosh extends Student {
   public Frosh (String name, int id) {...}

   public String toString() {
      // overriding superclass method
      return getName() + " is a Frosh!";
   }
}
```

Subclasses don't have access to private variables of superclass

# Questions?