

# Object orientated programming

The beginners guide

```
mirror_mod = modifier_ob.  
# Add mirror object to mirror  
mirror_mod.mirror_object =
```

```
operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
# Selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
= ("Selected" + str(modifier_ob.name))  
mirror_ob.select = 0  
= bpy.context.selected_objects  
data.objects[one.name].select  
print("please select exactly
```

--- OPERATOR CLASSES ---

```
bpy.types.Operator):  
    "X mirror to the selected  
    object.mirror_mirror_x"  
    "mirror X"
```

```
context):  
    context.active_object is not
```



# Preknowledge required

- I will try to explain things as clear as I can. But some preknowledge is required. For example: creating loops and understanding what variables are. Maybe I'll make a course about IP (Imperative programming) later. But since this one was requested. I'll start with this.



# Lesson 1

- What are objects and methods?
- Setup VSCode
- Instances
- Arrays
- Input/output
- toString()
- Concentration

# Objects and methods

```
... object to mirror...  
... _mod.mirror_object...  
... operation == "MIRROR_X"...  
... mirror_mod.use_x = False  
... mirror_mod.use_y = True  
... mirror_mod.use_z = False  
... operation == "MIRROR_Y"...  
... mirror_mod.use_x = False  
... mirror_mod.use_y = True  
... mirror_mod.use_z = False  
... operation == "MIRROR_Z"...  
... mirror_mod.use_x = False  
... mirror_mod.use_y = False  
... mirror_mod.use_z = True
```

```
... selection at the end - add...  
... mirror_ob.select= 1  
... mirror_ob.select=1  
... context.scene.objects.active  
... ("Selected" + str(modifier...  
... mirror_ob.select = 0  
... = bpy.context.selected_object  
... data.objects[one.name].select  
... print("please select exactly
```

--- OPERATOR CLASSES ---

```
... types.Operator):  
... X mirror to the selected  
... object.mirror_mirror_x"  
... mirror X"
```

```
... t):  
... test is not
```



# What is an object?

- Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class. ~Java docs
- That's not going to work for us
- Our definition: An object in java is a class that has certain state(s) (fields) and behaviors (methods).
- Later we will see why the documentation is more accurate. But for now we use the informal definition.



# What is a method?

- Let's skip over the rest of the definitions and go straight into practicality.
- A method will be our functions we are going to define functions later. But if you know python: a function in python is called a method in java.
- A state/fields will be our variables.



# Setup VSCode

```
... object to mirror  
..._mod.mirror_object =  
... operation == "MIRROR_X":  
..._mod.use_x = False  
..._mod.use_y = True  
..._mod.use_z = False  
... operation == "MIRROR_Y":  
..._mod.use_x = False  
..._mod.use_y = False  
..._mod.use_z = True  
... operation == "MIRROR_Z":  
..._mod.use_x = False  
..._mod.use_y = False  
..._mod.use_z = False
```

```
... selection at the end - add  
..._ob.select= 1  
..._ob.select=1  
... context.scene.objects.active  
... ("Selected" + str(modifier_name))  
..._ob.select = 0  
... = bpy.context.selected_objects  
... data.objects[one.name].select  
... print("please select exactly one object")
```

-- OPERATOR CLASSES --

```
... types.Operator):  
... X mirror to the selected  
..._object.mirror_mirror_x"  
... mirror X"
```

```
... t):  
... test is not
```



# Setup VSCode

- Install VSCode from: <https://code.visualstudio.com/download>
- Since you all study computer science, I expect you are able to install a simple program without any hassle. Tip on windows: Select “Show in file explorer” option during the installation. It can save you so much time.





# Setup Java

- Install JDK: Go to link in the course:  
<https://www.oracle.com/java/technologies/downloads/>
- Restart PC (because we can)
- Windows (and others) check if it's added to your PATH!



# Setup Java in VSCode

- Install some java extensions in VSCode
- I don't know from the top of my head the most important ones but I have the following installed that might come in handy:
  - **Extension Pack for Java**
  - **Java**
  - **Debugger for Java**
  - **Gradle for Java**
  - **Test Runner for Java**
- Links to each of them can be found on the website



# Setup Java

- Create a folder called Exercise 1 or download it from the zipped file
- Create a filename, for example: exercise1.java
- Create a class with the same name as the file
- Create a main function
- Print hello world!
- Pause the video if you want to try it yourself



# Setup Java in VSCode

- The code you should've made is:

```
public class exercise1 {  
    public static void main(String[] args) {  
        System.out.println("Hello world");  
    }  
}
```



# Setup Java in VSCode

- The code you should've made is:

```
public class exercise1 {  
    public static void main(String[] args) {  
        System.out.println("Hello world");  
    }  
}
```

- That should give you an output with: *Hello world!* In the console



# Setup Java in VSCode

- The code you should've made is:

```
public class exercise1 {  
    public static void main(String[] args) {  
        System.out.println("Hello world");  
    }  
}
```

- That should give you an output with: *Hello world!* In the console
- What in this code is the object? What is the method and what are the states? **Quiz question 3-6**



# Setup Java in VSCode

- I'm not going to cover troubleshooting here. But in short if it doesn't work:
  - Is Java set in your PATH? And did you restart the PC after adding it to your PATH?
  - Did you install all extensions on VSCode? Do you maybe need some other as well?
  - Did you instal JDK and not JRE?

# Constructor

```
... object to mirror...  
... _mod.mirror_object...  
... operation == "MIRROR_X":  
... mirror_mod.use_x = True  
... mirror_mod.use_y = False  
... mirror_mod.use_z = False  
... operation == "MIRROR_Y":  
... mirror_mod.use_x = False  
... mirror_mod.use_y = True  
... mirror_mod.use_z = False  
... operation == "MIRROR_Z":  
... mirror_mod.use_x = False  
... mirror_mod.use_y = False  
... mirror_mod.use_z = True
```

```
... selection at the end - add...  
... mirror_ob.select= 1  
... mirror_ob.select=1  
... context.scene.objects.active  
... ("Selected" + str(modifier...  
... mirror_ob.select = 0  
... = bpy.context.selected_object  
... data.objects[one.name].select  
... print("please select exactly
```

--- OPERATOR CLASSES ---

```
... types.Operator):  
... X mirror to the selected  
... object.mirror_mirror_x"  
... mirror X"
```

```
... t):  
... test is not
```



# What is the constructor?

- It's to pass variables into a class

# What is the constructor?

- How to create one?
- Let's say we have the following code:
- Then this will do nothing usefull yet.
- But it might be fun to see what the result is. So try it for yourself

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the post-it note  
    private String color;  
  
    public static void main(String[] args) {  
        example postIt = new example();  
        System.out.println(postIt);  
    }  
}
```



# What is the constructor?

- But first... some technical terms
- This code creates a class called “example” which we interpreted as “post\_it”
- Then we created two fields
  - ‘text’ – private String
  - ‘color’ – private String

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the post-it note  
    private String color;  
  
    public static void main(String[] args) {  
        example postIt = new example();  
        System.out.println(postIt);  
    }  
}
```

# What is the constructor?

- We saw that we had to define types before in C++, but what is 'example'?
- It means that the class example can be interpered as a type. (Practically (official name is a reference type))
- A 'new' variable made of that type is called an instance

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the post-it note  
    private String color;  
  
    public static void main(String[] args) {  
        example postIt = new example();  
        System.out.println(postIt);  
    }  
}
```

# What is the constructor?

- Now what is a the constructor?
- It starts with public
- Followed directly with the name of the class (so no type)
- Meaning we get a result like:

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the post-it note  
    private String color;  
  
    public static void main(String[] args) {  
        example postIt = new example();  
        System.out.println(postIt);  
    }  
}
```

# What is the constructor?

- Now what is a the constructor?
- It starts with public
- Followed directly with the name of the class (so no type)
- Meaning we get a result like:

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the  
    post-it note  
    private String color;  
  
    public example(){  
        // more code  
    }  
    public static void main(String[] args) {  
        example postIt = new example();  
        System.out.println(postIt);  
    }  
}
```

# What is the constructor?

- I know right. Really hard to make.
- So how to make it that we can create an actual post-it inside our main function?
- We add the magic of “parameters”

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the post-it note  
    private String color;  
  
    public example(){  
        // more code  
    }  
    public static void main(String[] args) {  
        example postIt = new example();  
        System.out.println(postIt);  
    }  
}
```



# What is the constructor?

- I know right. Really hard to make.
- So how to make it that we can create an actual post-it inside our main function?
- We add the magic of “parameters”

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the post-it note  
    private String color;  
  
    public example(String textInput, String colorInput){  
        // more code  
    }  
  
    public static void main(String[] args) {  
        example postIt = new example();  
        System.out.println(postIt);  
    }  
}
```

# What is the constructor?

- I know right. Really hard to make.
- So how to make it that we can create an actual post-it inside our main function?
- We add the magic of “parameters”
- And then add assignments

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the post-it note  
    private String color;  
  
    public example(String textInput, String colorInput){  
        // more code  
    }  
  
    public static void main(String[] args) {  
        example postIt = new example();  
        System.out.println(postIt);  
    }  
}
```

# What is the constructor?

- I know right. Really hard to make.
- So how to make it that we can create an actual post-it inside our main function?
- We add the magic of “parameters”
- And then add assignments

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the post-it note  
    private String color;  
  
    public example(String textInput, String colorInput){  
        text = textInput;  
        color = colorInput;  
    }  
    public static void main(String[] args) {  
        example postIt = new example("OOP with Thijmen almost lo  
        "Green");  
        System.out.println(postIt);  
    }  
}
```

# ToString

```
..._mod.mirror_object:  
operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
...selection at the end -add  
mirror_ob.select= 1  
mirror_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob))  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
print("please select exactly one")
```

--- OPERATOR CLASSES ---

```
...types.Operator):  
    "X mirror to the selected  
    object.mirror_mirror_x"  
    "mirror X"
```

```
...t):  
    test is not
```



# Are you still mad?

- We wrote this amazing looking code. But when we run it we get something like:
- `example@372f7a8d`
- So what happens?
- We try to print an object. But the compiler/Java does not know how to format that object to a String. Meaning we have to do that for Java.
- We can do this by overriding the main toString method. And how to do that?
- Simply create a function/method called: toString



# toString

- We don't need to specify `@Override` since we haven't covered it yet.

```
public String toString(){  
    return "Post-it text:\n" + text + "\nColor: " + color + "";  
}
```

- Adding this to our class does make it more readable when ran.

# toString

- So how does our complete code looks like?
- All that for a simple post-it system that isn't even that modifiable (no getters/setters).

```
public class example {  
    // example represents a post-it Note  
    private String text; // the text on the  
    post-it note  
    private String color;  
  
    public example(String textInput, String  
colorInput){  
        text = textInput;  
        color = colorInput;  
    }  
  
    public String toString(){  
        return "Post-it text:\n" + text +  
"\nColor: " + color + "  
";  
    }  
  
    public static void main(String[] args) {  
        example postIt = new example("OOP with  
Thijmen almost looks fun!", "Green");  
        System.out.println(postIt);  
    }  
}
```

# Encapsulation

```
..._mod.mirror_object =  
operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
...selection at the end -add  
mirror_ob.select= 1  
mirror_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob))  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
print("please select exactly one")
```

--- OPERATOR CLASSES ---

```
...types.Operator):  
    "X mirror to the selected  
    object.mirror_mirror_x"  
    "mirror X"
```

```
...t):  
    test is not
```



# What is the encapsulation?

- They talked a about it in the lectures, thus so do I.
- It means untill now: \*Hold on to your chairs\*
- Use getters and setters
- Use private if possible



# Getters and setters

- This actually is quite important in OOP
- Getters give you a private variable, without directly accessing that specific variable from the outside. Also don't make getters for variables that don't need to be read. This makes sure that outsiders can't get the information of private variables.
- Setters set the private variable, without directly accessing that specific variable from the outside. Also don't make setters for variables that don't need to be changed.



# Getters and setters

- How do they look like?
- First let's define what we want to model:
  - Be able to add/change the text on the post-it
  - Be able to read the text on the post-it
  - You are not able to change to color of the post-it
  - You are able to see what the color of the post-it is
- These restrictions mean we can alter the post-it code such that it looks like:



# Getters and setters

```
public class example2 {  
    // example represents a post-it Note  
    private String text; // the text on the post-it note  
    private String color;  
  
    public example2(String colorInput){  
        color = colorInput;  
    }  
  
    public String toString(){  
        return "Post-it text:\n" + text + "\nColor: " + color + "";  
    }  
  
    public static void main(String[] args) {  
        example2 postIt = new example2("Green");  
        System.out.println(postIt);  
    }  
}
```



# Getters and setters

- A getter is nothing more than just returning the current variable.
- Meaning a getter for 'text' and 'color' will look like:

```
public String getText(){  
    return text;  
}
```

```
public String getColor(){  
    return color;  
}
```





# Getters and setters

- A setter is nothing more than updating the variable:

```
public void setText(String textInput){  
    text = textInput;  
}
```



# Getters and setters

- A tip I might give. If things are like a counter. Don't use something like:
- `object.setAmount(object.getAmount() + 1);`
- But rather:
- `object.increaseAmount(1);`
- Meaning you are allowed to change getters and setters if you see fit.
- The first one is not wrong, but can get quite complex fast.

# Getters and setters

- Now we can change the main function to make sure it uses getters and setters.
- A toString function is better but for illustration purposes we use it like this.

```
public static void main(String[] args) {  
    example2 postIt = new example2("Green");  
    postIt.setText("OOP From Thijmen is amazing!");  
    System.out.println("Text: " + postIt.getText() + " | Color: " + postIt.getColor());  
}
```

# Static and dynamic

```
..._mod.mirror_object =  
operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
...selection at the end -add  
mirror_ob.select= 1  
mirror_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob))  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
print("please select exactly one")
```

--- OPERATOR CLASSES ---

```
...types.Operator):  
    "X mirror to the selected  
    object.mirror_mirror_x"  
    "mirror X"
```

```
...t):  
    test is not
```



# Fields

- Static fields are independent of the object.
- There can only be one per class



# Methods

- Static methods are independent of any object
  - Cannot access instance fields or methods

# Some remarks until now

```
..._mod.mirror_object =  
operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
...selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.name))  
mirror_ob.select = 0  
= bpy.context.selected_objects  
data.objects[one.name].select  
print("please select exactly one object")
```

--- OPERATOR CLASSES ---

```
...types.Operator):  
    "X mirror to the selected  
    object.mirror_mirror_x"  
    "mirror X"
```

```
...t):  
    test is not
```





# Before we continue

- Before we continue, I do have some terminology remarks. And a small overview of what we covered until now.
- A String in java is not a primitive, and uses the '+' to concatenate two strings.
- A class as type is called a reference type
- A boolean, integer, float, double, etc. is called a primitive type.
- Methods may change the values of the fields
- Constructors invoked implicitly via **new** to create an instance of a class
- Fields typically initialized by the constructor
- Static fields shared by all objects of a class
- Static methods cannot access instance fields



# Arrays

```
..._mod.mirror_object =  
operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
...selection at the end -add  
mirror_ob.select= 1  
mirror_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob))  
mirror_ob.select = 0  
= bpy.context.selected_objects  
data.objects[one.name].select  
print("please select exactly one")
```

--- OPERATOR CLASSES ---

```
...types.Operator):  
    "X mirror to the selected  
    object.mirror_mirror_x"  
    "mirror X"
```

```
...t):  
    test is not
```



# Array

- An array is an ordered list of variables of the same type. It's **ordered** and thus **not sorted**
- Initializing an array can be done in multiple ways:

```
double[] temperature = new double[3];  
double[] temperature = new double[3] { 3.3, 15.8, 9.7 };  
double[] temperature = { 3.3, 15.8, 9.7 };
```

- My recommendation is to use the first way if it has to be sort of dynamic (you change values) and the last one if you store constants (of course use the conventional CAPS))

# Array

- You can get the length of an array via the length attribute
- And access it via for loops in 2 ways (second one is recommendation for starters)

```
int[] array = { 1,2,3 };
int total1 = 0;
int total2 = 0;
// add each element's value to total. Read-only access, readable
for ( int number : array ) {
    total1 += number;
}
// add each element's value to total. less readable
for (int j = 0; j < array.length; j++) {
    total2 += array[j];
}
System.out.println( "Total of array elements: " + String.valueOf(total1));
System.out.println( "Total of array elements: " + String.valueOf(total2));
```

# Java input and output

```
... object to mirror...  
... _mod.mirror_object...  
... operation == "MIRROR_X":  
... mirror_mod.use_x = True  
... mirror_mod.use_y = False  
... mirror_mod.use_z = False  
... operation == "MIRROR_Y":  
... mirror_mod.use_x = False  
... mirror_mod.use_y = True  
... mirror_mod.use_z = False  
... operation == "MIRROR_Z":  
... mirror_mod.use_x = False  
... mirror_mod.use_y = False  
... mirror_mod.use_z = True
```

```
... selection at the end - add...  
... mirror_ob.select= 1  
... mirror_ob.select=1  
... context.scene.objects.active  
... ("Selected" + str(modifier_ob...  
... mirror_ob.select = 0  
... = bpy.context.selected_object...  
... data.objects[one.name].select...  
... print("please select exactly...  
... -- OPERATOR CLASSES --
```

```
... types.Operator):  
... X mirror to the selected...  
... object.mirror_mirror_x"  
... mirror X"
```

```
... t):  
... test is not
```



# I/O in Java

- Use a single class for all Input and Output in Java
- Use the Java Scanner for the input.



# End of lecture 1

```
..._mod.mirror_object =  
operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
...selection at the end -add  
mirror_ob.select= 1  
mirror_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_name))  
mirror_ob.select = 0  
= bpy.context.selected_objects  
data.objects[one.name].select  
print("please select exactly one object")
```

--- OPERATOR CLASSES ---

```
...types.Operator):  
    "X mirror to the selected  
    object.mirror_mirror_x"  
    "mirror X"
```

```
...t):  
    "test is not"
```