Java Training

Day 4: Looping Constructs, Class Objects, Instances and Subclasses

Download today's slides:
go/java+espresso-training/day4

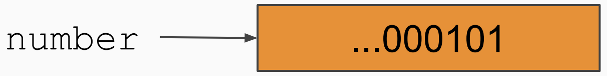
Primitive Types & Object Types in Java (and testing for equality)

The "==" operator compares the bits

Primitive Types

Define how Java interprets the value of bits at some memory location

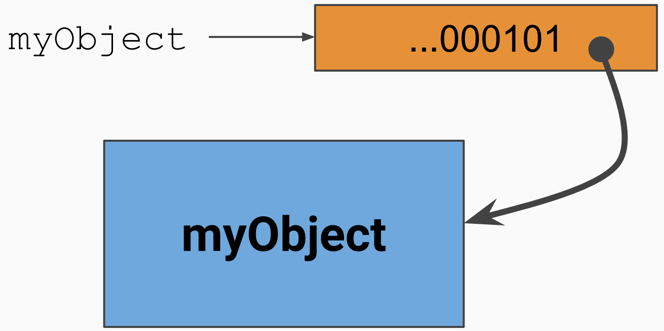
```
int number = 5;
```



Object Types

The bits at a memory location are interpreted as a pointer (or reference) to information about an object

```
SomeClass myObject = new SomeClass();
```



Compare objects using equals()

```
someObject.equals(anotherObject)
```

While Loops

While loops operate on a `boolean` condition (just like `if`)

Assuming an `int` called `count`

```
while (count < 10) {
    obj.doStuff();
}
```

This will continue to loop until `count` is equal-to or greater-than `10`

If `doStuff()` never alters the value of `count`, this will loop forever

doStuff() needs to increment count:

`count = count + 1` or `count++`

or just change the value to more than 9

There are also `break` and `continue` statements for short-circuiting loop execution

For-Loops

For loops include both initialization and incrementing within the loop syntax.

Assume `i` is an `int`

```
for (i=0; i < 6; i++){
    obj.doStuff();
}
```

This will execute `doStuff()` 6 times and doesn’t depend on `doStuff()` changing the loop condition to prevent infinite looping.

None of the for loop parameters are required.
This is an infinite loop:

```
for (;;){
}
```

The `break` and `continue` statement also work with `for` loops:

```
for (;;){
    break;
}
```

(executes only once)

‘Repeater’ Problem

Create a new application ‘Repeater’ that takes a single numeric argument *N* and repeats a text message *N* times

For example

```
> java Repeater 3
```

Will output:

```
Repeating
Repeating
Repeating
```

Repeater, One Possible Solution

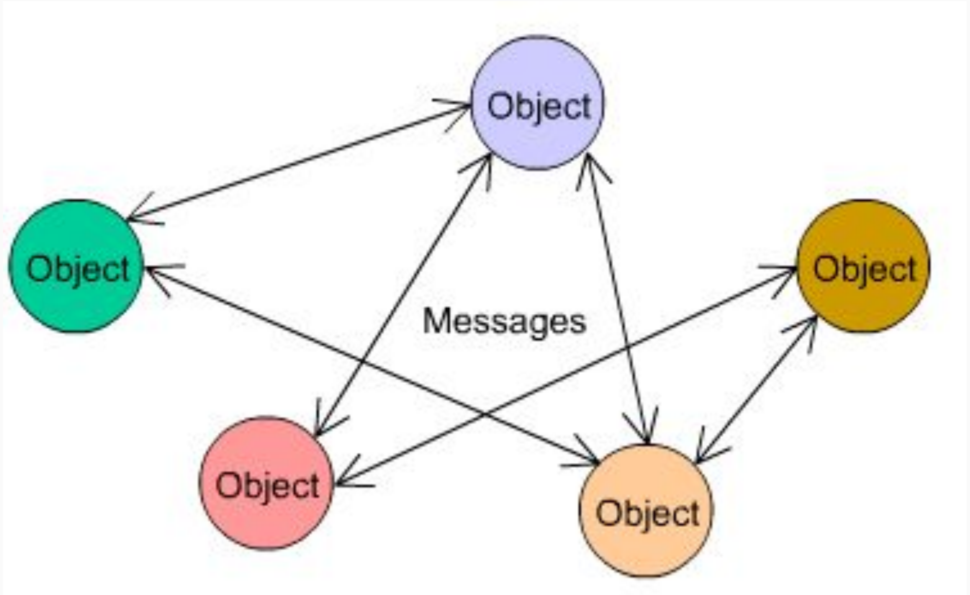
```
public class Repeater {
      public static void main(String[] args){
            int n = Integer.valueOf(args[0]);
            int i;

            for (i=0; i< n; i++){
                  System.out.println("Repeater");
            }
      }
}
```

This is one possible solution. Did anyone do the conditional differently?

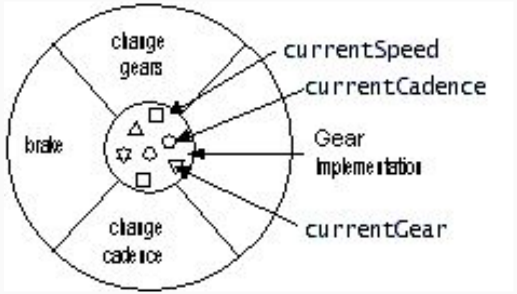
There is solution that does without the second `int` variable `i`

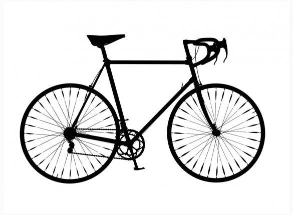
Back to the “message-passing” model



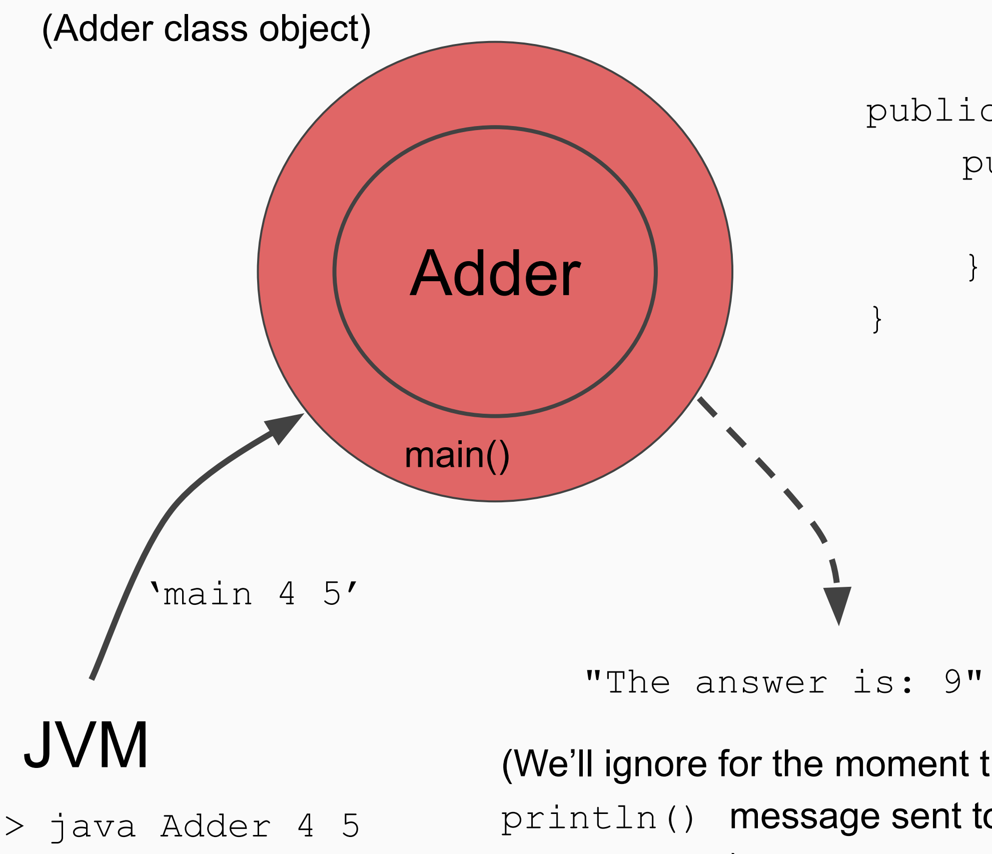
Interaction of objects via message passing

How does message-passing relate to the ‘Adder’ application?





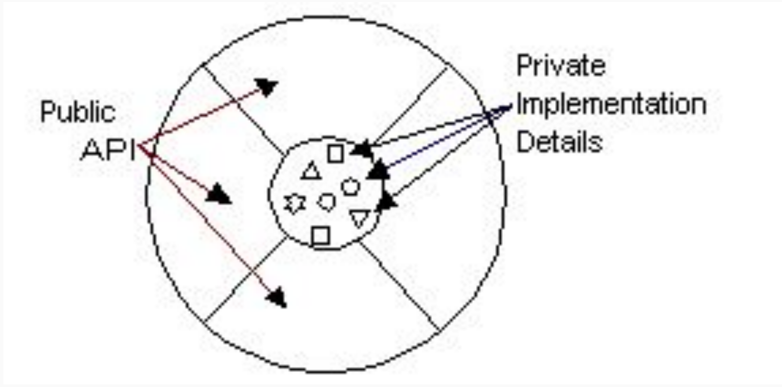
‘Adder’ described in terms of Message-Passing



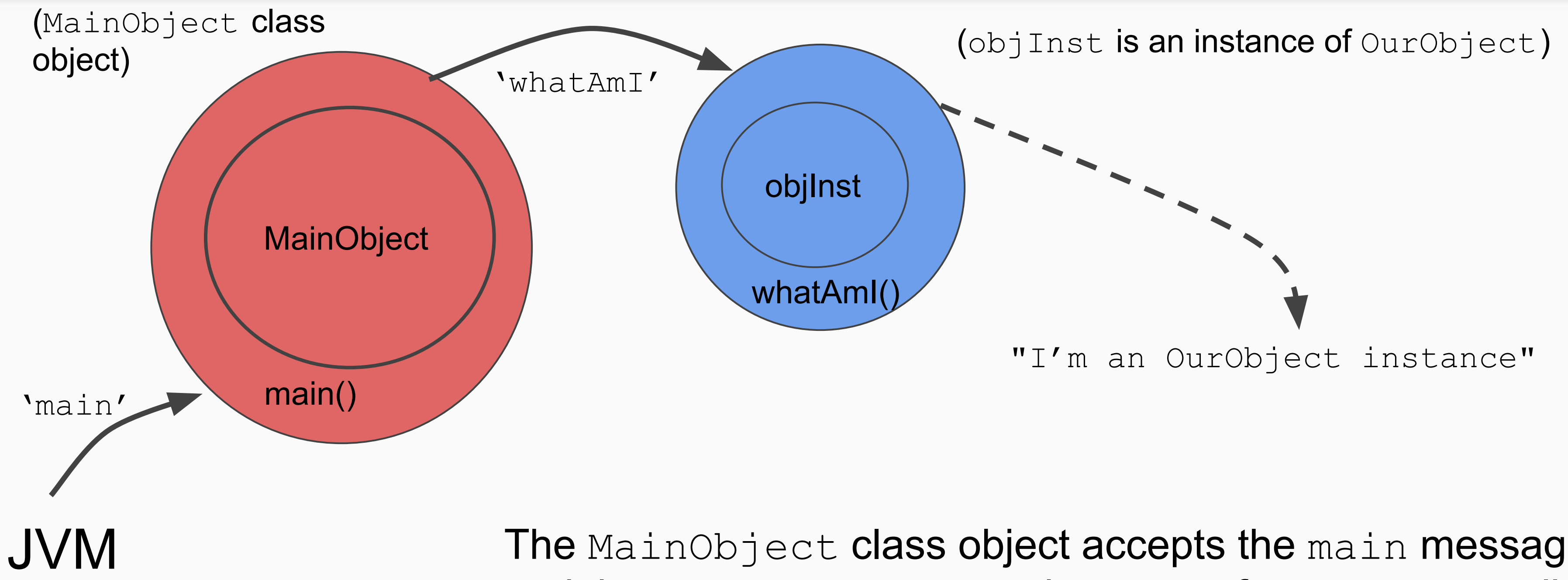
```
> java Adder 4 5
```

```
public class Adder {
     public static void main(String[] args){
          ...
     }
}
```

(We’ll ignore for the moment the `println()` message sent to `System.out`)



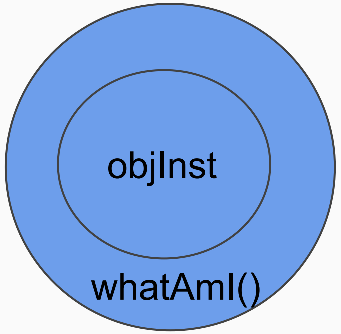
A more interesting Message-Passing System



(MainObject class object)
'whatAmI'
(objInst is an instance of OurObject)
MainObject
main()
objInst
whatAmI()
'main'
"I'm an OurObject instance"
JVM

The MainObject class object accepts the main message and, in response, creates an instance of OurObject called objInst and sends it a whoAmI message

The OurObject class



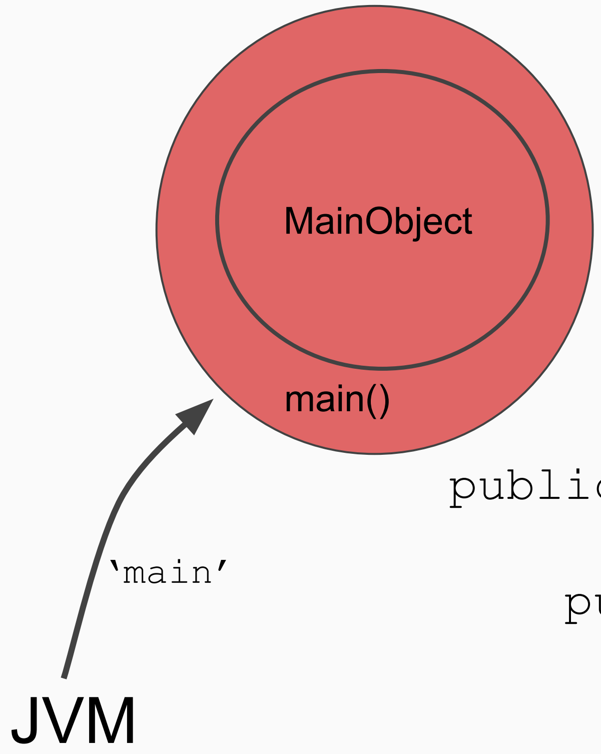
Instances of `OurObject` class will accept the `whatAmI` message and generate the appropriate output

Create a new file, `OurObject.java`

```
public class OurObject {

    public void whatAmI(){
            System.out.println("I'm an OurObject instance");
    }
}
```

MainObject Implementation



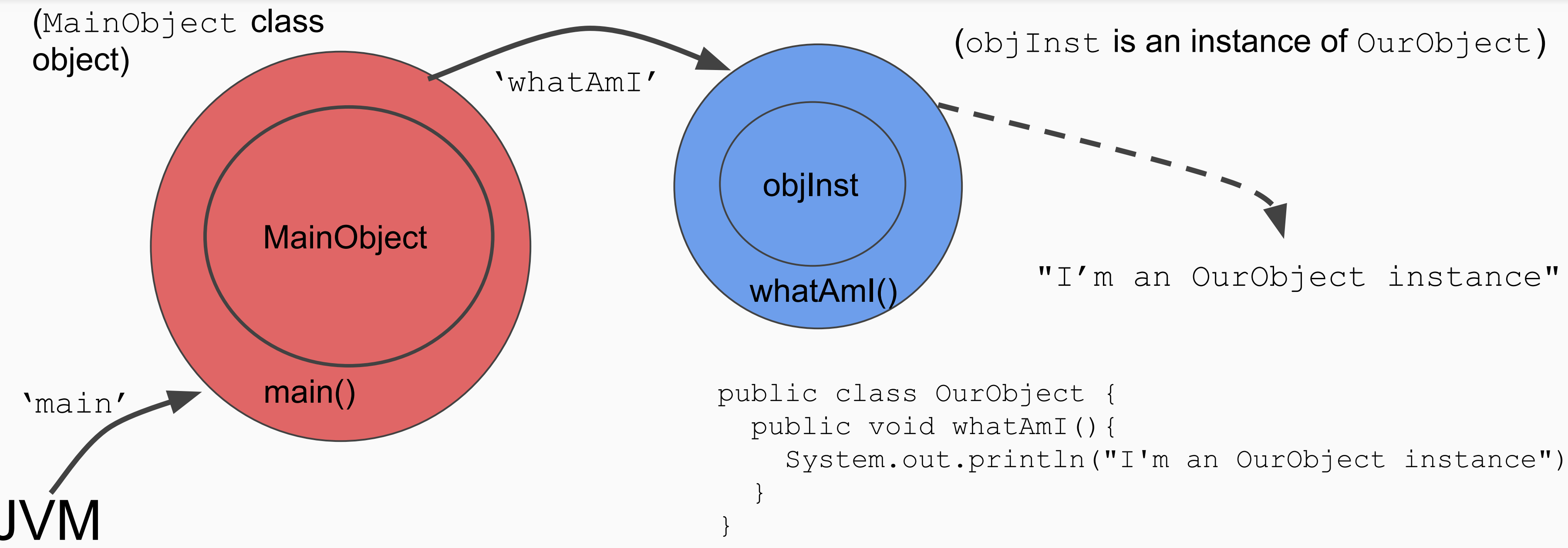
The `MainObject` class object accepts the `main` message and, in response, creates an instance of `OurObject` and sends it a `whatAmI` message

Create a new file, `MainObject.java`

```
public class MainObject {

    public static void main(String[] args){
        OurObject objInst = new OurObject();
        objInst.whatAmI();
    }
}
```

The Complete System


(MainObject class object)
'whatAmI'
(objInst is an instance of OurObject)
MainObject
main()
objInst
whatAmI()
"I'm an OurObject instance"
'main'
JVM

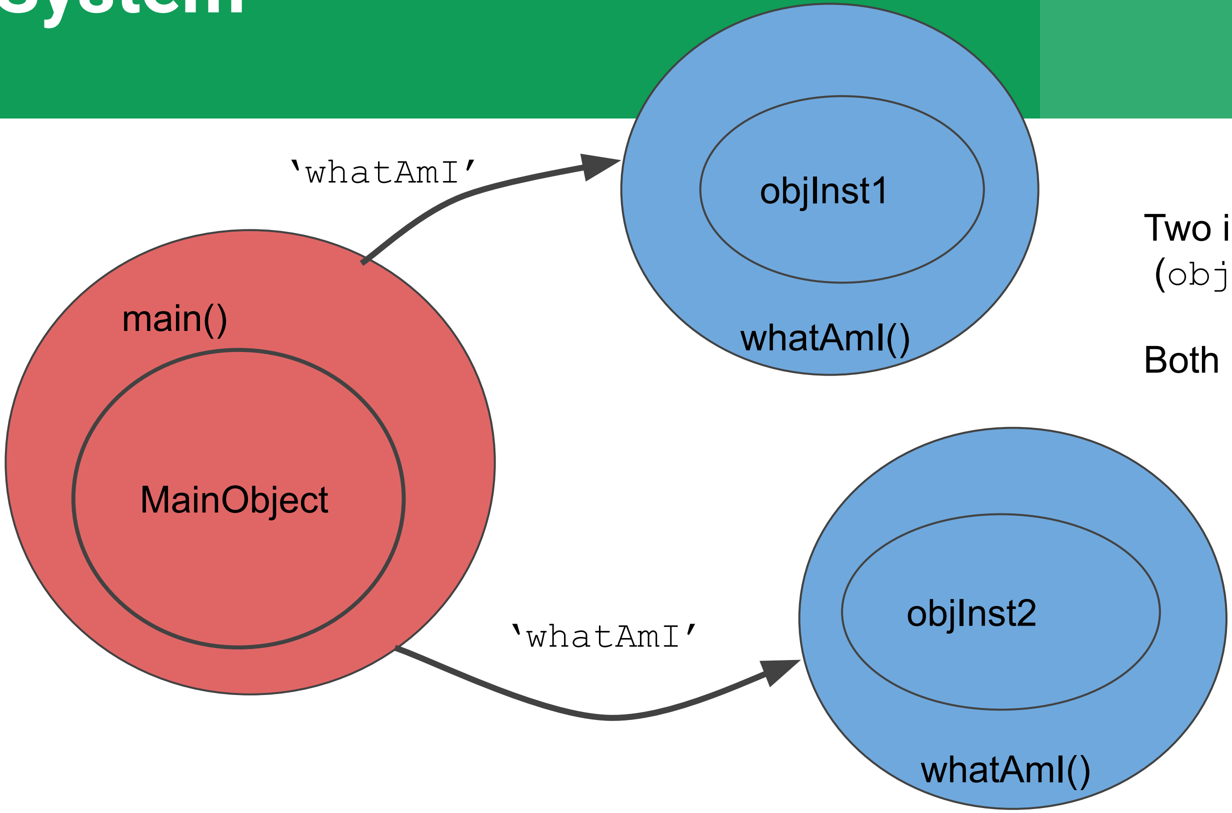
```
public class OurObject {
  public void whatAmI(){
    System.out.println("I'm an OurObject instance");
  }
}
```

```
public class MainObject {
   public static void main(String[] args){
     OurObject objInst = new OurObject();
     objInst.whatAmI();
  }
}
```

It should all now work:

```
> javac MainObject.java
> java MainObject
```

Problem #1
Update MainObject to create this System



Two instances of `OurObject` (`objInst1` and `objInst2`)

Both receive a `whoAmI` message

Problem #1
Possible Solution

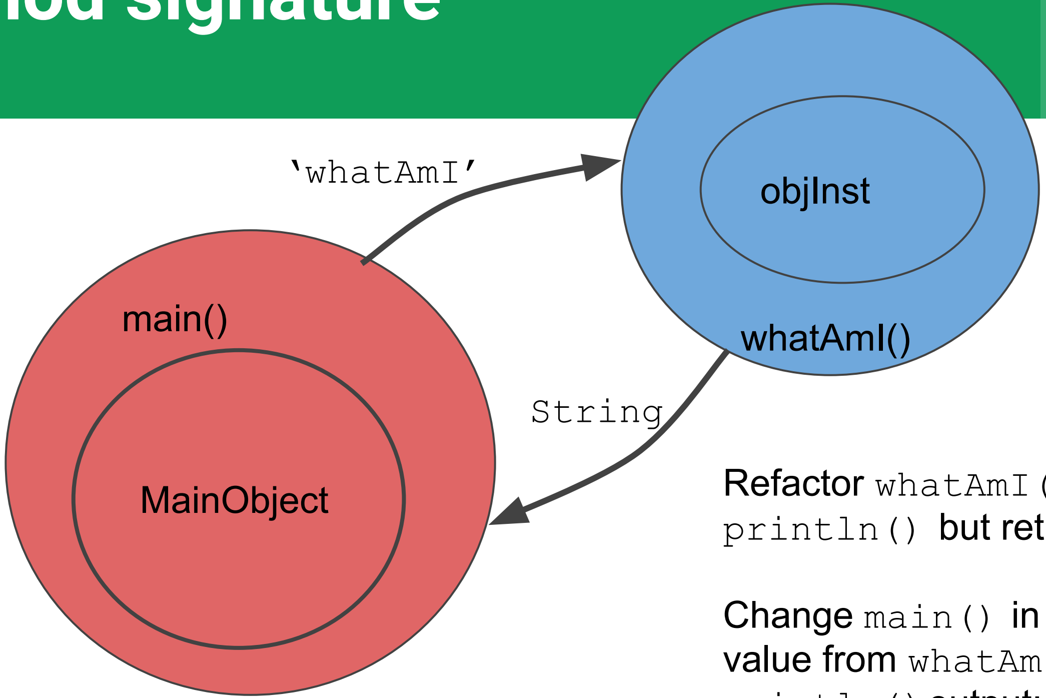
```
public class MainObject {

    public static void main(String[] args){
            OurObject objInst1 = new OurObject();
            OurObject objInst2 = new OurObject();

            objInst1.whatAmI();
            objInst2.whatAmI();
    }
}
```

The `OurObject` implementation stays the same

Problem #2
Adding a return value to a method signature



One instance of `OurObject` (`objInst1` and `objInst2`)

Refactor `whatAmI()` so that it no longer calls `println()` but returns the String "`OurObject`"

Change `main()` in `MainObject` so that it uses the return value from `whatAmI()` to generate the `println()` output:

```
"I'm an OurObject instance"
```

To return `someValue` from a method:

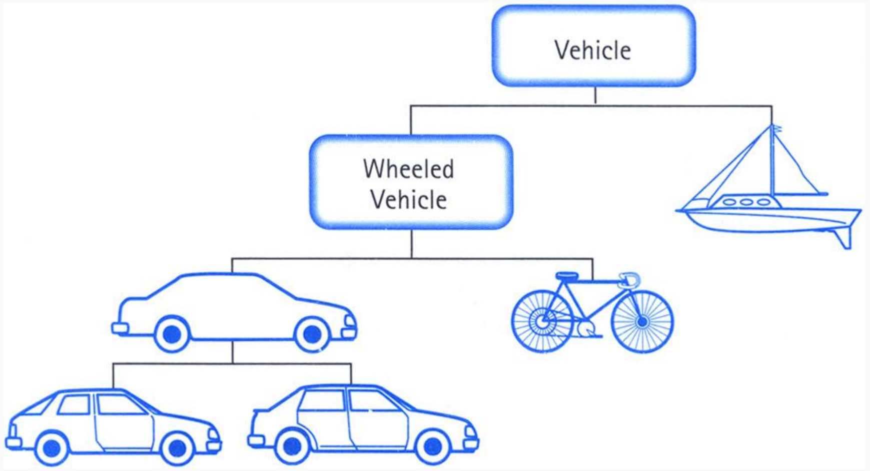
```
    return someValue;
```

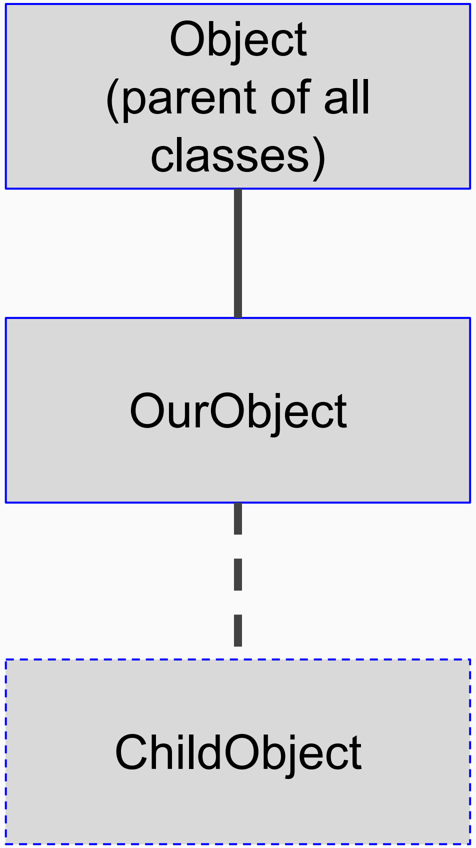
Problem #2
Possible Solution

```
public class OurObject {
    public String whatAmI(){
        return "OurObject";
    }
}


 public class MainObject {
        public static void main(String[] args){
                OurObject objInst = new OurObject();
                String s = objInst.whatAmI();
                System.out.println("I am an " + s + " instance");
         }
 }
```

Inheritance with OurObject





`OurObject` is the beginning of a new class hierarchy where `ChildObject` will be a subclass of `OurObject`

```
public class ChildObject extends OurObject {
}
```

Create a new file `ChildObject.java` and add the code above

Problem #3
Make an instance of ChildObject

Change `MainObject` so that it creates an instance of `ChildObject` instead of OurObject.

Run MainObject:

```
> java MainObject
```

What happens?

Problem #3 solution

`MainObject.java` should look something like:

```
public class MainObject {
        public static void main(String[] args){
                ChildObject objInst = new ChildObject();
                String s = objInst.whatAmI();
                System.out.println("I am an " + s + " instance");
        }
}
```