CS111
Introduction to Computer Science

Fall 2015

- Object Oriented Programming
- Class
- Object

Programming Paradigm

- Programming Paradigm is the style of computer programming
- Paradigms:
  - Imperative, declarative, functional, procedural, object-oriented, logic, symbolic
- We'll focus on object-oriented programming
  - Java, C#, C++, Perl, Python

Object-Oriented Programming

- Groups together data (variables) and behavior (methods) related to that data into an object

- Programs are made out of objects that interact with one another

- A class is a <u>description</u> of an object
  - it describes the object in terms of its data and its behavior

Defining a Class

- Suppose we want to write a OO program to keep grades for each student
  - we define a class with:
    - variables to store the student's name and grades
    - methods to compute the average grade

```
public class Student {
   public String name;
   public double midterm1, midterm2, finalExam;

   public double getAverage() {
      return (midterm1+midterm2+finalExam)/3;
   }
}
```

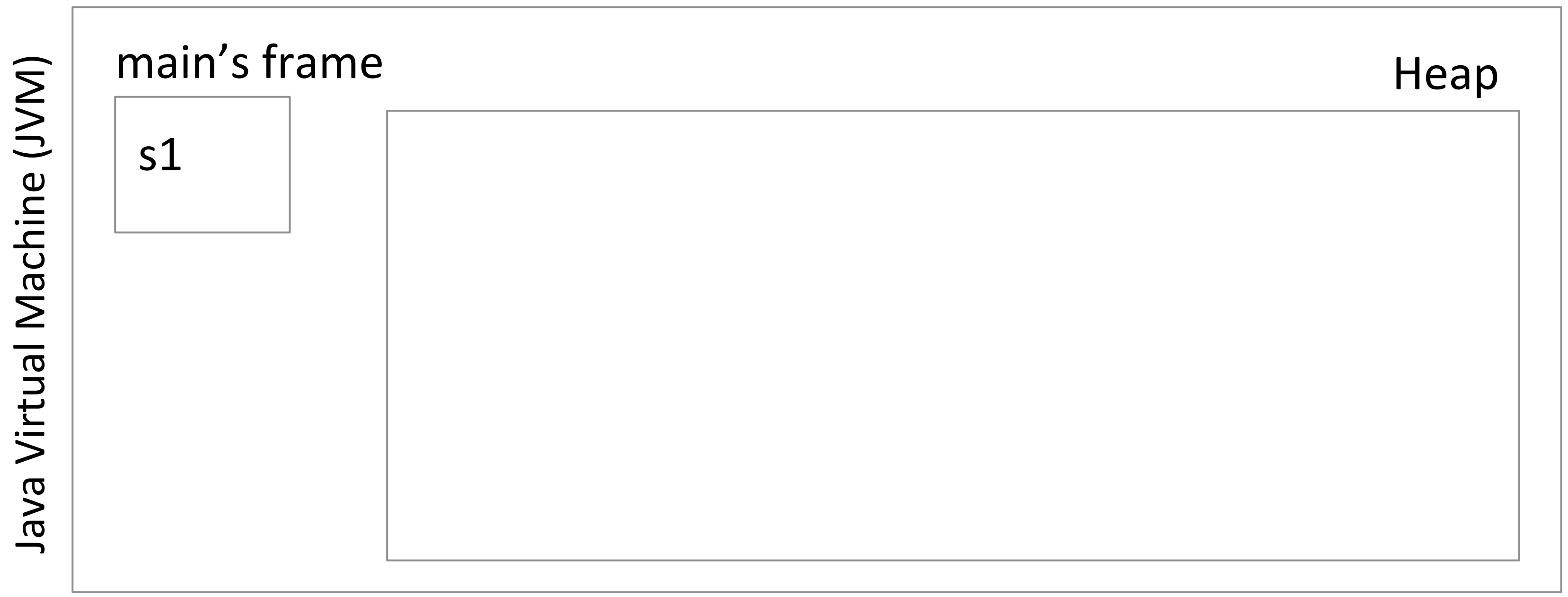
Variables or Fields

Method

Declaring a variable

```
Student s1;
```

Declares a variable of type Student

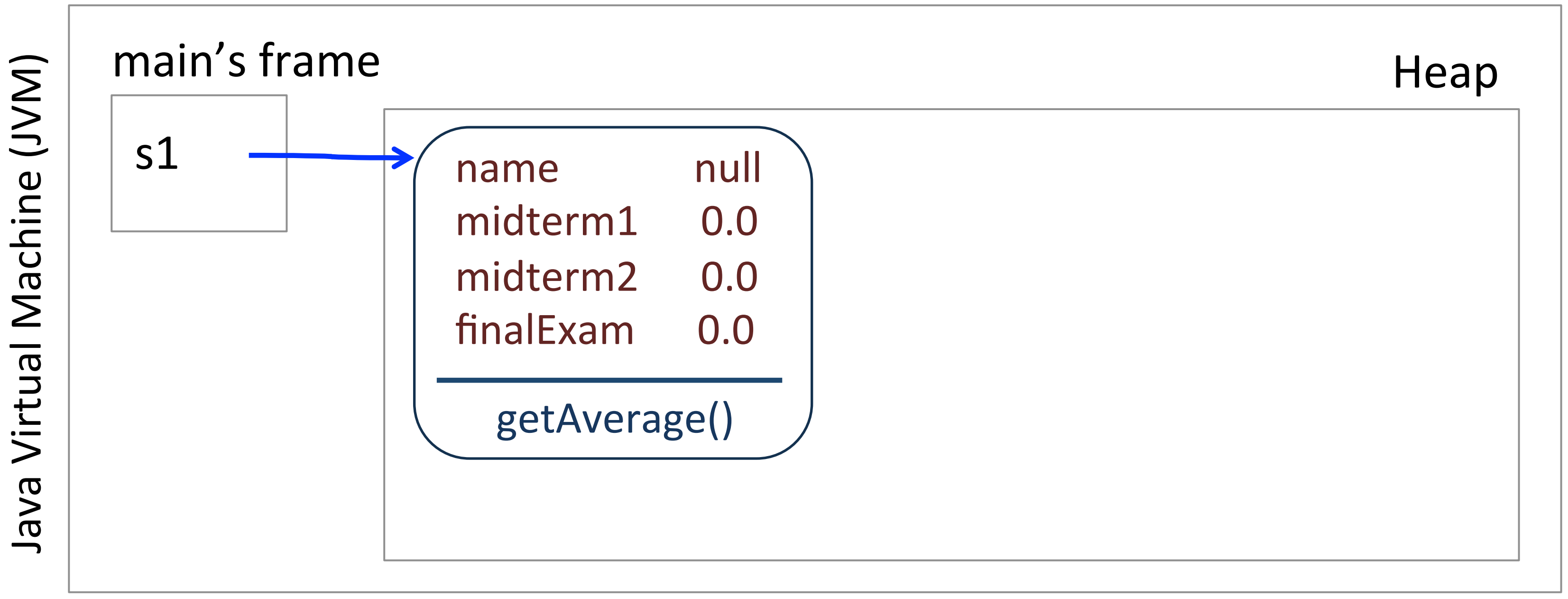


Instantiating a Class: Object

- Default constructor

```
Student s1 = new Student();
```

Declares a variable **s1** of type Student, creates space in memory for the object, and initializes its fields

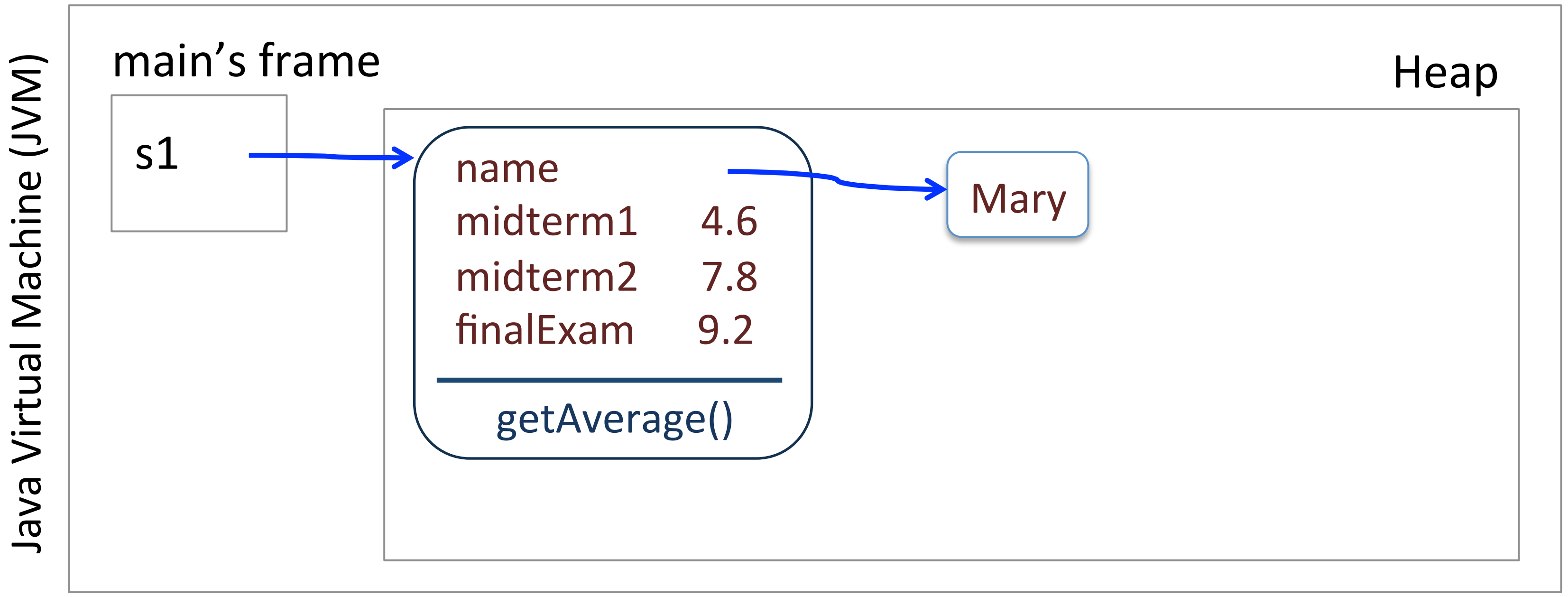


Instance Variables and Methods

```
Student s1 = new Student();
s1.name = new String("Mary");
s1.midterm1 = 4.6;
s1.midterm2 = 7.8;
s1.finalExam = 9.2;
s1.getAverage();
```

Assigning values to object **s1** variables

Invoke object **s1** method getAverage



Constructor

- Initializes the object variables

```
public class Student {
    public String name;
    public double midterm1, midterm2, finalExam;

    public Student (String name) {
        this.name = new String(name);
        midterm1 = midterm2 = finalExam = 0.0;
    }

    public double getAverage() {
        return (midterm1+midterm2+finalExam)/3;
    }
}
```

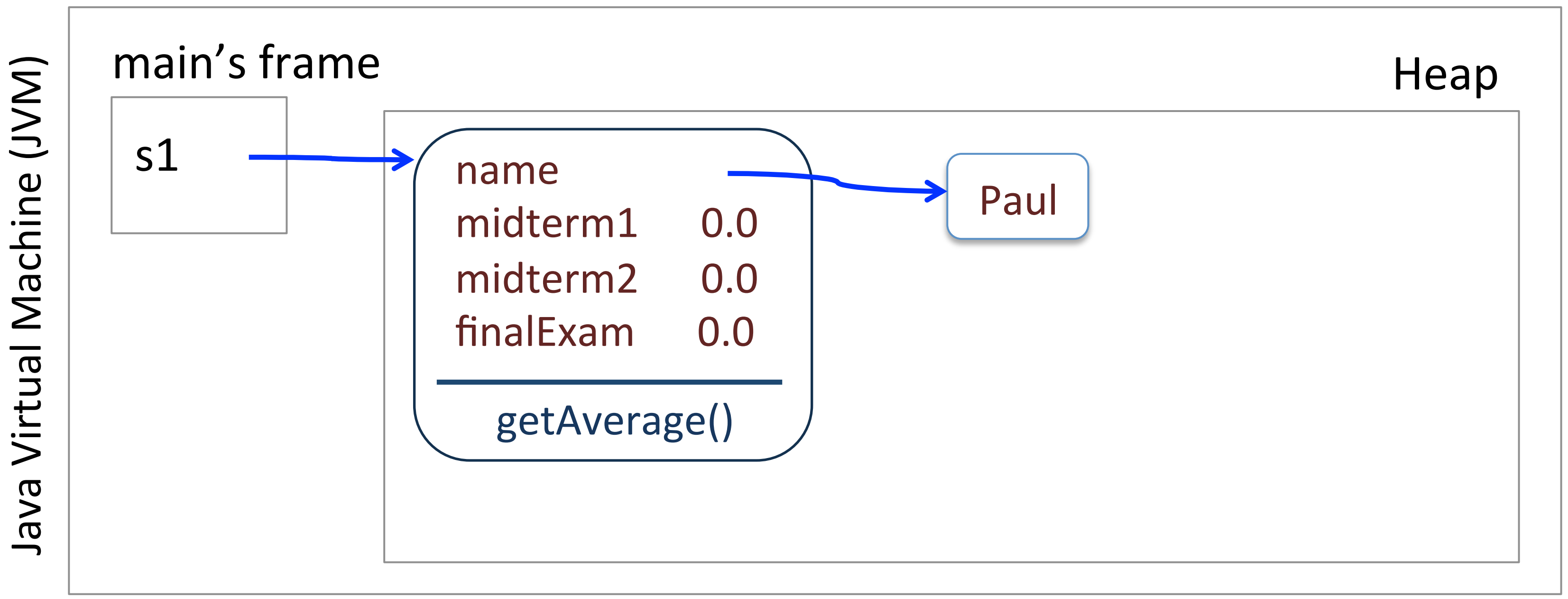
The Constructor initializes the object

this refers to current object

Instantiating a Class: Object

- Constructor from our class

```
Student s1 = new Student(“Paul”);
```

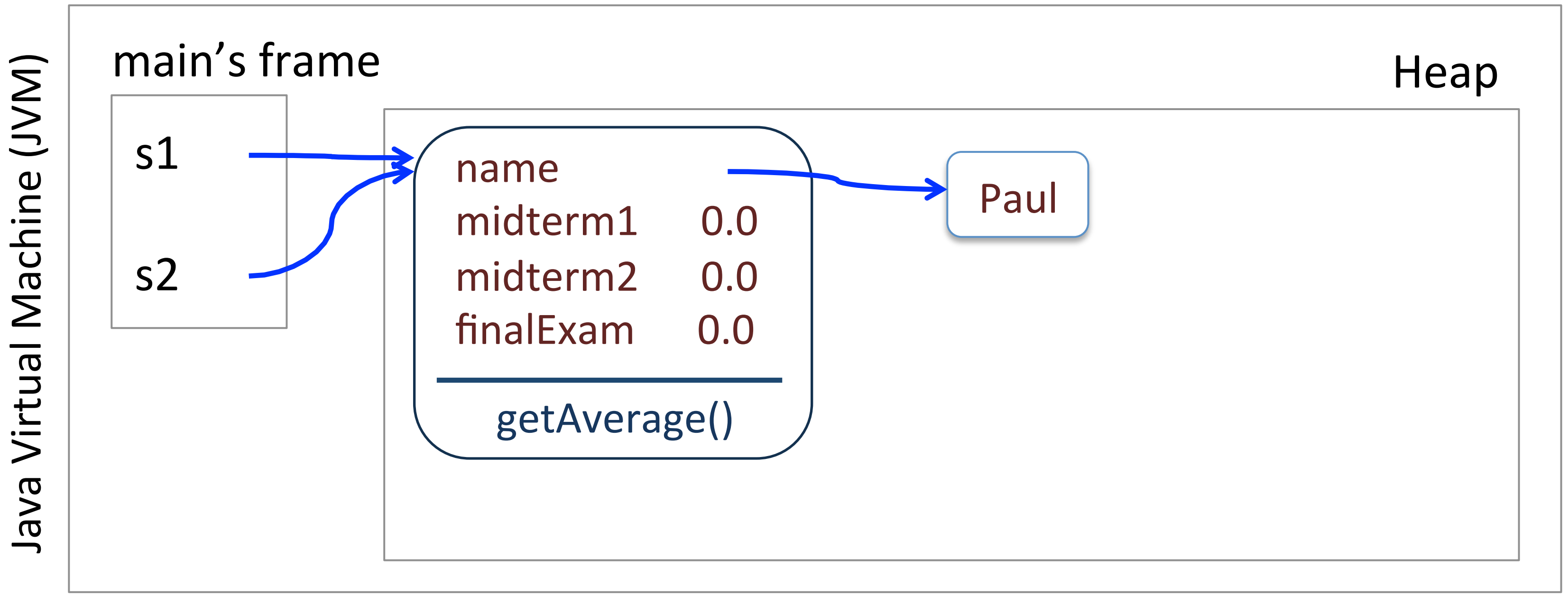


Variables do not hold Objects

- A variable can only hold a reference to an object

```
Student s1 = new Student("Paul");
Student s2 = s1;
```

Declares a variable **s2** of type Student that refers to the same Student object **s1** refers to.

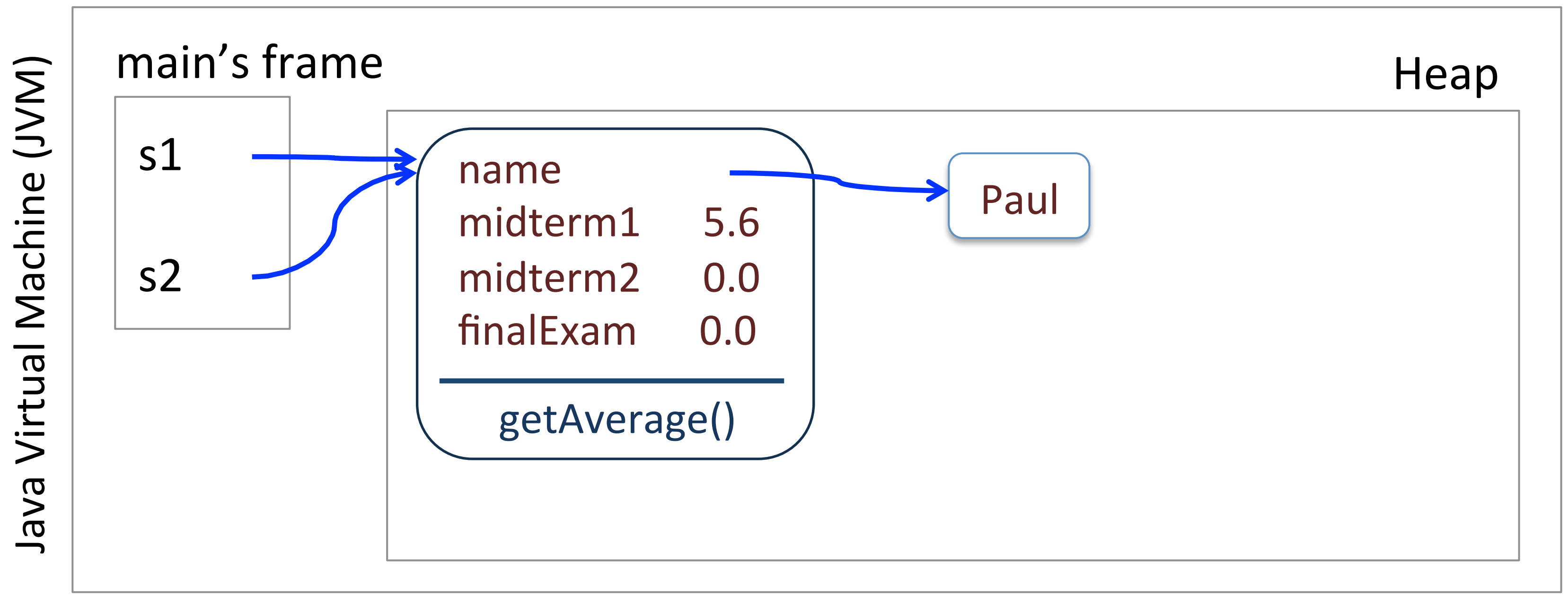


Modifying an Object

- A variable can only hold a reference to an object

```
Student s1 = new Student("Paul");
Student s2 = s1;
s2.midterm1 = 5.6;
```

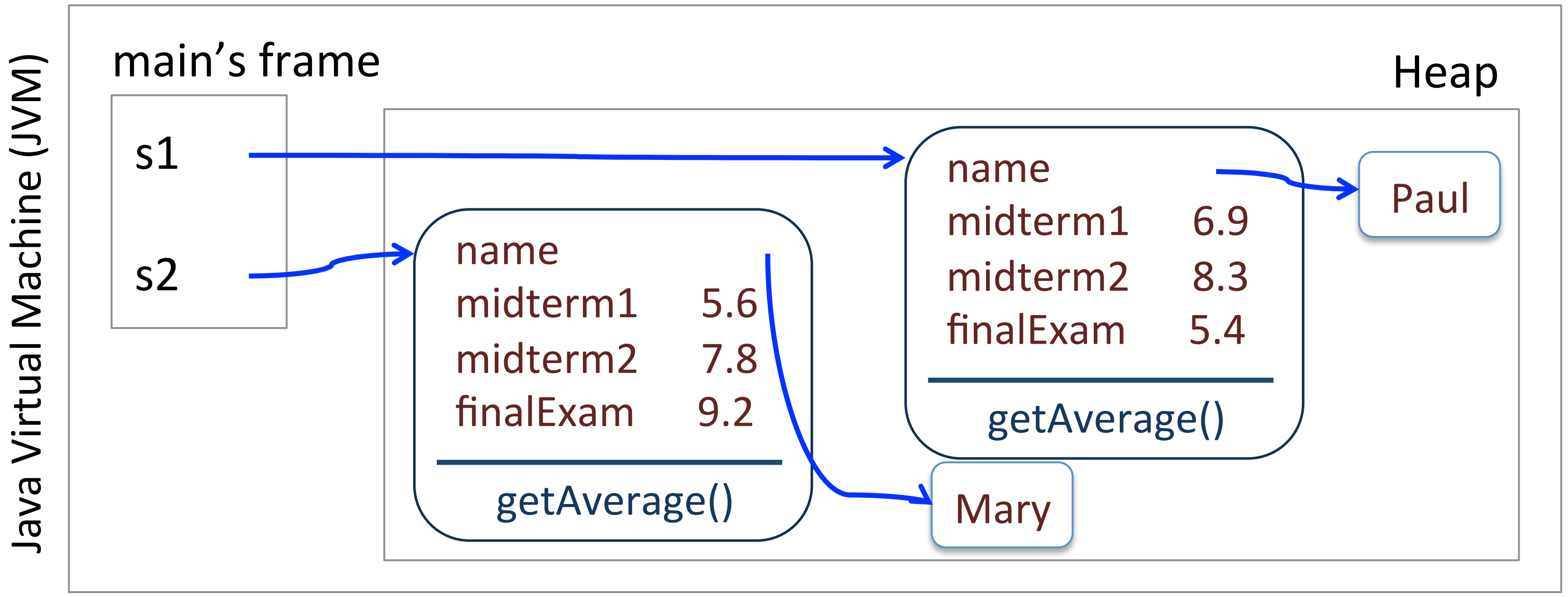
s2 modifies the object but **s1** also sees the modification.



Creating Multiple Objects

```
Student s1 = new Student("Paul");
Student s2 = new Student("Mary");
s2.midterm1 = 5.6;
s2.midterm2 = 7.8;
s2.finalExam = 9.2;
```

```
s1.midterm1 = 6.9;
s1.midterm2 = 8.3;
s1.finalExam = 5.4;
s1.getAverage();
s2.getAverage();
```



CS111
Introduction to Computer Science

Fall 2015

- Testing objects equality
- Objects as parameters
- Garbage collector
- Static and non static variables and methods

Testing for Objects Equality

- Suppose s1 and s2 refer to Student objects
- When you make a test

```
if (s1 == s2)
```

  - tests if the values stored in s1 and s2 are the same
  - Since s1 and s2 are references to objects you are comparing if s1 and s2 refer to the same object!

```
if (s1.equals(s2))
```

  - tests if the objects s1 and s2 have the same contents

See Student.java for the equals method implementation

Passing Objects as Parameters

- The method receives a reference to an object
  - Through the reference the object can be modified

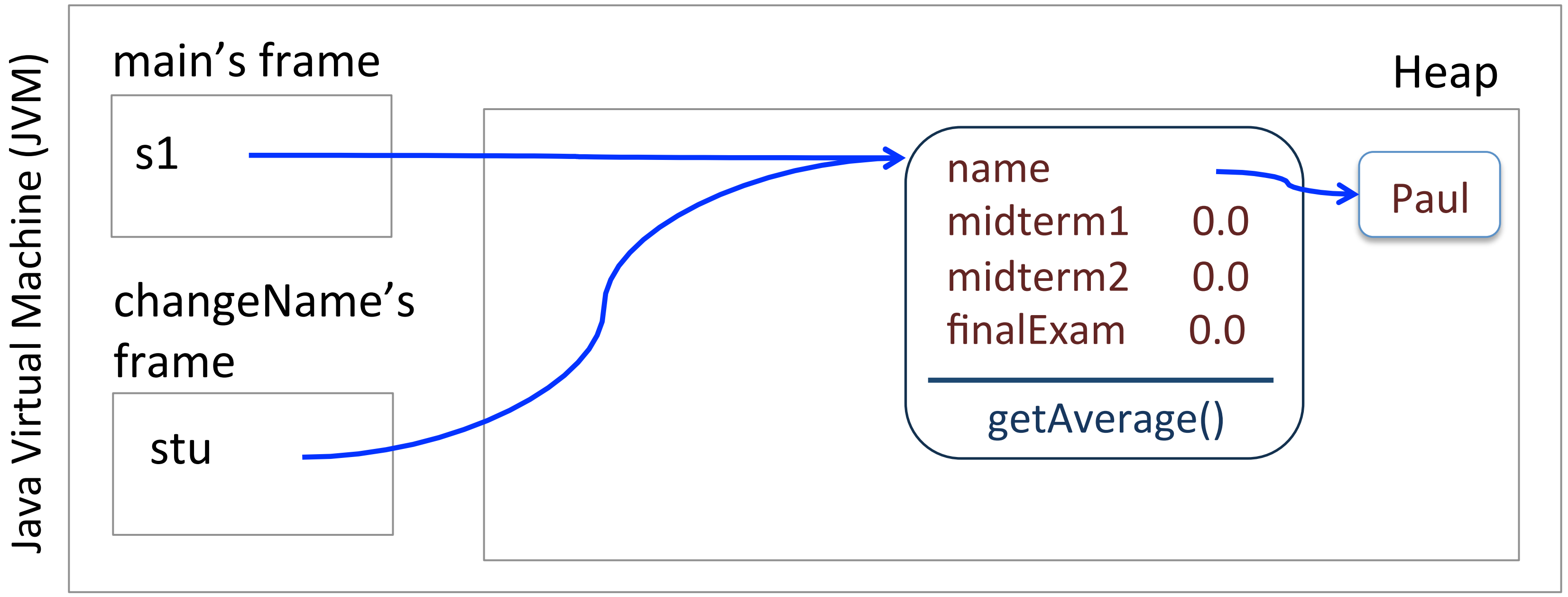
```
void changeName (Student stu) {
    stu.name = new String(“Mary”);
}

Student s1 = new Student(“Paul”);
changeName(s1);
System.out.println(s1.name);

output is “Mary”
```

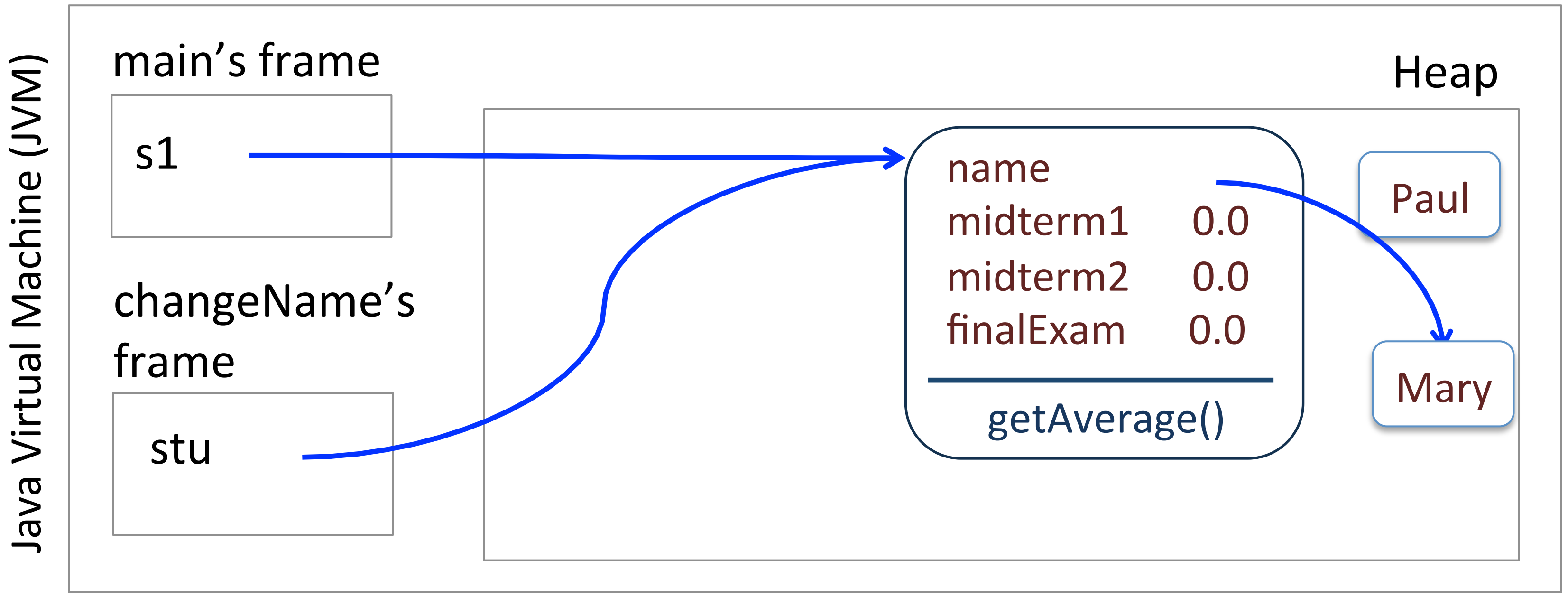
Passing Objects as Parameters

- The method receives a reference to an object
  - Through the reference the object can be modified



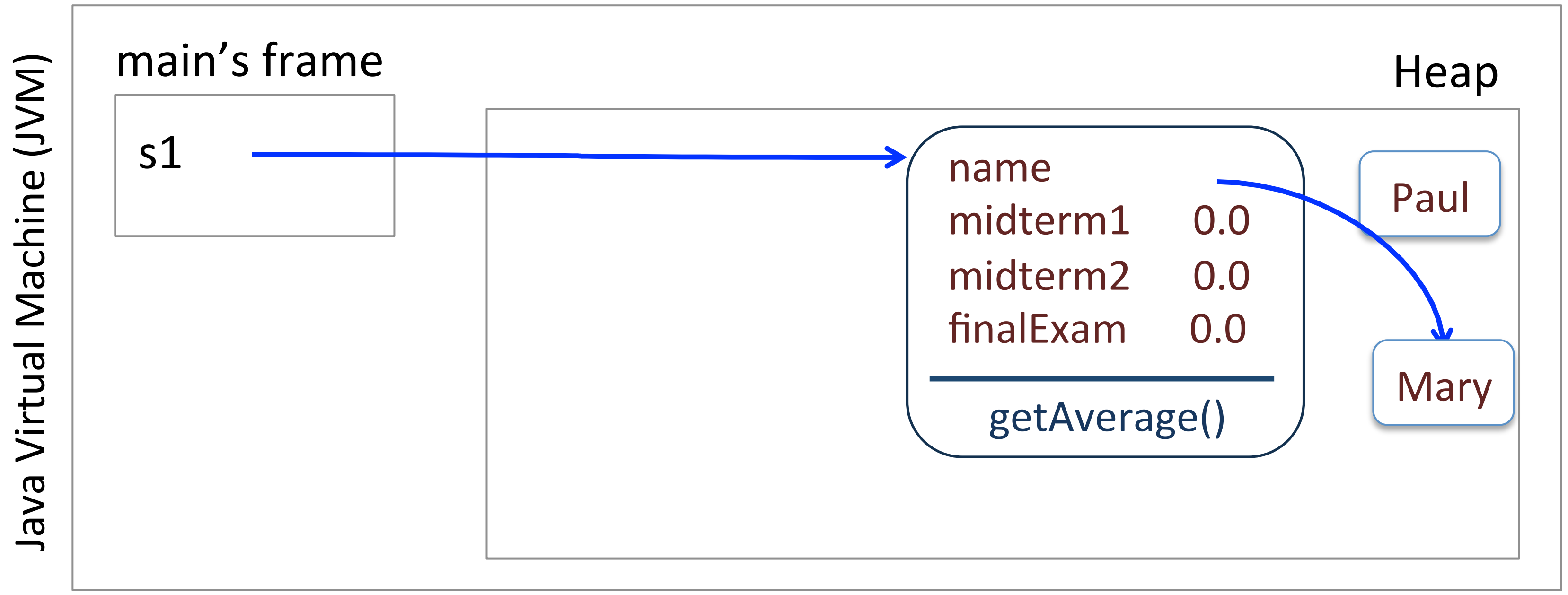
Passing Objects as Parameters

- The method receives a reference to an object
  - Through the reference the object can be modified



Passing Objects as Parameters

- The method receives a reference to an object
  - Through the reference the object can be modified
  - When the method returns the object has changed



Passing Objects as Parameters

- The method receives a reference to an object
  - The caller reference cannot be changed

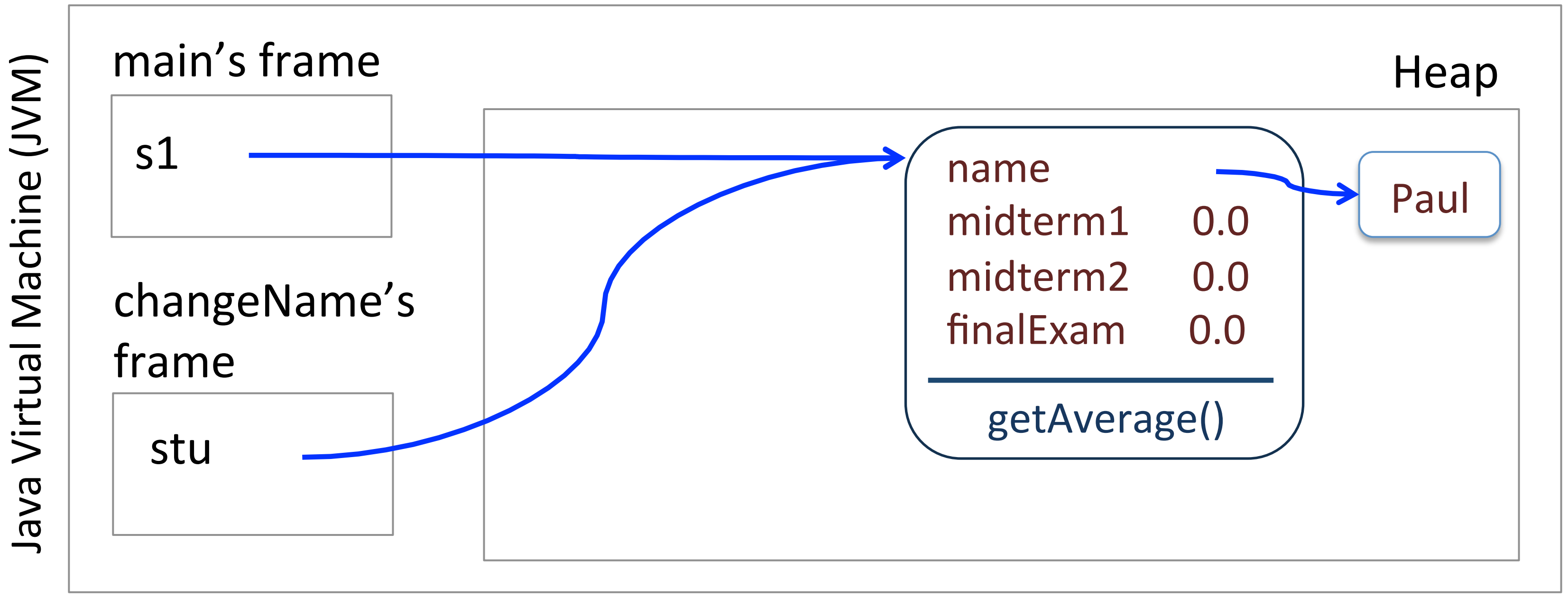
```
void changeName (Student stu) {
    stu = new Student("Mary");
}

Student s1 = new Student("Paul");
changeName(s1);
System.out.println(s1.name);

output is "Paul"
```

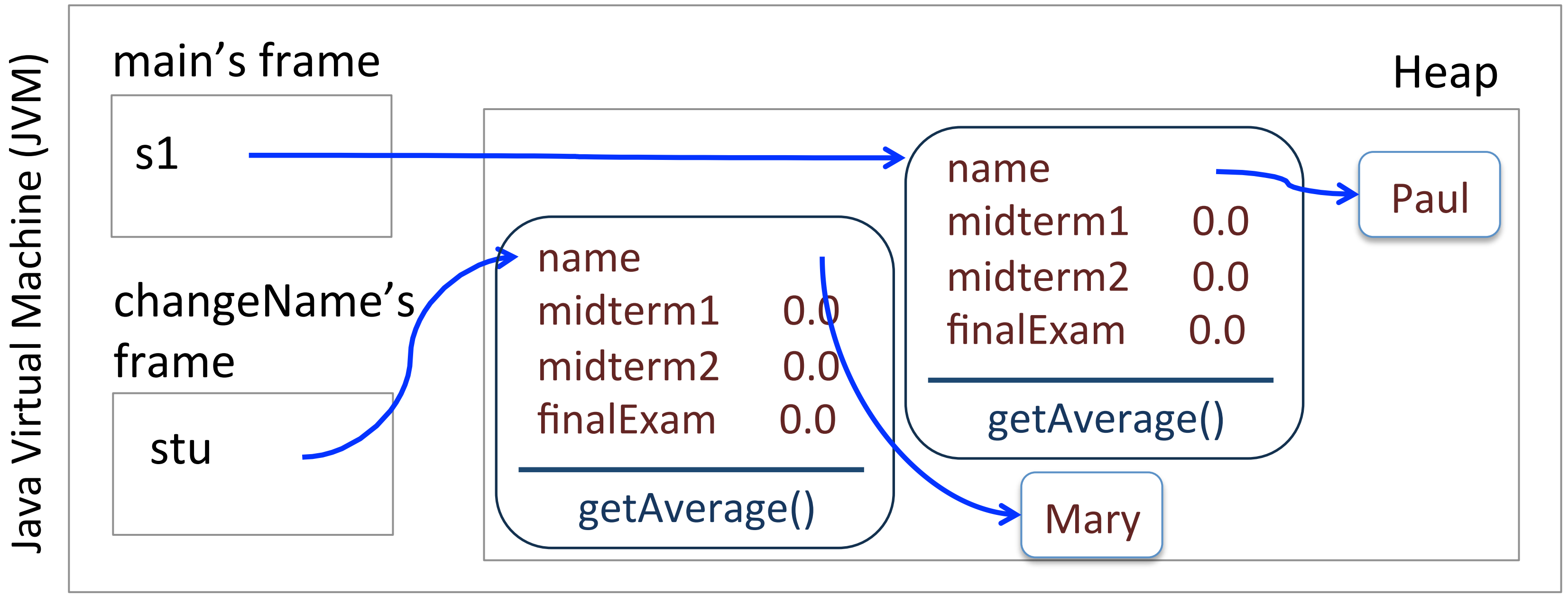
Passing Objects as Parameters

- The method receives a reference to an object
  - The caller reference cannot be changed



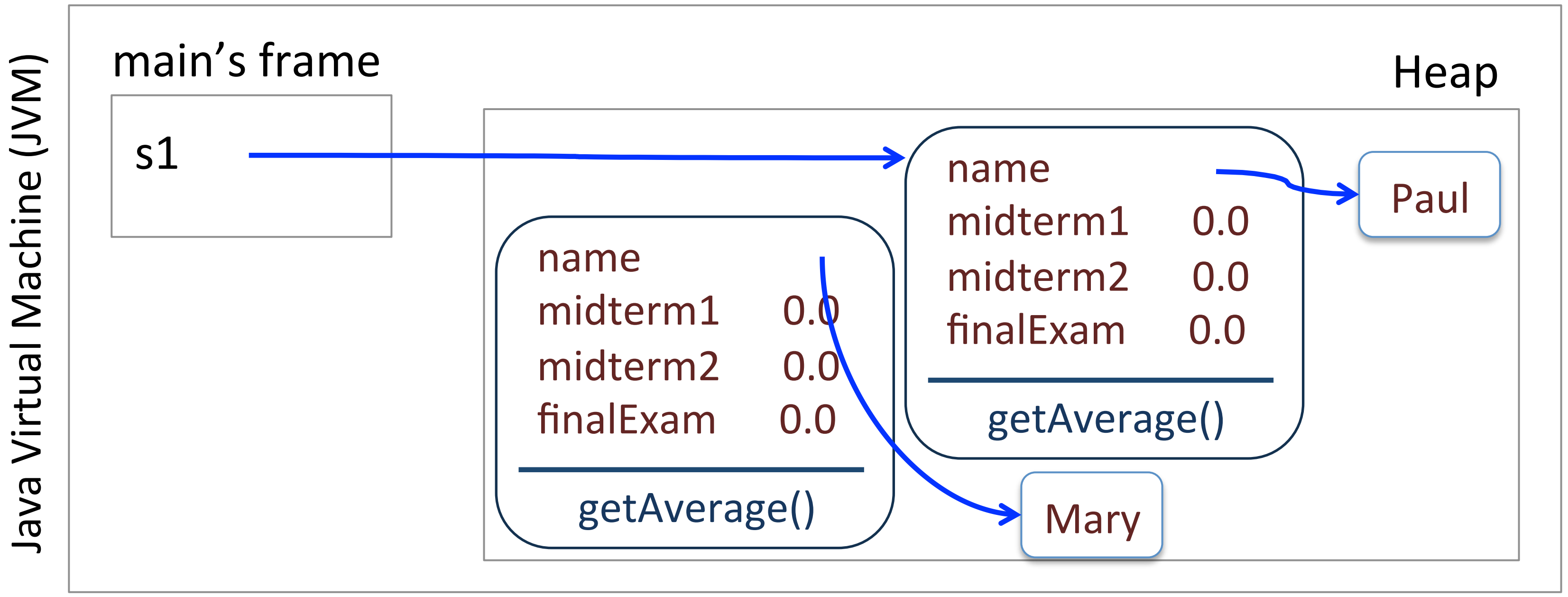
Passing Objects as Parameters

- The method receives a reference to an object
  - The caller reference cannot be changed
  - The reference inside the method points to a new object



Passing Objects as Parameters

- The method receives a reference to an object
  - The caller reference cannot be changed
  - When the method returns the reference still points to the same object



Garbage Collector

- In the previous slide the Student object “Mary” has no reference variable pointing to it: it becomes inaccessible
  - no way to find the object again: becomes garbage
- The destruction of such objects is automatic in Java
  - The garbage collector looks for objects with no references to them and remove them from the heap, freeing memory space

Static and Non Static

- Variables and Methods can be static or not
  - Static belongs to the Class
  - Non static belongs to the Object

Also called instance variables or instance methods because it belongs to an instance of a class: the object.

Also called class variables or class methods because it belongs to the Class. Only <u>ONE</u> instance of it.

Static Variable: studentCount

- Only one instance of the variable is kept in the class

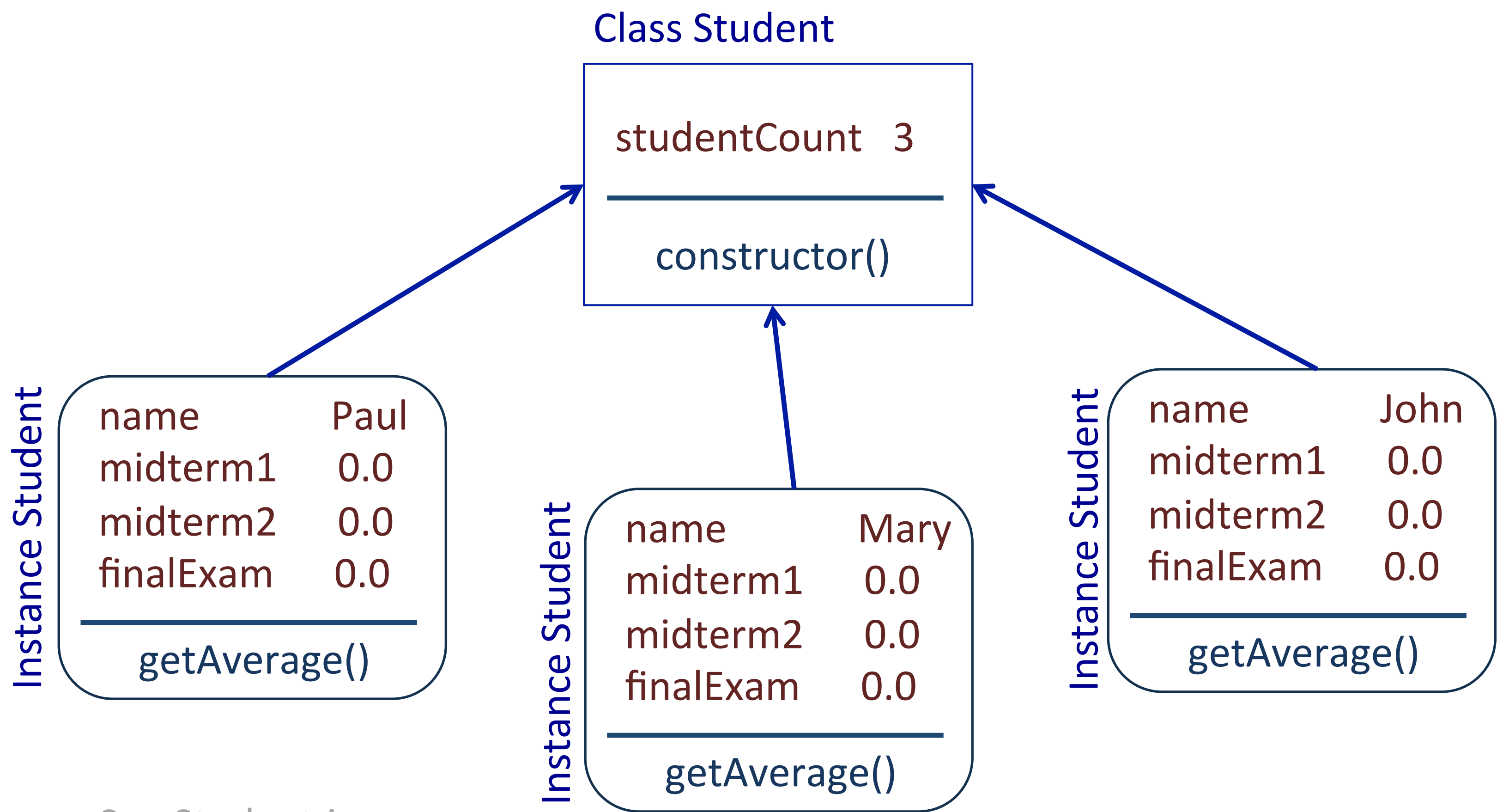
```
public class Student {
    public String name;
    public double midterm1, midterm2, finalExam;
    public static int studentCount;

    public Student (String name) {
        this.name = new String(name);
        midterm1 = midterm2 = finalExam = 0.0;
        Student.studentCount++;
    }

    public double getAverage() {
        return (midterm1+midterm2+finalExam)/3;
    }
}
```

Access through ClassName.staticVariableName

Static Variable: studentCount



See Student.java

The final modifier

- A variable can be declared final to ensure that the value stored cannot be changed after initialization

```
final int a = 0;
static final double interestRate = 0.05;
final Student s1 = new Student(“Paul”);
```

- The reference s1 cannot be changed but the object can

```
s1 = new Student(“Mary”); not allowed
s1.midterm1 = 7.9;        allowed
```

null

A reference variable can have a null value

Student s1 = null;

- s1 does not reference any objet
- if you try to use a null reference you'll get a NullPointerException error
  - s1.getAverage()

CS111
Introduction to Computer Science

Fall 2015

- Access Modifiers
- Inheritance
  - The Object class
- Polymorphism

Access Modifiers

- Public: method or variable can be accessed from anywhere (even outside the class)
- Private: method or variable can be accessed only form inside the class
- Protected: will make sense when we learn inheritance. Method or variable can be accessed from subclasses.

Setters and Getters

- Provides access to private variables

```
public class Student {
    private String name;
    ...

    public Student (String name) {
        setName(name);
        ...
    }
    public void setName (String name) {
        this.name = new String(name);
    }
    public String getName() {
        return name;
   }
}
```

Inheritance

- Extending existing classes
  - Sometimes it is useful to define a class B as "just like class A except ..."

```
public class B extends A {
   //changes and additions
}
```

- To extend an existing class

```
public class <subclass name> extends <existing class name>{
   <additional variables and methods>
   <replacement methods>
}
```

Subclass and Superclass

```
public class A {
    //variables and methods
}

public class B extends A {
    //changes and additions
}

public class C extends B {
    //changes and additions
}
```

See Vehicle.java

- B and C inherits the structure and behavior of A
  - A is a superclass of B
  - B is subclass of A
  - C is subclass of B and A

Example

```
public class Vehicle {
    int registrationNumber;
    Person owner;
    void transferOwnership(Person newOwner) {
        ...
    }
}

public class Car extends Vehicle {
    int numberOfDoors;
    ...
}

public class Truck extends Vehicle {
    int numberOfAxels;
    ...
}
```

Object Class

- If we create a class and don't explicitly make it a subclass of another, then it automatically becomes the subclass of the class Object
- Every class in a Java program is a subclass of the Object class

toString() method

- *toString()* is an instance method of the Object class
  - returns a value of type String that is a representation of the object

```
System.out.println("Vehicle " + truck);
```

See Vehicle.java, Car.java, Truck.java

this and super

- this
  - refers to “this object” or the “object that contains this method”

- super
  - refers to the superclass

instanceof

A variable that can hold a reference to an object of class A can also hold a reference to an object belonging to any subclass of A

```
Vehicle myVehicle = null;
Truck t1 = new Truck();
Car c1 = new Car();

myVehicle = t1;
myVehicle = c1;

if (myVehicle instanceof Car) {
    ...
} else if (myVehicle instanceof Truck) {
    ...
}
```

The object remembers it is an instance of a Car, not just Vehicle

2nd Midterm November 16

Monday 9:40PM to 11PM
at Scott Hall 135

If you have a legit conflict notify atjang@cs.rutgers.edu by Wednesday November 11

2nd Midterm November 16

- Nested loops
- Writing methods (method signatures, return types)
- Strings (string methods substring, indexOf, length, etc)
- Objects (calling constructors, declaring variables)
- Arrays
- 2D arrays
- OOP (class design, constructors, fields, instance methods)

Polymorphism

- The ability to have the same operation on different types of data

```
Vehicle myVehicle = null;
Truck t1 = new Truck();
Car c1 = new Car();

myVehicle = t1;
myVehicle.toString();
myVehicle = c1;
myVehicle.toString();
```

See Vehicle.java, Car.java, Truck.java

Interfaces

- Java does not allow multiple inheritance
  - One class extending multiple classes
- An interface is not a class, it defines a set of variables and methods (without implementations).
- A class that *implements* an interface must implement all methods defined by the interface

Java Virtual Machine Memory

- Rough organization of the run-time memory
  - https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html

Heap	Method Area	Native Area
Arrays and Objects	Classes and The Constant Pool for Strings	A Stack that keeps the frames (activation records) of each method that is invoked