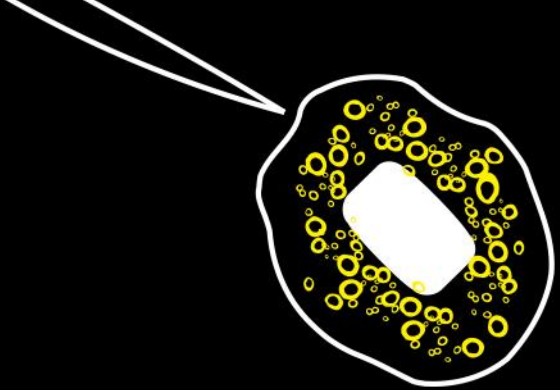
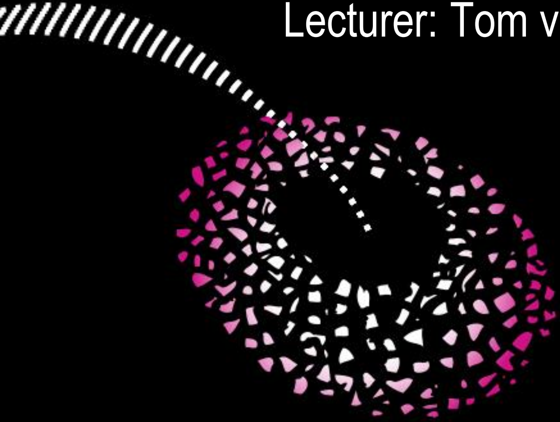
UNIVERSITY OF TWENTE.

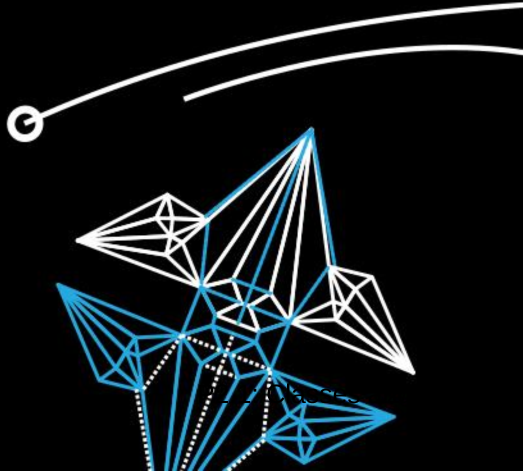


Constructors

Topic of Software Systems (TCS module 2)

Lecturer: Tom van Dijk





CONSTRUCTORS

INITIALIZATION

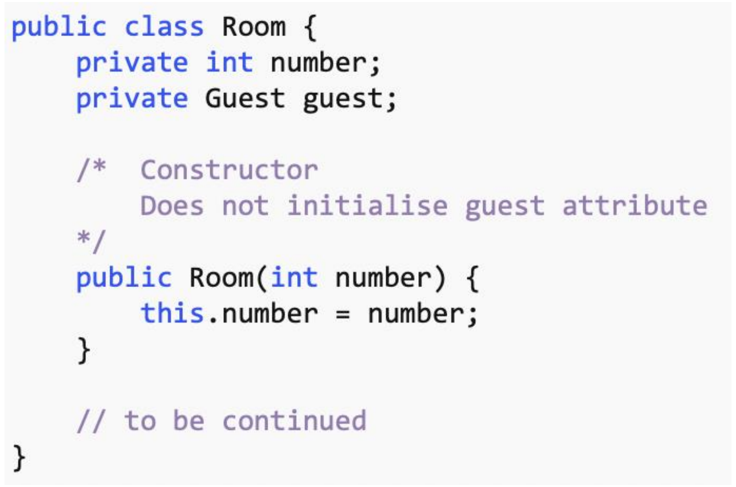
Variables always have some initial value.

- Local variables must be initialized before use
- Instances and class variables have default values

Type	Default value
boolean	false
byte, short, int, long	0
float, double	0.0
char	‘\u0000’
reference types	null

CONSTRUCTOR

- A method with the same name as its class and no return type



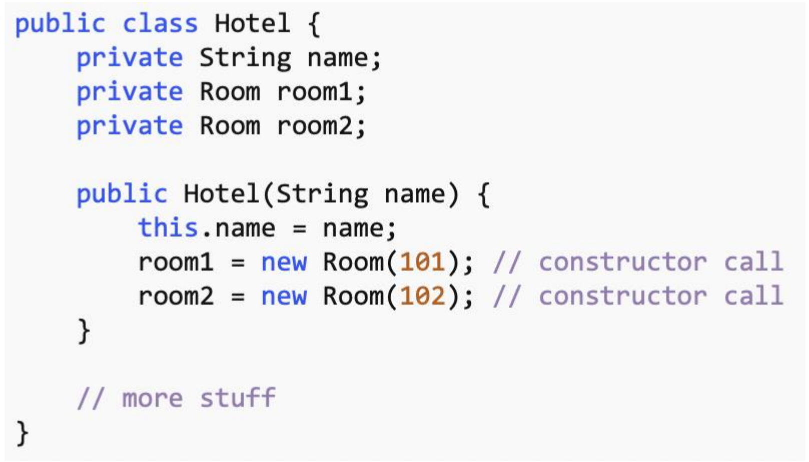
```java
public class Room {
    private int number;
    private Guest guest;

    /*  Constructor
        Does not initialise guest attribute
    */
    public Room(int number) {
        this.number = number;
    }

    // to be continued
}
```

CONSTRUCTOR

- Called by: `new Constructor(arguments);`
- This returns a newly allocated object initialized by the constructor



```java
public class Hotel {
    private String name;
    private Room room1;
    private Room room2;

    public Hotel(String name) {
        this.name = name;
        room1 = new Room(101); // constructor call
        room2 = new Room(102); // constructor call
    }

    // more stuff
}
```

DEFAULT CONSTRUCTOR

- If you don’t explicitly initialize a field, it will have its default value

- You can define many different constructors
  - Same method name
  - Different parameters

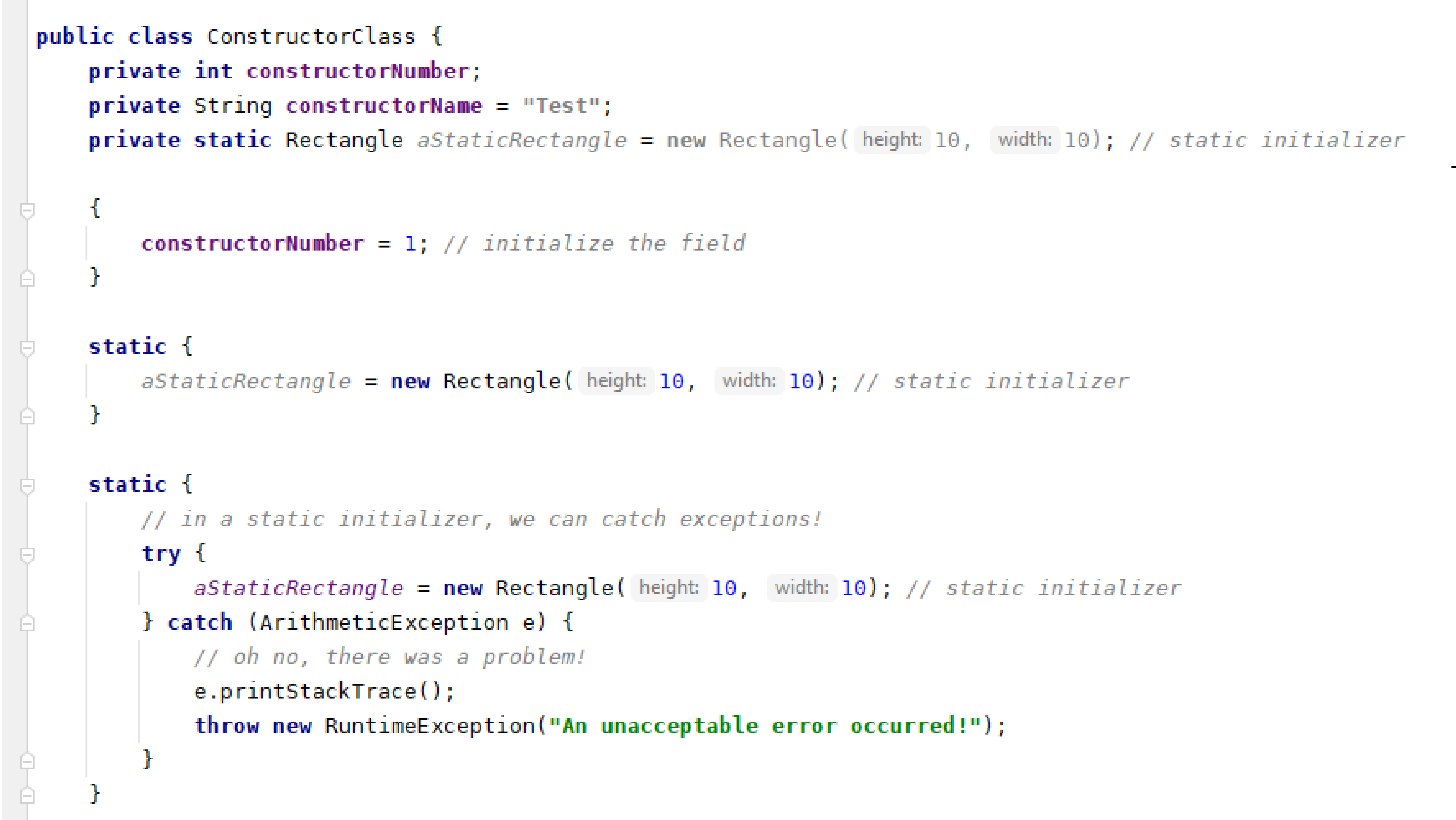
- Only if you don’t define any constructor, the default constructor is an empty public constructor

INITIALIZERS

Another feature of Java: initializers

A `{ code block }` inside a class that initializes values

UNIVERSITY OF TWENTE.



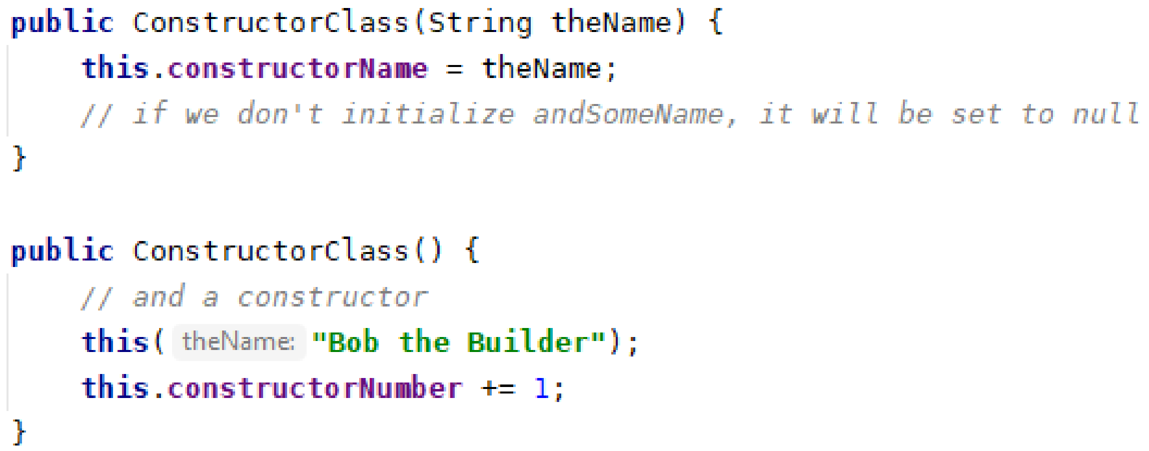
```java
public class ConstructorClass {
    private int constructorNumber;
    private String constructorName = "Test";
    private static Rectangle aStaticRectangle = new Rectangle( height: 10,  width: 10); // static initializer

    {
        constructorNumber = 1; // initialize the field
    }

    static {
        aStaticRectangle = new Rectangle( height: 10,  width: 10); // static initializer
    }

    static {
        // in a static initializer, we can catch exceptions!
        try {
            aStaticRectangle = new Rectangle( height: 10,  width: 10); // static initializer
        } catch (ArithmeticException e) {
            // oh no, there was a problem!
            e.printStackTrace();
            throw new RuntimeException("An unacceptable error occurred!");
        }
    }
```

CONSTRUCTOR EXAMPLE



```
public ConstructorClass(String theName) {
    this.constructorName = theName;
    // if we don't initialize andSomeName, it will be set to null
}

public ConstructorClass() {
    // and a constructor
    this( theName: "Bob the Builder");
    this.constructorNumber += 1;
}
```

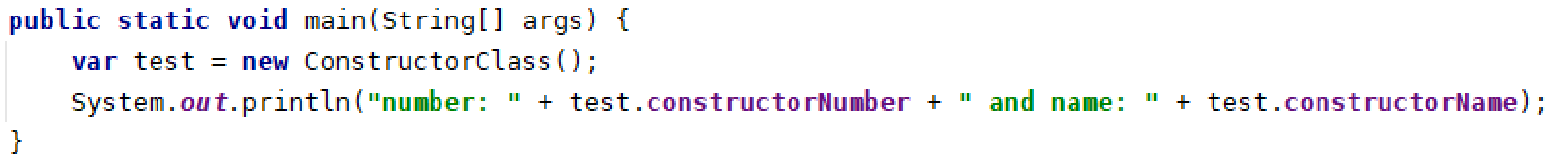
ORDER FOR INITIALIZATION

Order of initialization:

- static variables and static initializers (in order)
- instance variables and instance initializers (in order)
- constructor

CONSTRUCTOR EXAMPLE

- What happens if this is executed:



```java
public static void main(String[] args) {
    var test = new ConstructorClass();
    System.out.println("number: " + test.constructorNumber + " and name: " + test.constructorName);
}
```

- First the static initializer sets constructorNumber to 1
- Then the constructor increases constructorNumber by 1

DESTRUCTORS

- Some languages also have destructors to cleanup after an object is deleted: free memory, release system resources
- This is not needed in Java

CONCLUSION

- Initializers and constructors initialize object fields
- Pay attention to the execution order
- Java only creates a default constructor if no constructor is defined