CSSE 220

Objects

Import *SuperSimpleObjects* from repo
Import *TeamGradebook* from repo

Plan for today

- Talk about object references and box and pointer diagrams
- Talk about static methods
- Continue working on writing your own classes
- Talk about variable scope
- Get started on TeamGradebook, your new assignment

TeamGradebook

- Just a quick demo

Finishing up from last time...

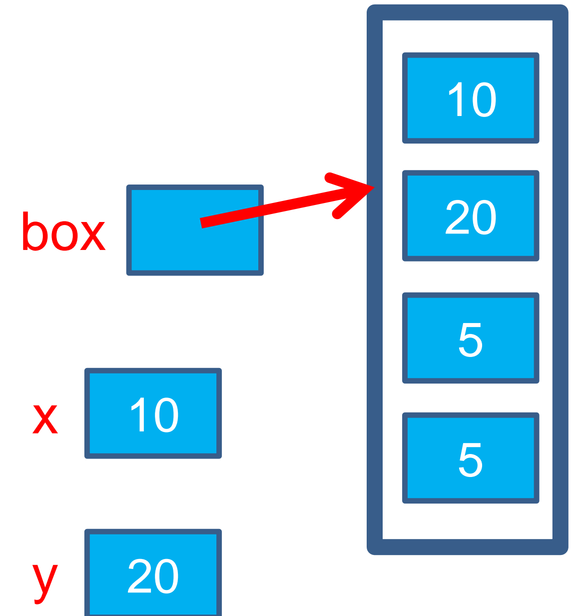
- Complete the StudentAssignments problem in the SuperSimpleObject project (or the one from last class)

Differences between primitive types and object types in Java

OBJECT REFERENCES

What Do Variables Really Store?

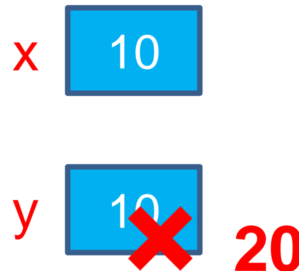
- Variables of primitive type store *values*
- Variables of class type store *references*

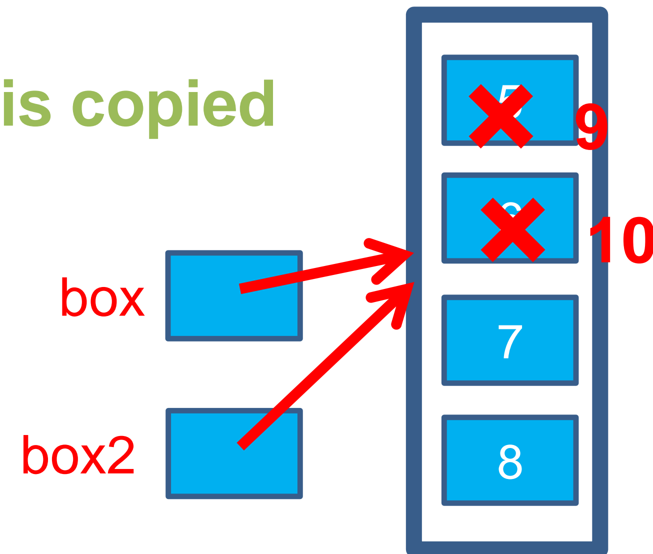



```
1. int x = 10;
2. int y = 20;
3. Rectangle box = new Rectangle(x, y, 5, 5);
```

Assignment Copies Values

- Actual value for number types
- **Reference** value for object types
  - The actual **object is not copied**
  - The **reference value** ("the pointer") **is copied**
- Consider:





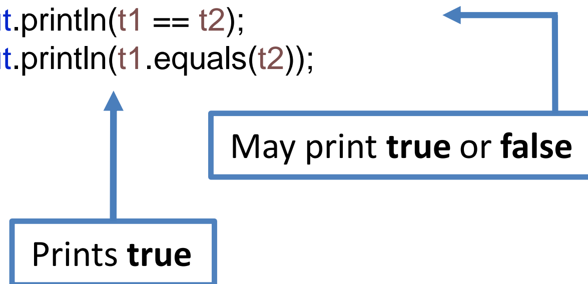

```
1. int x = 10;
2. int y = x;
3. y = 20;

4. Rectangle box = new Rectangle(5, 6, 7, 8);
5. Rectangle box2 = box;
6. box2.translate(4, 4);
```

Reference vs Value Equality

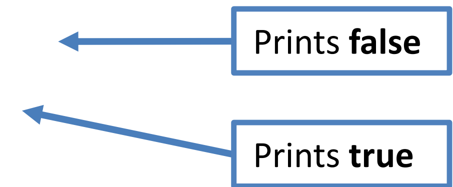
What gets printed?

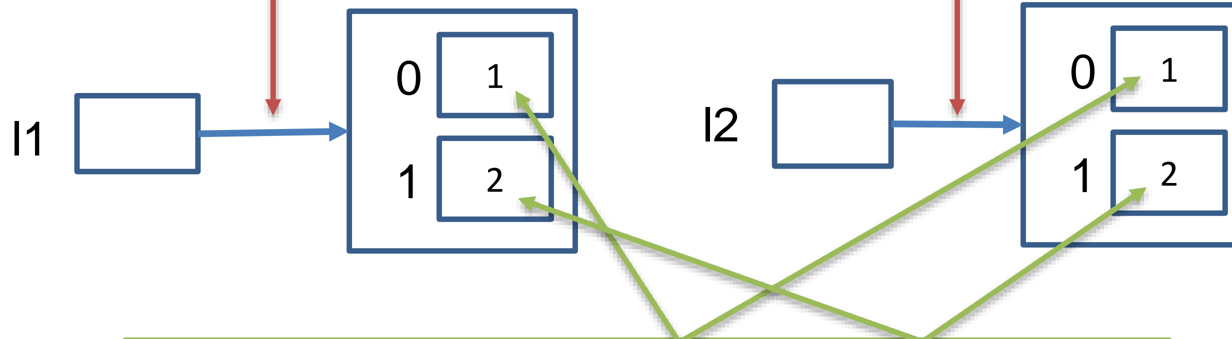
```
String t1 = "hello";
String t2 = "hello";
System.out.println(t1 == t2);
System.out.println(t1.equals(t2));
```



What gets printed here?

```
ArrayList<Integer> l1 = new ArrayList<Integer>();
l1.add(1);
l1.add(2);
ArrayList<Integer> l2 = new ArrayList<Integer>();
l2.add(1);
l2.add(2);
System.out.println(l1 == l2);
System.out.println(l1.equals(l2));
```





Q11

Box and pointer exercise

Q1 – Q10

Understanding static

STATIC

Why fields can't always be static

```java
public class Student {
 private String name;
 private char grade;

 public Student(
     String name,
     char grade){
   this.name = name;
   this.grade = grade;
 }

 @Override
 public String toString(){
   return name +
     " has a grade of "
     + grade;
 }
}
```

Client program – of Student Class

```java
public static void main(String[] args) {
  Student a = new Student("Adam", 'A');
  Student b = new Student("Bryan", 'B');
  Student c = new Student("Chris", 'C');
  System.out.println(a);
  System.out.println(b);
  System.out.println(c);
}
```

```
OUTPUT – from Client program:
Adam has a grade of A
Bryan has a grade of B
Chris has a grade of C
```

Why fields can’t always be static

```java
public class Student {
 private String name;
 private static char grade;

 public Student(
     String name,
     char grade){
   this.name = name;
   Student.grade = grade;
 }

 @Override
 public String toString(){
   return name +
     " has a grade of "
     + grade;
 }
}
```

Client program – of Student Class

```java
public static void main(String[] args) {
  Student a = new Student("Adam", 'A');
  Student b = new Student("Bryan", 'B');
  Student c = new Student("Chris", 'C');
  System.out.println(a);
  System.out.println(b);
  System.out.println(c);
}
```

OUTPUT – from Client program:

```
Adam has a grade of C
Bryan has a grade of C
Chris has a grade of C
```

Static means there's only one instance of a field/method for all instances of a class that's created. So when you change a grade, it changes for all instances.

When do we make methods static?

- Utility Methods
  - Things like abs, sqrt, etc.
  - Don’t need an instance of a class to run them
- How do I know?
  - No references to non-static fields/methods
  - No “this” keyword used in method

When do we make fields static?

- Never
  - Seriously, this is disallowed for all the code you submit in CSSE220 (exception: CONSTANTS)
  - It makes your designs worse
- If it wasn’t disallowed, when would you use it?
  - Very rarely for memory efficiency, state that can’t be duplicated, or really meta code
  - BUT even professional programmers misuse static and cause themselves major problems
  - They’ll talk about some positive uses in CSSE374

```java
public class Car {

    private double mileage;

    //other stuff

    public double getMilesTravelled() {
        return this.mileage;
    }

    public static double convertMilesToKm(double numberOfMiles) {
        return numberOfMiles * 1.609344f;
    }

}
```

```java
//Elsewhere in a client program of Car class
```

```java
//requires you to have a car object
Car myCar = new Car();
// getMilesTravelled requires you to have a car object
System.out.println(myCar.getMilesTravelled());//output depends on code
//convertMilesToKm can be called on the class Car itself
System.out.println(Car.convertMilesToKm(77));//output is 123.919488
```

```java
public class Bicycle {

    private int speed;
    private static int numCreated = 0;

    public Bicycle(int speed) {
        this.speed = speed;
        Bicycle.numCreated++;
    }
    public int getSpeed() {
        return this.speed;
    }
    public static int getNumCreated() {
        return Bicycle.numCreated;
    }
}
```

```java
// Client does not need Bicycle object for calling getNumCreated
System.out.println(Bicycle.getNumCreated());
Bicycle myBike1 = new Bicycle(18);
Bicycle myBike2 = new Bicycle(1);
System.out.println(Bicycle.getNumCreated() + " " + myBike1.getSpeed());
```

0

2 18

Q12 - Q16

Two ways to do one thing: Static and Instance

- Consider the Point class we used as a Quiz
- Let's write code to enable the follow to run

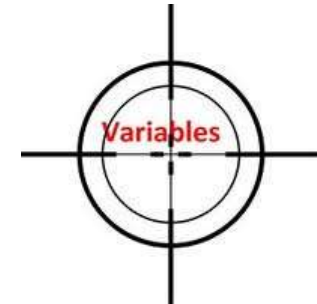
```
Point a = new Point( 0,0 );
Point b = new Point( 3, 4);
System.out.println(   Point.distanceBetween(a,b) );
System.out.println(  a.distanceTo( b ) );
```

Live code

Why **this**?

- Fills same role as “self” in python
- Keep track of what variables belong to the instance of the class
- (Object inside which code is running)
- Helps differentiate instance and local variables
- Variable Scope (next)

Variable Scope



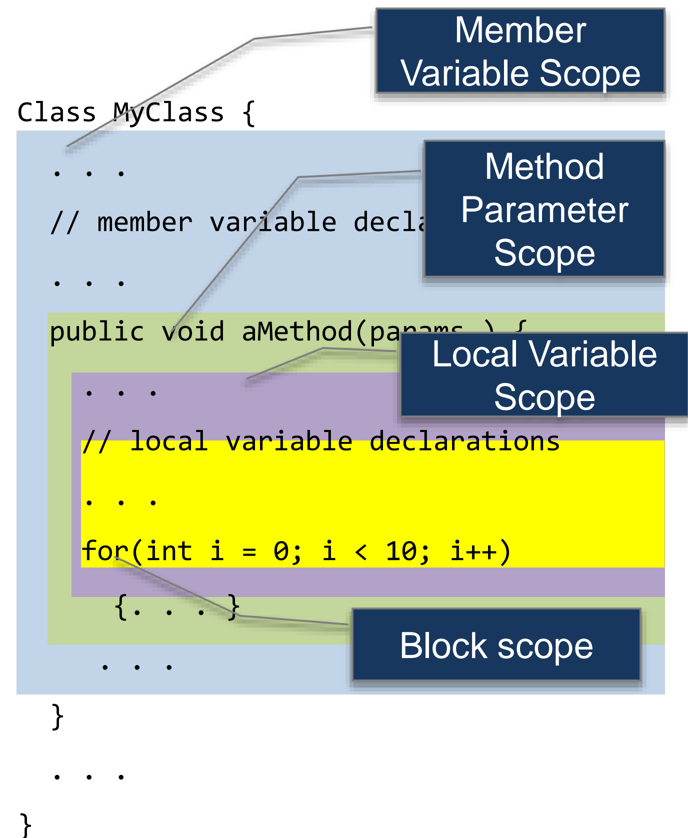
Scope is the region of a program in which a variable can be accessed

- *Parameter scope:* the whole method body
- *Local variable scope:* from declaration to block end

```
public double myMethod() {
    double sum = 0.0;
    Point2D prev = this.pts.get(this.pts.size() - 1);
    for (Point2D p : this.pts) {
      sum += prev.getX() * p.getY();
      sum -= prev.getY() * p.getX();
      prev = p;
    }
    return Math.abs(sum / 2.0);
}
```

Member Scope (Field or Method)

- ***Member scope:*** anywhere in the class, including *before* its declaration
  - Lets methods call other methods later in the class
- `public static` class members can be accessed from outside with "class qualified names"
  - `Math.sqrt()`
  - `System.in`



Overlapping Scope and Shadowing

```
public class TempReading {
        private double temp;

        public void setTemp(double temp) {
                this.temp = temp;

        }
        // …
}
```

What does this “temp” refer to?

Always qualify field references with `this`. It prevents accidental shadowing.

Exercise

- Start working on the TeamGradeBook homework. Try to finish the code for both add-student, add-absence and get-absences today
- If you are confused about what to do, get help!