

Return Values

Method Return Values

Parameters passed to methods are not changed.

What is the **output** of the code below?

```
int x = 5;

example(x);
System.out.println(x);

void example(int x) {
    x = x + 1;
    System.out.println(x);
}
```

Method Return Values

Parameters passed to methods are not changed.
What is the **output** of the code below?

```
int x = 5;

example(x);
System.out.println(x);

void example(int x) {
    x = x + 1;
    System.out.println(x);
}
```

6
5

Method Return Values

You can return any data type (including objects) from a method.

```
int x = 5;  
x = example(x);
```

```
int y = example(121);
```

```
int example(int x) {  
    x = x + 1;  
  
    return x;  
}
```

Method Return Values

You can return any data type (including objects) from a method.

```
Student billy = createStudent();
```

```
Student createStudent() {  
    Student s = new Student();  
    s.totalGrade = 0;  
    s.year = 1;  
    s.generateID();  
  
    return s;  
}
```

Method Return Values

Objects have an interesting behavior.
What is the **output** of the code below?

```
Team x = new Team("Justice League");
```

```
example(x);
```

```
System.out.println(x.name);
```

```
void example(Team t) {  
    t.name = "Avengers";  
}
```

Method Return Values

Objects have an interesting behavior.
What is the **output** of the code below?

```
Team x = new Team("Justice League");  
  
example(x);  
  
System.out.println(x.name);  
  
void example(Team t) {  
    t = new Team("Avengers");  
}
```

Method Return Values

Objects have an interesting behavior.
What is the **output** of the code below?

```
Team x = new Team("Justice League");
```

```
x = example(x);
```

```
System.out.println(x.name);
```

```
Team example(Team t) {  
    t = new Team("Avengers");  
    return t;  
}
```


Method Return Values

Methods may have several input parameters, but they can only return one value.

```
int a = distance(5, 10, 6);
```

```
int distance(int x, int y, int z) {  
    return Math.sqrt((x * x) + (y * y) + (z * z));  
}
```

Method Return Values

However, you can **overload** a method to have a different return value.

```
int a = distance(5, 10, 6);  
double b = distance(3.2, 5.2, 8.7);
```

```
int distance(int x, int y, int z) {  
    return Math.sqrt((x * x) + (y * y) + (z * z));  
}
```

```
double distance(double x, double y, double z) {  
    return Math.sqrt((x * x) + (y * y) + (z * z));  
}
```

```
// No! Will not compile! Must be different parameters!  
double distance(int x, int y, int z) {  
    return (double)Math.sqrt((x * x) + (y * y) + (z * z));  
}
```

Variable Scope

Variable Scope

Variables inside a method are only accessible inside that method.

```
public class Student {  
  
    void startHere() {  
        doSomething();  
        System.out.println(x); // Will not compile!  
    }  
  
    void doSomething() {  
        int x = 5;  
        x = x + x;  
        System.out.println(x);  
    }  
}
```

Variable Scope

Class properties can be accessed by all methods inside the class.

```
public class Student {  
    // Properties  
    private String name;  
  
    // Methods  
    void startHere() {  
        doSomething();  
        System.out.println(name);  
    }  
  
    void doSomething() {  
        name = "(unknown)";  
    }  
}
```

Demo

Don't Forget!

Check the syllabus / schedule for reading assignments and due dates!

static properties and methods



static

When using the **Math** class, you may have noticed we did not instantiate it with **new**. We were able to access a property such as **Math.PI** as well as **Math.sqrt**. That is because these were declared as **static**.

When we instantiate a class, each object has its own copies of the properties and the methods operate on those copies of the properties.

```
Student s1 = new Student("Billy");  
Student s2 = new Student("Sally");
```

```
s1.addScore(80.0);  
s2.addScore(90.0);
```


static

A **static** property is created only once per program execution and is **shared** by all instances of your class.

```
public class Student {  
    public static int nextId = 100;  
  
    private int myId;  
    private String name;  
  
    public Student(string n) {  
        myId = nextId++;  
        name = n;  
    }  
}
```

static

What is the output of the code below (assume the first id is 100):

```
Student s1 = new Student("Billy");  
Student s2 = new Student("Sally");
```

```
System.out.println(s1.getId());  
System.out.println(s2.getId());
```

```
System.out.println(Student.nextId);
```

static

We can also create static methods. Static methods can access static properties. Because they are not part of an instantiated class, they do not have their own copies of properties.

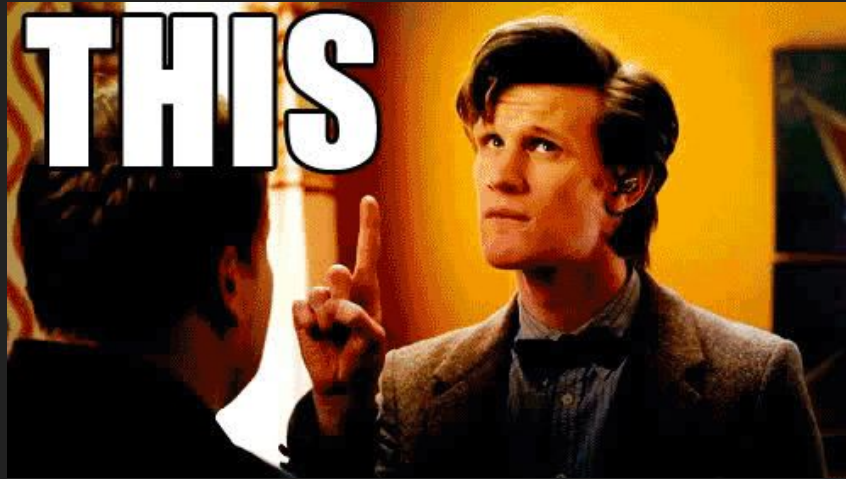
```
public class Student {  
    public static int nextId = 100;  
  
    private int myId;  
    private String name;  
  
    public static void doSomething() {  
        nextId = nextId + 5;           // OK!  
        name = "Voldemort;           // Will not compile!  
    }  
}
```

static

Static methods can be used without using new.

```
Student.doSomething();
```

```
Student s1 = new Student();  
s1.setName("Ron");
```



this

To avoid confusion (and errors) you can use the **this** keyword to explicitly state that you are referring to the current copy of an object.

```
// OK!
public void setName(String name) {
    this.name = name;
}
```

```
// Has no effect!
public void setName(String name) {
    name = name;
}
```

this

You can not use **this** in a static method! This is due to the static method not being part of an instance of a class.

```
public class Student {  
    private String name;  
  
    public static void doSomething() {  
        this.name = "(unknown)"; // Will not compile!  
    }  
}
```

Let's Code

Don't Forget!

Check the syllabus / schedule for reading assignments and due dates!