Objects Recap

and

Pointers!

AKA: Why is .equals different than ==?

Find a partner for the day!
public class Robot {

}
public class Robot {

}

Instance variables and methods go inside the curly braces (so that they “belong to” the class)

By yourself:
• Add the code that defines three instance variables: one int, one String, and one of whatever type you’d like. Name them whatever you want.

With a partner:
• Compare your results. Do you agree on the syntax?
public class Robot {
    private int numberOfWheels;
    private String name;
    private double speed;
    private TYPE NAME;

    (In general. Note that there’s no value yet!)

}
public class Robot {
    private int numberOfWheels;
    private String name;
    private double speed;
    private TYPE NAME;
    (In general. Note that there’s no value yet!)
}

To let code outside Robot use these data, we need getter methods.

By yourself: add a getter for one of your instance variables. Then compare with your partner.
public class Robot {
    private int numberOfWheels;
    private String name;
    private double speed;

    public double getSpeed() {
        return speed;
    }

    public String getName() {
        return name;
    }
}

By convention, getters are named getWhatever().

Note “public”

public double getSpeed() {
    return speed;
}

Because getSpeed() is inside Robot, it can use the (private) speed instance variable

public return type name(parameters)

public String getName() {
    return name;
}

}
public class Robot {
    private int numberOfWheels;
    private String name;
    private double speed;

    public double getSpeed() {
        return speed;
    }

    public String getName() {
        return name;
    }

    // By yourself: add a setter for one of your instance variables. Then compare with your partner.
}
public class Robot {
    private int numberOfWheels;
    private String name;
    private double speed;

    public double getSpeed() {
        return speed;
    }

    public void setSpeed(double newSpeed) {
        speed = newSpeed;
    }
}

void means “doesn’t return anything”

Setters change the internal state of an object.
public class Robot {
    private int numberOfWheels;
    private String name;
    private double speed;

    public double getSpeed() {
        return speed;
    }

    public void setSpeed(double newSpeed) {
        speed = newSpeed;
    }
}

void means “doesn’t return anything”

Setters change the internal state of an object.

By yourself: write a constructor for your class. Then compare with your partner.
public class Robot {
    private int numberOfWheels;
    private String name;
    private double speed;

    public Robot(int w, String n, double s) {
        numberOfWheels = w;
        name = n;
        speed = s;
    }
}
In the usual way: match lefts to rights.

And then: put the rights into an order that will run. And figure out what it prints.
In the usual way: match lefts to rights.

And then: put the rights into an order that will run. And figure out what it prints.
Pointers!

```java
int x = 5;
Robot pr2 = new Robot(8, "PR2", 0.5);
```
int x = 5;
Robot pr2 = new Robot(8, "PR2", 0.5);

Primitives take up very little memory (each)

Objects (potentially) take up lots of memory (each). (So do arrays!)
int x = 5;
Robot pr2 = new Robot(8, "PR2", 0.5);

Primitives take up very little memory (each)

Objects (potentially) take up lots of memory (each).

Every object variable is storing a pointer.

A pointer. (also memory-cheap)
Pointers!

```c
int x = 5;
int y = x;
int z = x;
```

In the usual way: fill the boxes in. Yes, it’s really easy.
Pointers!

Robot pr2 = new Robot(8, "PR2", 0.5);
Robot a = pr2;
Robot b = pr2;

In the usual way: finish the picture.
Pointers!

Robot pr2 = new Robot(8, "PR2", 0.5);
Robot a = pr2;
Robot b = pr2;

In the usual way: finish the picture.

Objects (potentially) take up lots of memory (each).
So you want to be able to not copy them if you don’t have to!
Pointers!

System.out.println("PR2 speed: "+ pr2.getSpeed());
a.setSpeed(10);
System.out.println("PR2 speed: "+ pr2.getSpeed());

In the usual way: what prints?

8
"PR2"
0.5
11
"foo"
...
.equals

Snarf Strings

Before you run it, predict what will print. Compare that with your partner.

Then run it, and see if you were right.
Snarf Strings

Before you run it, predict what will print. Compare that with your partner.

Then run it, and see if you were right.

I got:

```java
!=
.equals
b == c
.equals(b)
```
String a = new String("Hello");
String b = new String("Hello");

if (a == b) {
    System.out.println("==");
} else {
    System.out.println("!=");
}

if (a.equals(b)) {
    System.out.println(".equals");
} else {
    System.out.println("not .equals");
}

String c = b;

if (c == b) {
    System.out.println("c == b");
} else {
    System.out.println("c != b");
}

if (c.equals(b)) {
    System.out.println("c.equals(b)");
}

== compares pointers.
String a = new String("Hello");
String b = new String("Hello");

if (a == b) {
    System.out.println("==");
} else {
    System.out.println("!=");
}

if (a.equals(b)) {
    System.out.println(".equals");
} else {
    System.out.println("not .equals");
}

String c = b;

if (c == b) {
    System.out.println("c == b");
} else {
    System.out.println("c != b");
}

if (c.equals(b)) {
    System.out.println("c.equals(b)");
} else {
    System.out.println("not c.equals(b)");
}

== compares pointers.

.equals compares values
String c = b;

if (c == b) {
    System.out.println("c == b");
} else {
    System.out.println("c != b");
}

if (c.equals(b)) {
    System.out.println("c.equals(b)");
} else {
    System.out.println("not c.equals(b)");
}

= copies pointers.
String c = b;

if (c == b) {
    System.out.println("c == b");
} else {
    System.out.println("c != b");
}

if (c.equals(b)) {
    System.out.println("c.equals(b)");
} else {
    System.out.println("not c.equals(b)");
}

String a = new String(
    "Hello"
);
String b = new String(
    "Hello"
);
String c = b;

= copies pointers.

.clone copies values.
Jotto!

Hangman due at midnight.

New assignment coming out today: Jotto!
   (Due Monday)

Three new APTs coming out today!
   (Due Friday)

Demo time!

http://goo.gl/9tx4P