# Summary Week 2

## Recap Week 1

- **Differences between variables and attributes**
  - Attributes are variables that belong to an object
  - (Local) variables are only valid within a method

- **Define method**

```
void dig(){
        //this method defines how to dig a hole
        ...
}
```

- **Call method**

```
dig(); //calling the method digs the hole
```

- **Method with return value**

```
String getName(){
        return "Duke";
}
```

## Array

Container for items of the same data type

```
<<data type>>[] <<arrayIdentifier>>;
```

- Initialization with new initializes all elements of an array with the data type's default value
- Declaration and initialization can be done at the same time:

```
String[] cases = new String[5];
```

```
<<data type>>[] <<arrayIdentifier>>
                = new <<same data type>>[<<size>>];
```


- In Java you start counting with index 0

- In Java, Arrays **can't** change size!

  → dynamically adding more elements is not possible

- <arrayname>.length returns the indexed size of an array

Loops

- Executes a statement multiple times
- Has a
  - **loop condition** specifying the iteration
  - **loop body** containing the statement(s) to be executed

- **Types:**
  - Count-controlled loop: for (from x to y, e.g. count y times)
  - Condition-controlled loop:
    - while (as long as x is true, do y)
      - ➔ check condition then execute statement
    - do while (do y as long as x is true)
      - ➔ execute statement then check condition
      - ➔ statements in body are always executed at least once
  - Collection-controlled loop

## For - loop

```
1  for (int i = 0; i < 3; i++) {
2          // this is a count-controlled loop
3          // it will be processed three times
4  }
```

loop condition

loop body

## While – loop

- Condition will be checked before every iteration of the loop
- Loop body will be executed as long as loop condition is true
- Any desired condition is possible,
  - i < j, a != b, a, !a, ...
  - userHasNotCancelled(), dukeHasNotCaughtBadGuy(), ...

**Infinite loops possible!**

## Method Signature, Parameters, Arguments

- Arguments passed to a method have to match expected parameters

- Defines the public interface of a method.
- Components
  - Identifier
  - Return type
  - Amount and type of expected parameters

**Method signature**

```
String concat(String p1, String p2, String connector)
```

**Return type** | **Identifier** | **Parameters → Arguments**

```
String myString = concat("Hello", "Duke", ":::");
```

**Method call**

## Scope

- Defines the "zone of validity" of a variable
  - The variable can only be accessed (read/write) within that zone
- Defined/delimited by a set of curly braces
  - Class
  - Method
  - Control structure (condition/loop)

## Constructor

- `new` is used to call the **constructor** of the `class`
- The **constructor** defines the initial state of the object

- If no explicit constructor is defined, a default constructor is used
  - The default constructor has no parameters
  - All attributes are initialized with default values

    (e.g. `int`: `0`, `boolean`: `false`)

Default constructors are not existing/visible in the source code They exist implicitly.

Once a custom constructor is defined, the default constructor is no longer available and has to be explicitly added to the code if required.

- The constructor is a method with some special features:
  - Initializes attributes
  - **Same** identifier as its `class`
  - **No** explicit return type – it returns the newly created object
  - It can take **any** number of parameters (including zero)

- Constructor with parameter
  - If the parameter has the same identifier than the attribute, `this` is used to clearly distinguish the local parameter from the attribute
  - `this` always refers to the current object

## Constructore Overload

- Overloading constructors is only possible when
  - the amount of parameters – or –
  - the types of the parameters differ

## This

```
this.<<attributeIdentifier>>
```

- Allows to access the current object's attributes and methods
- Required in case of a naming conflict between parameter or local variable and attribute
- Implicitly set when no collision, often used anyway for better readability

```
this(<<argumentList>>);
```

- Calling an overloaded constructor of the same class
- Can only be called from within a constructor
- Has to be the first statement in the constructor
- Improves maintainability

## Method Overload

- **Define multiple versions of a method with the same identifier:**
  - Different **amount** of **parameters**
  - Different **type** of **parameters**
- **A different return type is not sufficient**
- **Different identifiers** for the parameters **are not sufficient**
- **Do not overload methods that do different things**
- Rather use methods with different identifiers

Methods that share the same identifier, but have different parameters (type or amount) are called **overloaded** methods.

Every method **can** be overloaded.

Constructors are a special form of methods ➔ overloaded.

## Null

- Reserved keyword
- Placeholder for non-existing objects
- Test whether a value is `null` with == (equality operator)
- Test whether a value is not `null` with != (inequality operator)

| Data types | Default values (Attributes) |
|---|---|
| `int` | `0` |
| `double` | `0.0` |
| `char` | `'\u0000'` |
| `boolean` | `false` |
| `String` | `null` |
| `All Objects` | `null` |

- If attributes are not initialized by the developer, Java provides them with a default value

**(Local) variables do not have default values!**

## Exceptions

```
Exception in thread "main"
java.lang.NullPointerException
```

Exception Type

```
at ExceptionExamples.npe(ExceptionExamples.java:7)
```

Class Method Filename Line

```
at ExceptionExamples(ExceptionExamples.java:3)
```

where method npe() was called

- "Exceptional Event"
- Event that disrupts the normal flow of a program when an error occurs
- It's an object

```
try {
  // could potentially raise an exception
} catch (<<ExceptionType>> e) {
  // recover from exception
}
```


- Type of Exception to catch goes into the brackets (e.g. `IOException`, `ArrayIndexOutOfBoundsException`, ...)
- A way to let the program handle the exception (without terminating the program)

## Nested Loops

```
for (int i = 0; i < 2; i++) {
    System.out.print("Ping - ");
    for (int j = 0; j < 3; j++){
        System.out.print("Pong!");
    }
    System.out.print("\n");
}
```

## Two-Dimensional Arrays

```
int[][] image = new int[w][h];
```

```
for (int y = 0; y < h; y++) {
    for (int x = 0; x < w; x++){
        System.out.print(image[y][x]+" ");
    }
    System.out.print("\n");
}
```

## Continue, Break, Return