

# CSSE 220

Objects

Check out *SuperSimpleObjects* and *TeamGradebook* from SVN

# Plan for today

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

# Exercise

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

# TeamGradebook

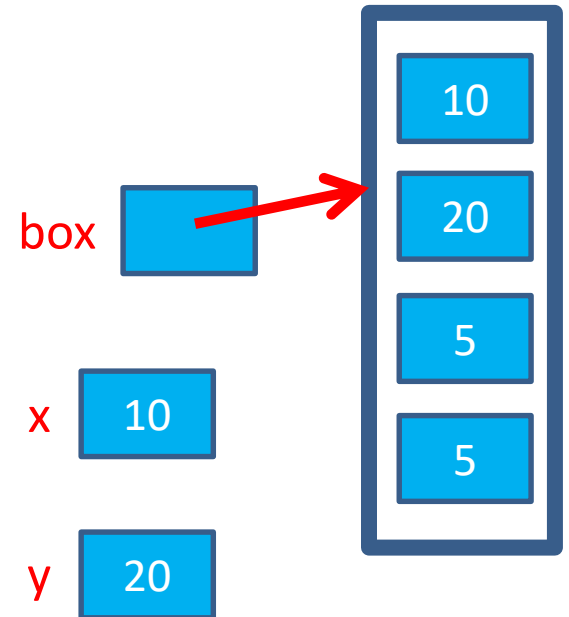
- Just a quick demo

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*



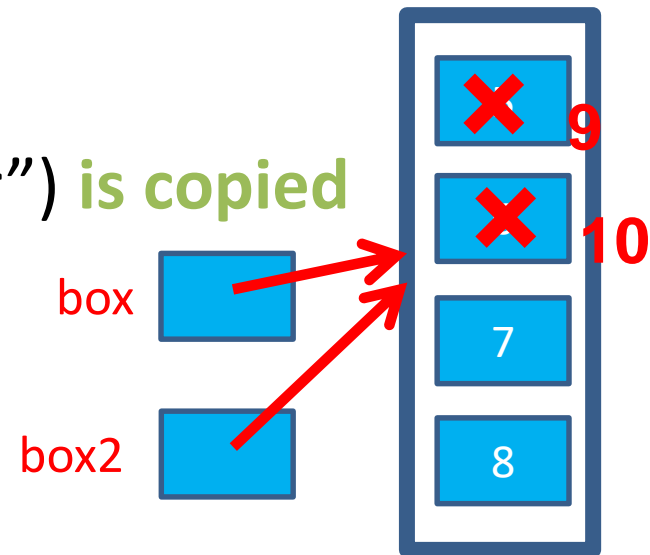
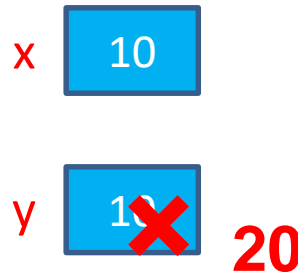
```
1. int x = 10;  
2. int y = 20;  
3. Rectangle box = new Rectangle(x, y, 5, 5);
```

# 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));
```

Prints **true**

May print **true** or **false**

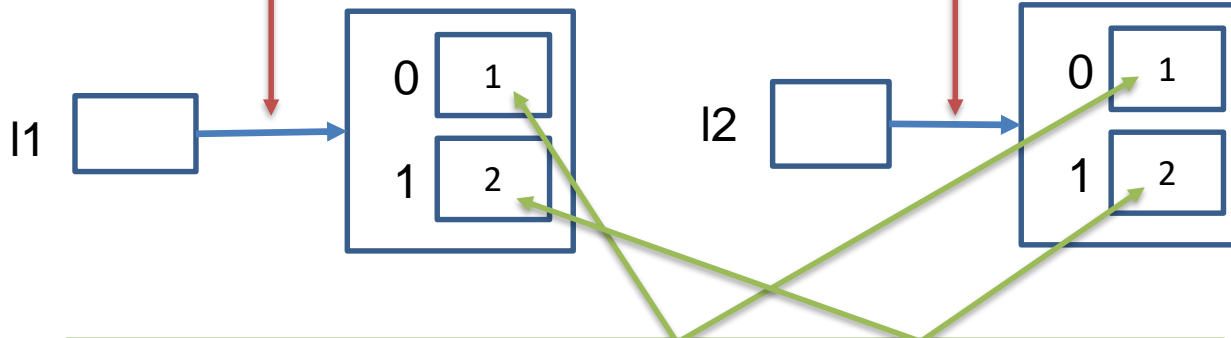
## What gets printed here?

```
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));
```

Prints **false**

Prints **true**

`==` operator compares references of two objects



`equals()`, in general, compares values of two objects



# Boxes and lines exercise

Understanding static

**STATIC**

# Why fields can't always be static

```
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;  
    }  
}
```

```
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:

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

# Why not make the grade static?

```
public class Student {  
    private String name;  
    private static 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;  
}  
}
```

```
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:

```
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 every instance of a class that's created. So when you change a grade, they all change.

# 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

```
public class Car {  
  
    double mileage;  
  
    //other stuff  
  
    public double getMilesTravelled() {  
        return this.mileage;  
    }  
  
    public static double convertMilesToKm(double numberOfMiles) {  
        return numberOfMiles * 1.609344f;  
    }  
  
}  
  
//Elsewhere...  
  
//requires you to have a car object  
Car myCar = new Car();  
//requires you to have a car object  
System.out.println(myCar.getMilesTravelled());//output depends on code  
//can be called on the class Car itself  
System.out.println(Car.convertMilesToKm(77));//output is 123.919488
```

```
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;
```

```
    }
```

```
}
```

```
//No requirement to have a Bicycle yet...
```

```
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
```

# Exercise

- Complete the StudentAssignments problem in the SuperSimpleObject project
- 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!