Introduction to programming

Paul Bodean

Get to know each other

PAUL BODEAN

QA AUTOMATION LEAD

HELP THE LOCAL COMMUNITY
TO GET INTO THE IT FIELD

Summary

Recap

Introduction to testing

Test design techniques

Risk based testing

Testing tools

Agile and waterfall

Theory

How to deal in practice

What you need to know for automation testing

- Data types
- Basic operators
- Loop control
- Decision making

Practice

Git basics

Local setup

Let’s play a game

Play around

Theory

OOP basics

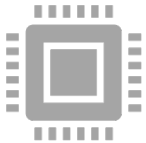
Practice

Work as a team

Let’s do some exercises

Goal

Everyone to understand and accomplish the the config part of the working environment



Understand the benefits of programming in QA industry

Everyone to manage the implementation of at least two programs

Publish code on a repo

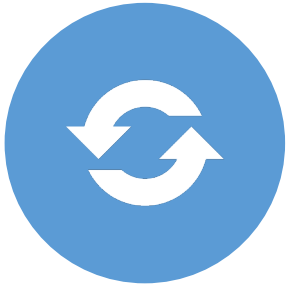
Get familiar with OOP

Recap – Intro to testing

Why we need this?

Bug – first thing to keep in mind

Types of testing



SDLC & STLC

Recap – Intro to testing
Types of testing

Functional Testing	Non-functional Testing
• Unit Testing	• **Performance Testing**
• **Integration Testing**	• Load Testing
• System Testing	• **Stress Testing**
• Sanity Testing	• Volume Testing
• **Smoke Testing**	• **Security Testing**
• Interface Testing	• Compatibility Testing
• **Regression Testing**	• Install Testing
• Beta/Acceptance Testing	• Recovery Testing
	• Reliability Testing
	• **Usability Testing**
	• Compliance Testing
	• **Localization Testing**

Recap – Intro to testing – SDLC & STLC

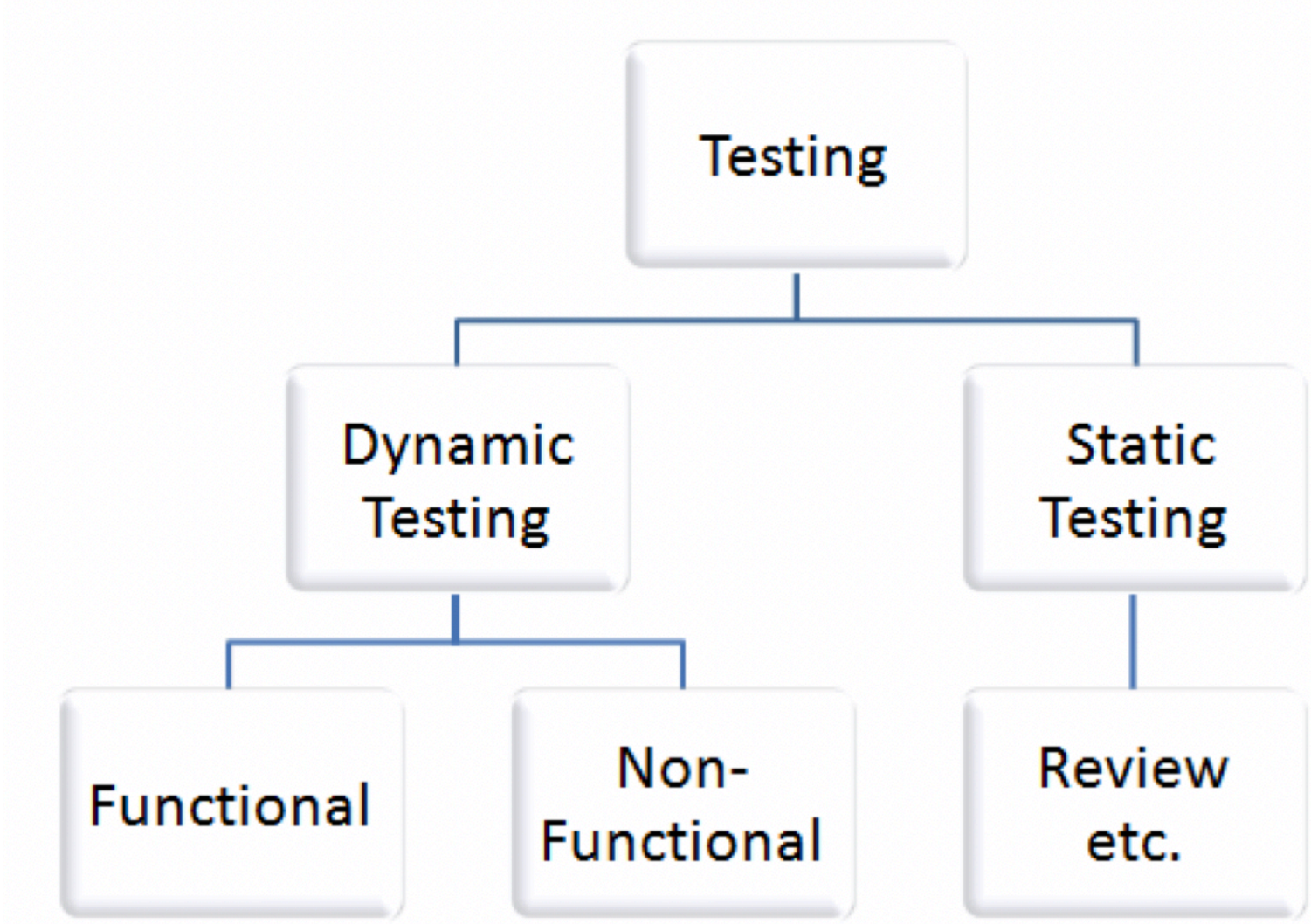
SDLC

- Planning and Requirement Analysis
- Defining Requirements
- Designing the Product Architecture
- Building or Developing the Product
- Testing the Product
- Deployment in the Market and Maintenance

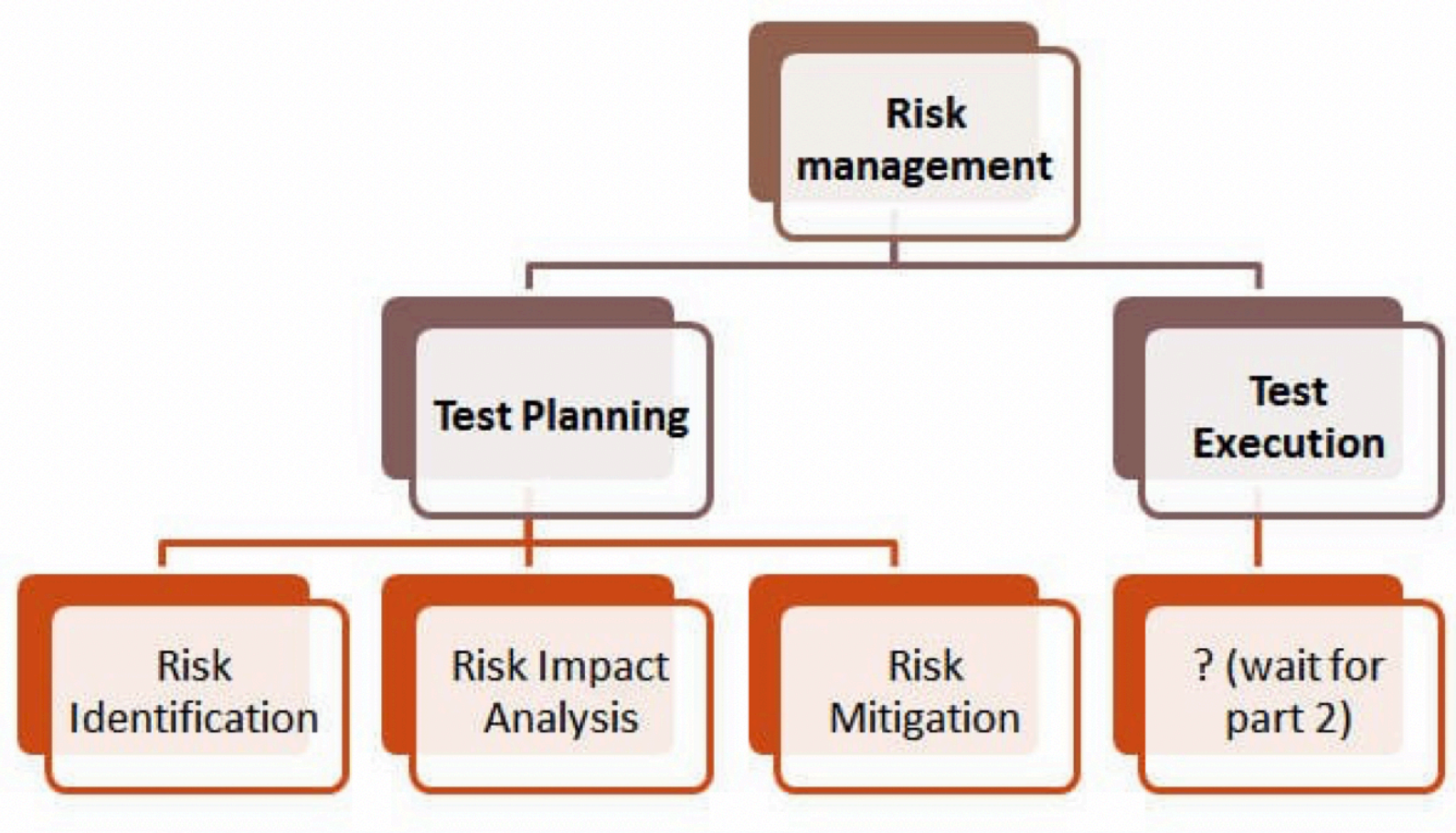
STLC

- Requirements phase
- Planning Phase
- Development Phase
- Test environment setup
- Execution Phase
- Closure Phase

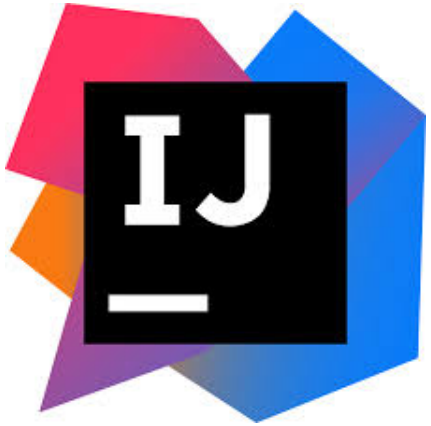
Recap – Test design techniques



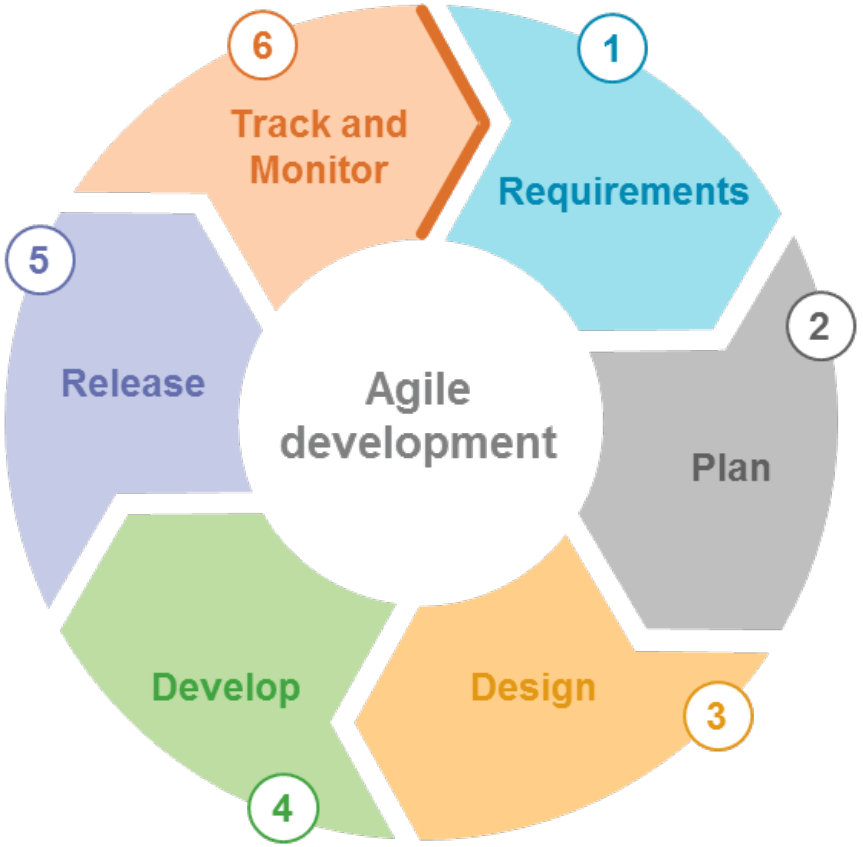
Recap – Risk based testing

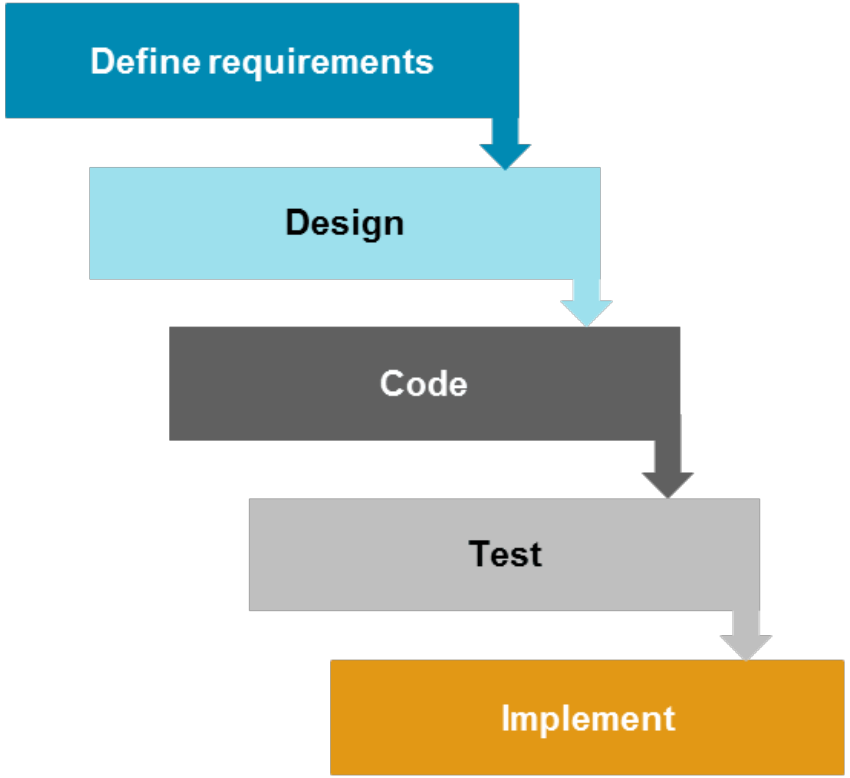


Recap – Testing tools

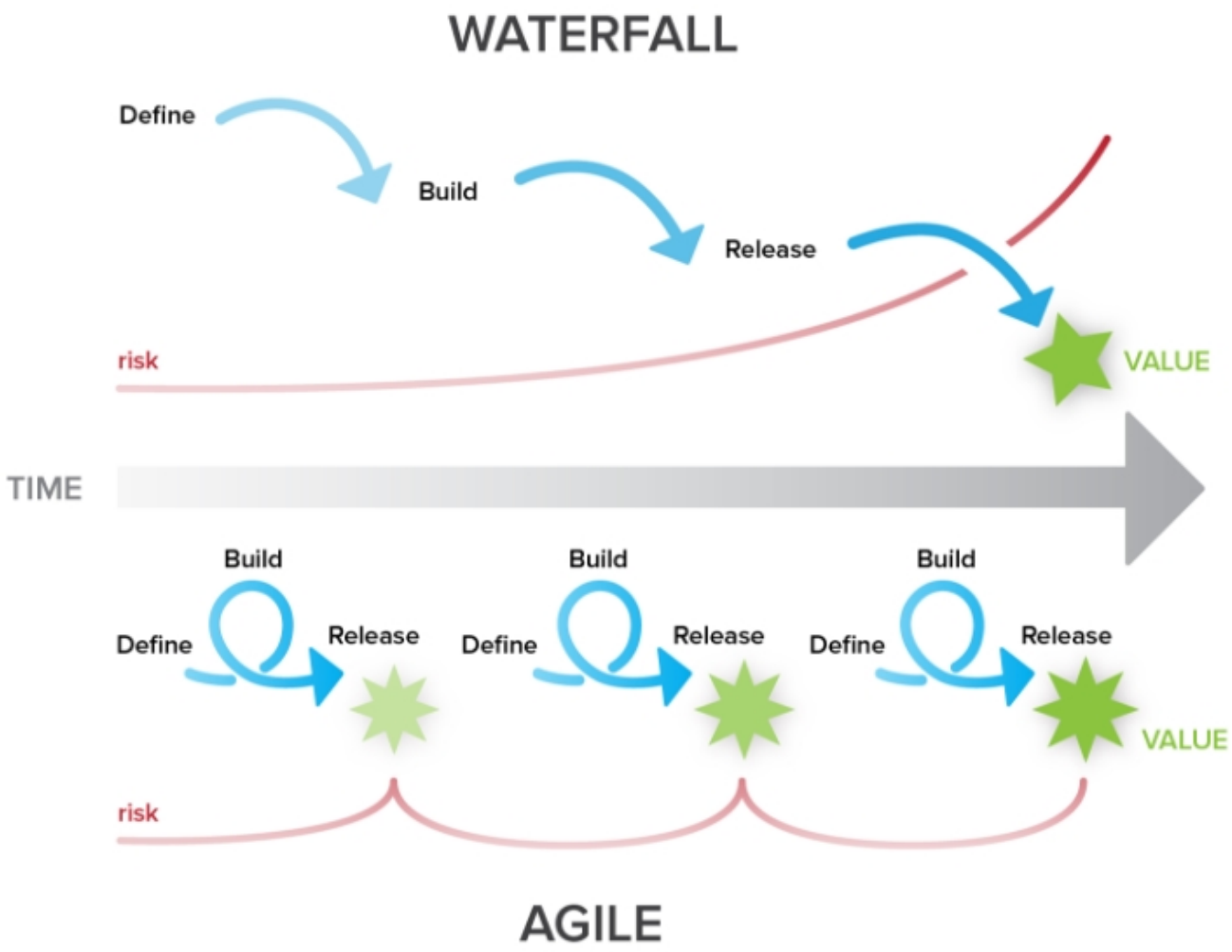


Recap – Agile and Waterfall



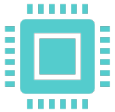


Recap – Agile and Waterfall



Theory - How to deal in practice

Why we need this?



What you can do with programming?

How to start?

Useful documentation

Stackoverflaw

Tutorials point

http://rogerdudler.github.io/git-guide/

Let's start!

Theory - Why Java? – the complex part

Object oriented – everything is an object

Platform independent – not compiled into platform specific machine

Simple – if you know OOP, it's a peace of cake

Secure – virus free

Architectural-neutral – compiled code runs on many processors

Portable – runs on every hardware with JVM installed

Robust – error is returned when complied

Multithreaded

High-performance – code converted into machine on the fly

Distributed – can run on computer networks

Dynamic – design to adapt to an evolving environment

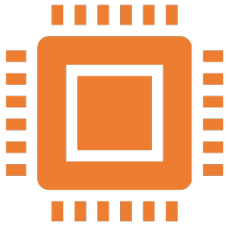
Theory - Why Java? – to keep in mind

Object oriented – everything is an object (if you know OOP everything is a peace of cake)

Robust (Compiled language) – error is returned when complied

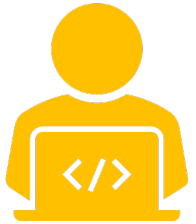
Write automated tests

Theory - Why Java? (Needs)



• JDK (Java Development Kit) – the software for programmers who want to write Java programs

• JRE (Java Runtime Environment)– the software for consumers who want to run Java programs



• IDE (Integrated Development Environment) – a software application which enables users to more easily write and debug Java programs

Install our IDE

Download and install IntelliJ

If JDK not available install it: https://jdk.java.net/13/

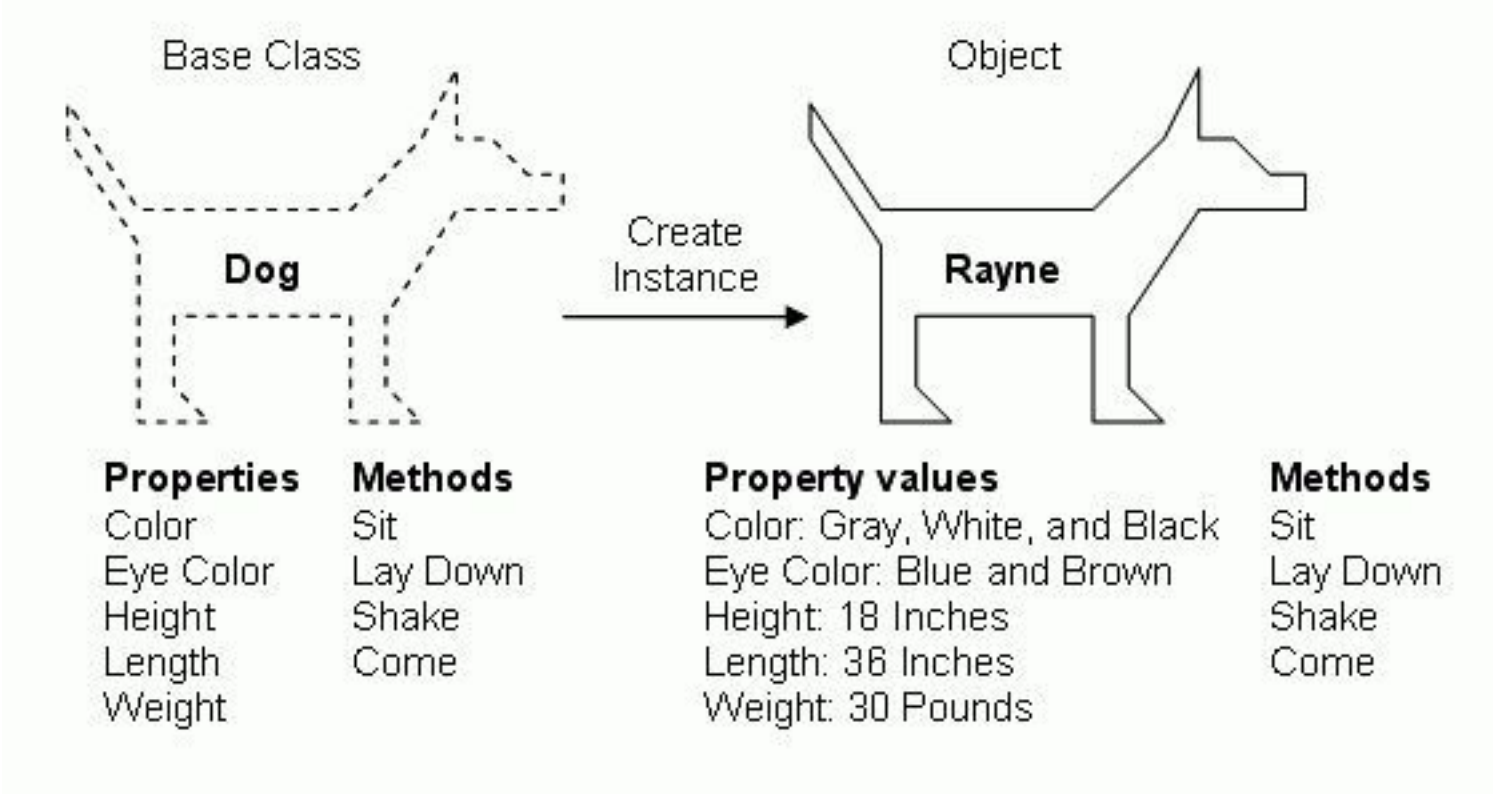
Create an empty Java project

https://www.jetbrains.com/help/idea/installation-guide.html

Theory - Basics

Java program - a collection of objects that communicate via invoking each other's methods

- Class
- Object
- Method
- Instance variables



Theory – Basic syntax

Case Sensitivity – Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.

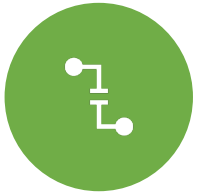
Class Names – For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

Example: *class MyFirstJavaClass*

Method Names – All method names should start with a Lower-Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. *public void myMethodName()*

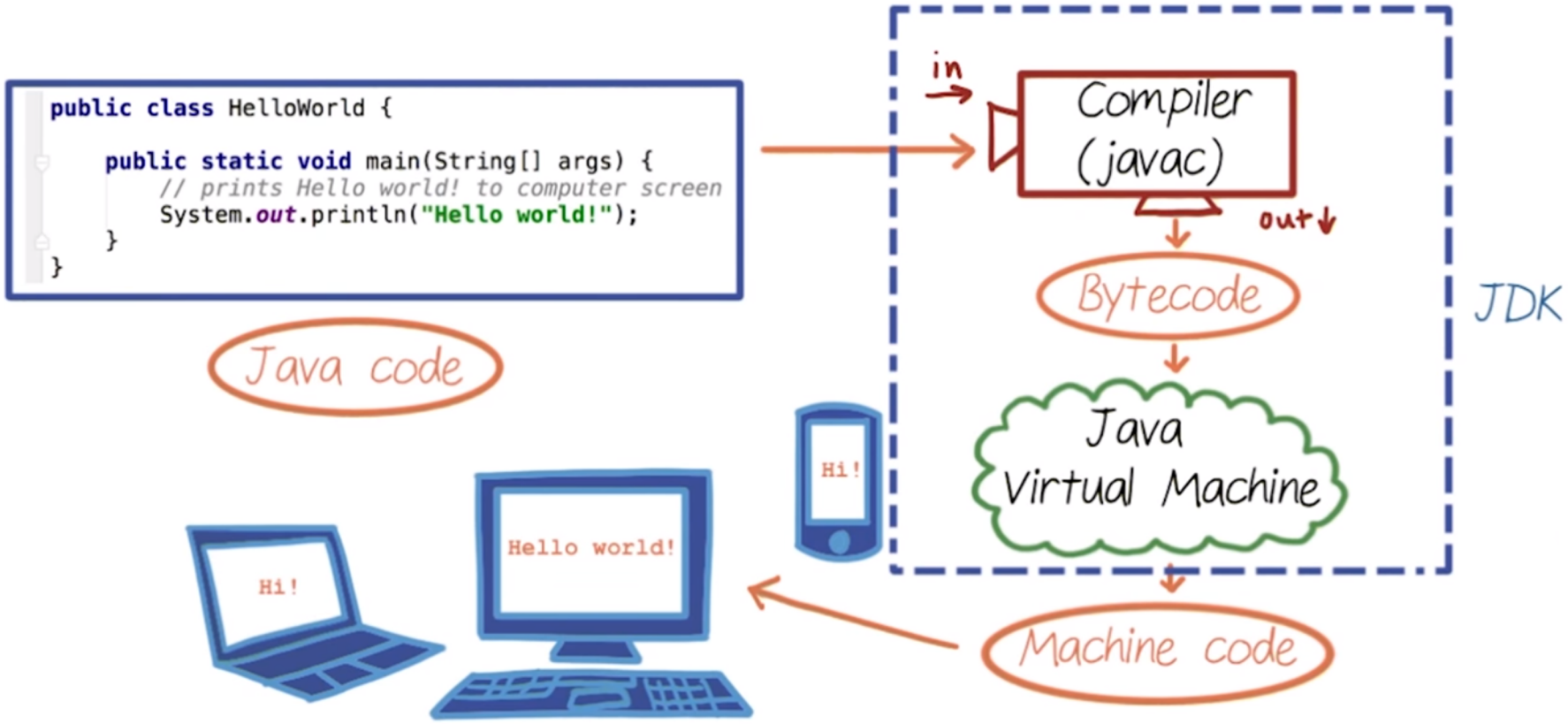
Program File Name - should exactly match the class name. When saving the file, you should save it using the class (if the file name and the class name do not match, your program will not compile).

Example: if 'MyFirstJavaProgram' is the class name. Then the file should be *'MyFirstJavaProgram.java'*



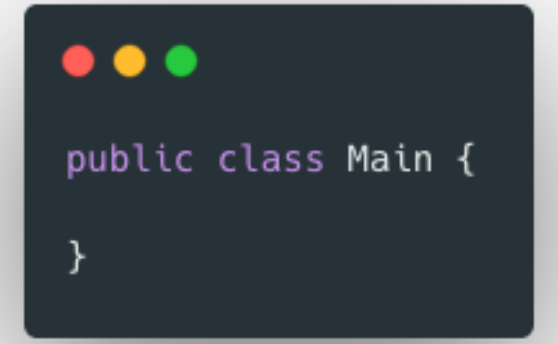
public static void main(String args[]) – Java program processing starts from the main() method which is a mandatory part of every Java program.

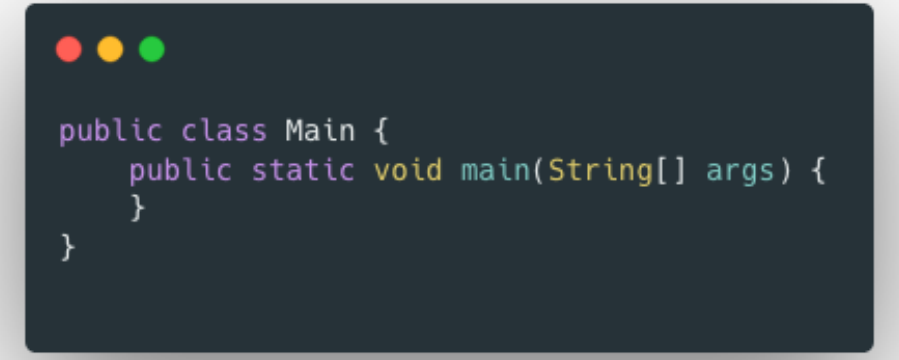
Theory – Basic syntax



Theory – First program

- Under the src module, right click, select Java class
- Set name to “Main”
- Secondly add a method inside the class. Hint -> type psvm
- Now we have the class define, a method and a code block between square brackets





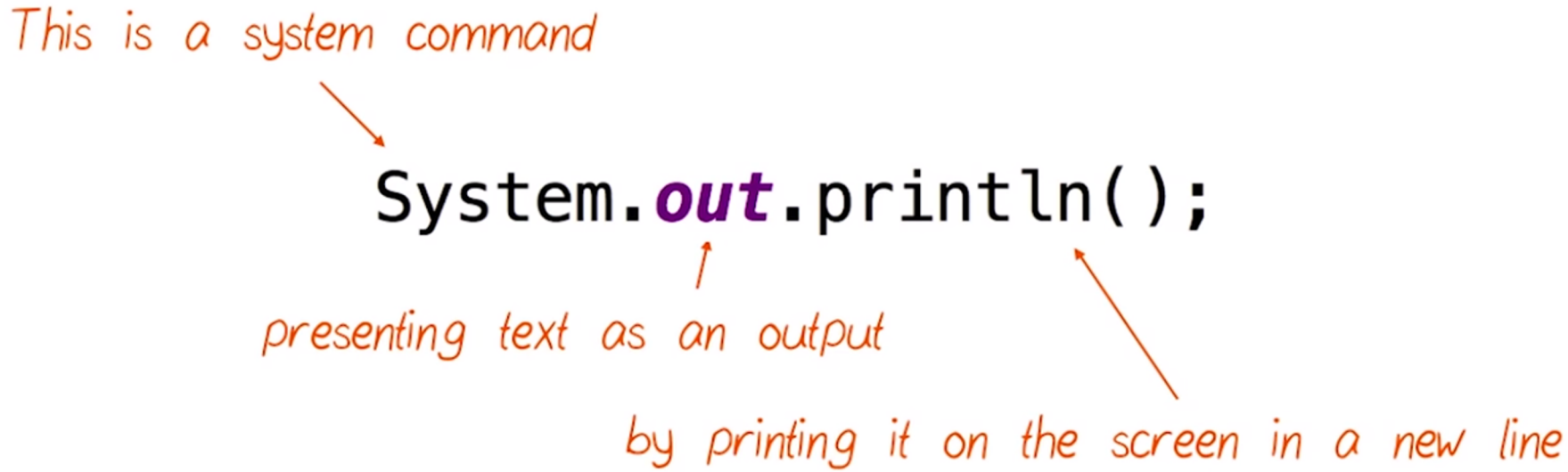
Theory – First program

Main method – the entry point of an application

- public - access modifier
- static – no instance needed to call it
- void – return nothing
- main – name of the method
- String args[] - stores Java *command line arguments* and is an array of type *java.lang.String* class

```java
public class Main {
    public static void main(String args[]) {
        System.out.println("My first Java program");
    }
}
```

Theory – First program



Theory – First program

Change default text, that is printed on the console. E.g. „Hello, Mike!”.

Hello, Mike!

Print the same text twice.

Hello, World!
Hello, World!

Print different text in multiple lines.

Hello, World!
It’s a great day, to learn something new.

Split line in the middle – use only one System.***out***.println method. Tip “\n”

Hello,
World!

Theory – Comments

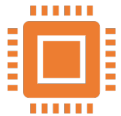
Single line

- Comments can be used to explain Java code, and to make it more readable.
- It can also be used to prevent execution when testing alternative code.
- Single-line comments start with two forward slashes (//).
- Any text between // and the end of the line is ignored by Java (will not be executed)
- Example: // This is a comment

Multi line

- Multi-line comments start with /* and ends with */.
- Any text between /* and */ will be ignored by Java.
- Example: Example: /* The code below will print the words Hello World to the screen, and it is amazing */

Theory – Variables



A variable is a storage location in a computer program.

Each variable has a name and holds a value.

In Java, every variable has a type.

Good practice – use a short, descriptive, meaningful variable name!

variable names are case sensitive

variables names must begin with a letter

cannot have white spaces

the first letter should be lowercase, and then normal CamelCase rules should be used

There are four types of variables in java: block, local, instance, static.

Theory – Variables

```
/*
    Creation
*/

// Syntax
type variable = value;

//Example
int myNum = 15;
System.out.println(myNum);

// Display: println()

// To combine both text and a variable, use the + character
System.out.println("Hello " + name);

//You can also use the + character to add a variable to another variable
String fullName = firstName + lastName;
System.out.println(fullName);
```

Theory – Data types

All Java **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code.

The general rules for constructing names for variables (unique identifiers) are:

Names can contain letters, digits, underscores, and dollar signs

Names must begin with a letter

Names should start with a lowercase letter and it cannot contain whitespace

Names can also begin with $ and _ (but we will not use it in this tutorial)

Names are case sensitive ("myVar" and "myvar" are different variables)

Reserved words (like Java keywords, such as int or boolean) cannot be used as names

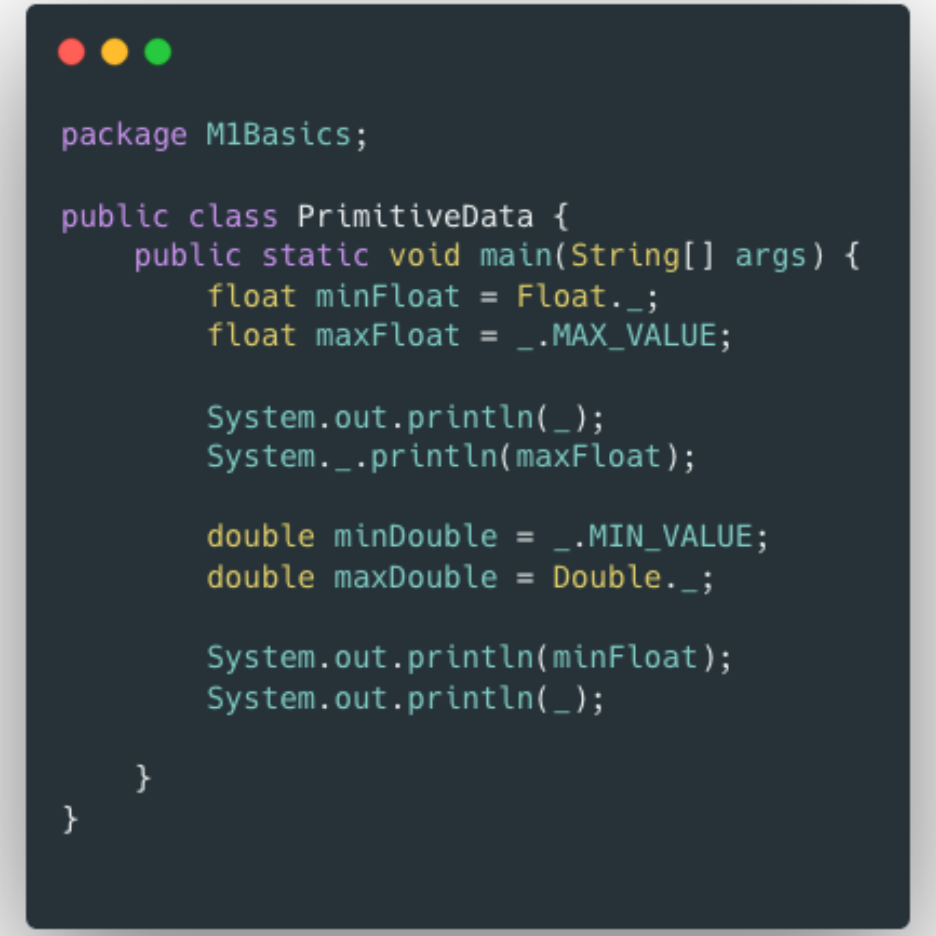
Theory – Primitives Data Types

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	**4 bytes**	**Stores whole numbers from -2,147,483,648 to 2,147,483,647**
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	**8 bytes**	**Stores fractional numbers. Sufficient for storing 15 decimal digits**
boolean	**1 bit**	**Stores true or false values**
char	2 bytes	Stores a single character/letter or ASCII values

Theory – Primitives Data Types

```java
package M1Basics;

public class PrimitiveData {
    public static void main(String[] args) {
        int minIntVal = Integer.MIN_VALUE;
        int maxIntVal = Integer.MAX_VALUE;
        System.out.println(minIntVal);
        System.out.println(maxIntVal);

        byte minByteVal = Byte.MIN_VALUE;
        byte maxByteVal = Byte.MAX_VALUE;
        System.out.println(minByteVal);
        System.out.println(maxByteVal);

        short minShortVal = Short.MIN_VALUE;
        short maxShortVal = Short.MAX_VALUE;
        System.out.println(minShortVal);
        System.out.println(maxShortVal);

        long minLongVal = Long.MIN_VALUE;
        long maxLongVal = Long.MAX_VALUE;
        System.out.println(minLongVal);
        System.out.println(maxLongVal);
    }
}
```

Theory – Primitives Data Types Replace _ with the right code



```java
package M1Basics;

public class PrimitiveData {
    public static void main(String[] args) {
        float minFloat = Float._;
        float maxFloat = _.MAX_VALUE;

        System.out.println(_);
        System._.println(maxFloat);

        double minDouble = _.MIN_VALUE;
        double maxDouble = Double._;

        System.out.println(minFloat);
        System.out.println(_);

    }
}
```

Theory – Non-Primitive Data Types

Basics

Non-primitive data types are called **reference types** because they refer to objects.

Examples of non-primitive types are **Strings, Arrays**, Classes, Interface

String

a collection of characters surrounded by double quotes

String greeting = "Hello";

txt.length();

txt.toUpperCase()

txt.toLowerCase()

txt.indexOf("locate")

firstName.concat(lastName)

Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value

String[] cars;

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars[0]);

System.out.println(cars.length);

Theory – Primitive vs Non-Primitive Data Types

Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).

Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.

A primitive type has always a value, while non-primitive types can be null.

A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.

The size of a primitive type depends on the data type, while non-primitive types have all the same size.

Variables and Data types – Which is an incorrect statement?

```
char me = 'I';
boolean fact = true;
boolean number = 17;
String text = "text";
double price = 23.75;
long total = 100.1;
```

Variables and Data types – What this code snippet will print out

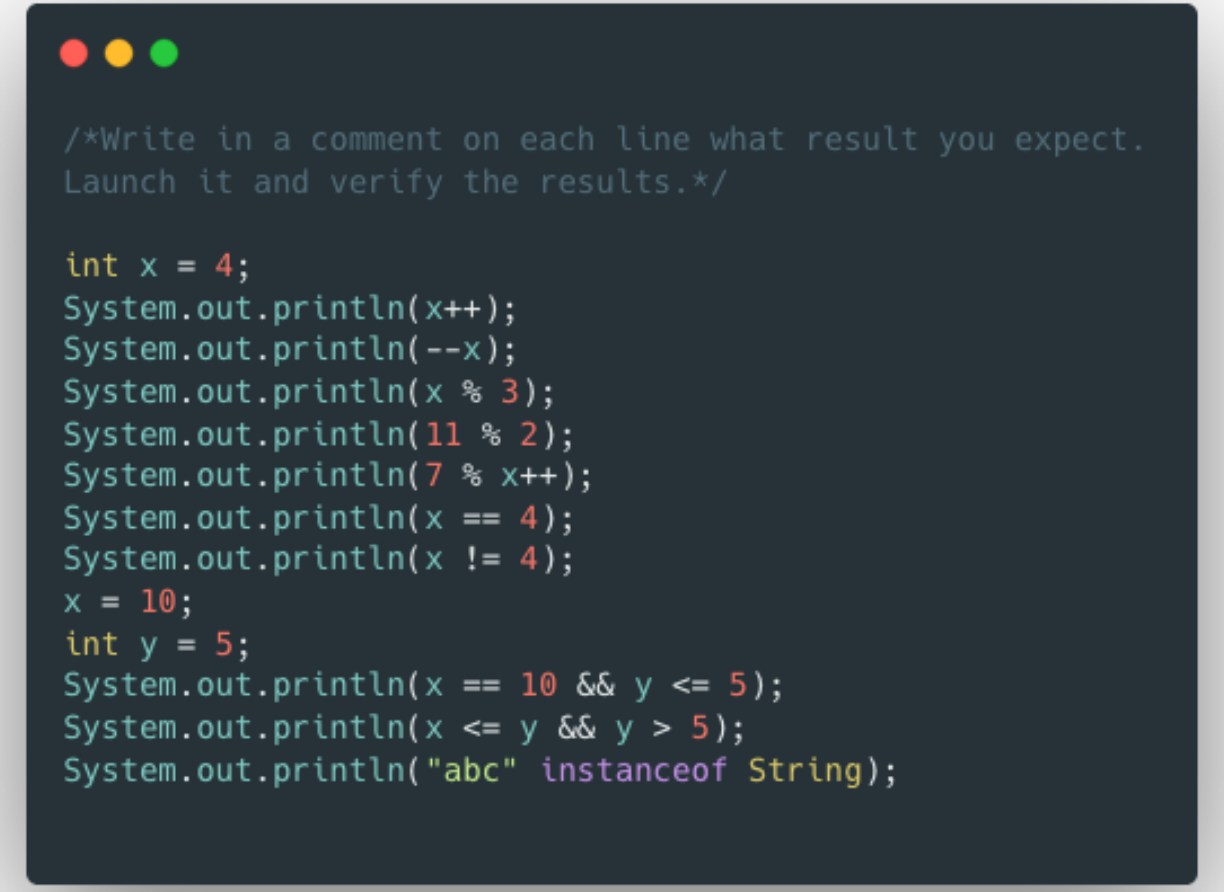
```
int x = 0;
int y = 4;
double z = 3;
x = x + 2;
z = x + y - 7;
y = x * 3;
System.out.println("x = "+x);
System.out.println("y = "+y);
System.out.println("z = "+z);
```

- ○
```
x = 2
y = 0
z = -3.0
```
- ○
```
x = 0
y = 0
z = 1.5
```
- ○
```
x = 0
y = 4
z = 3.0
```
- ○
```
x = 2
y = 6
z = -1.0
```

Theory - Basic operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value from another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Theory - Basic operators

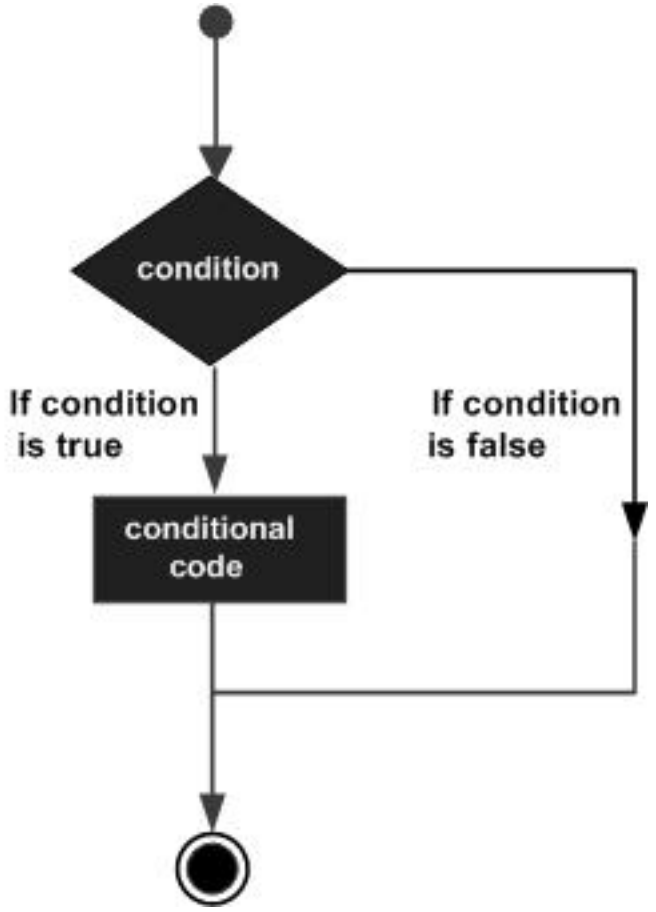


```java
/*Write in a comment on each line what result you expect.
Launch it and verify the results.*/

int x = 4;
System.out.println(x++);
System.out.println(--x);
System.out.println(x % 3);
System.out.println(11 % 2);
System.out.println(7 % x++);
System.out.println(x == 4);
System.out.println(x != 4);
x = 10;
int y = 5;
System.out.println(x == 10 && y <= 5);
System.out.println(x <= y && y > 5);
System.out.println("abc" instanceof String);
```

Theory - Decision making – if

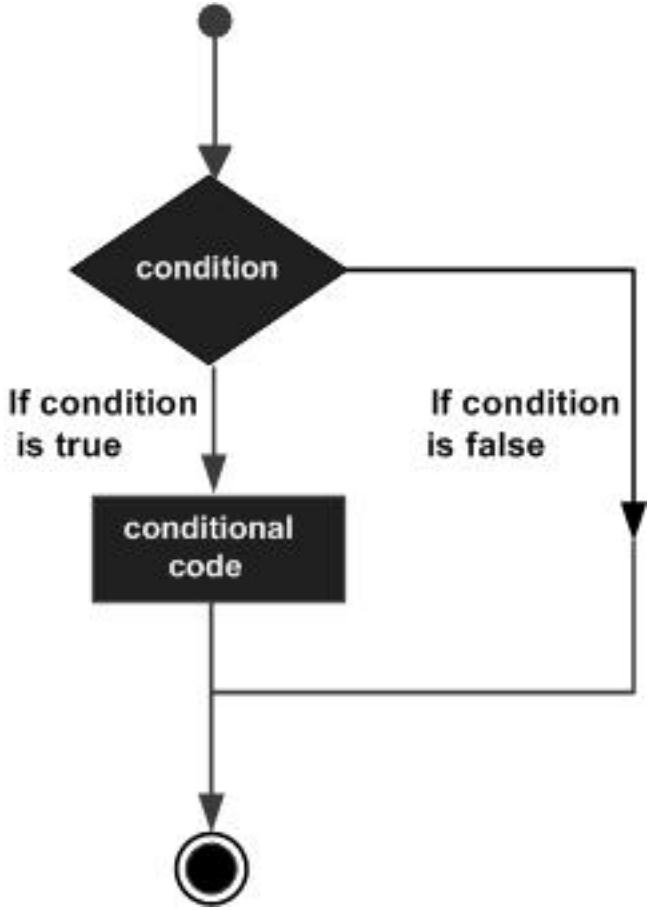
```
if(Boolean_expression) {
   // Statements will execute if the Boolean expression is true
}
```

```
int x = 10;

if( x < 20 ) {
   System.out.print("This is if statement");
}
```



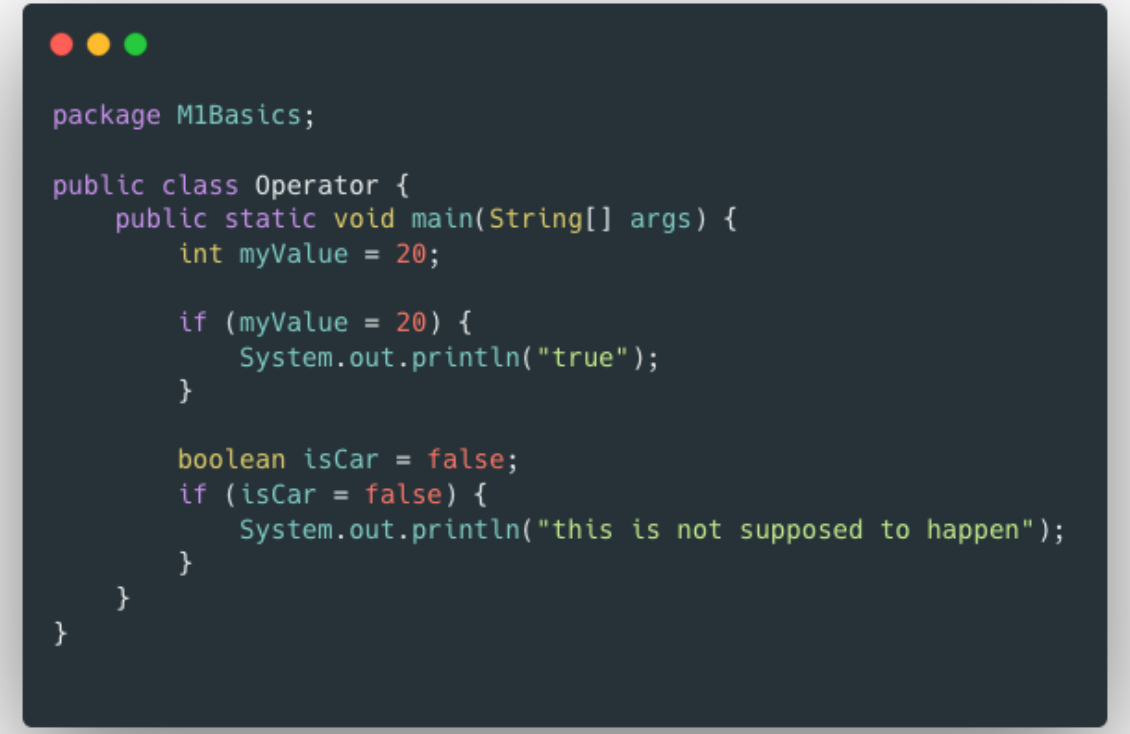
Theory - Decision making – if else

```
if(Boolean_expression) {
   // Executes when the Boolean expression is true
}else {
   // Executes when the Boolean expression is false
}
```




```
int x = 30;

if( x == 10 ) {
   System.out.print("Value of X is 10");
}else if( x == 20 ) {
   System.out.print("Value of X is 20");
}else if( x == 30 ) {
   System.out.print("Value of X is 30");
}else {
   System.out.print("This is else statement");
}
```

Theory - Decision making – if else



```java
package M1Basics;

public class Operator {
    public static void main(String[] args) {
        int myValue = 20;

        if (myValue = 20) {
            System.out.println("true");
        }

        boolean isCar = false;
        if (isCar = false) {
            System.out.println("this is not supposed to happen");
        }
    }
}
```

Theory - Decision making – switch

```
switch(expression) {
   case value :
      // Statements
      break; // optional

   case value :
      // Statements
      break; // optional

   // You can have any number of case statements.
   default : // Optional
      // Statements
}
```

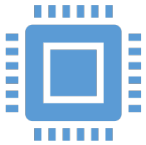
```
char grade = 'C';

switch(grade) {
   case 'A' :
      System.out.println("Excellent!");
      break;
   case 'B' :
   case 'C' :
      System.out.println("Well done");
      break;
   case 'D' :
      System.out.println("You passed");
   case 'F' :
      System.out.println("Better try again");
      break;
   default :
      System.out.println("Invalid grade");
}
System.out.println("Your grade is " + grade);
```

Achievements

Succeeded to implement the first Java project

Understand what main method does

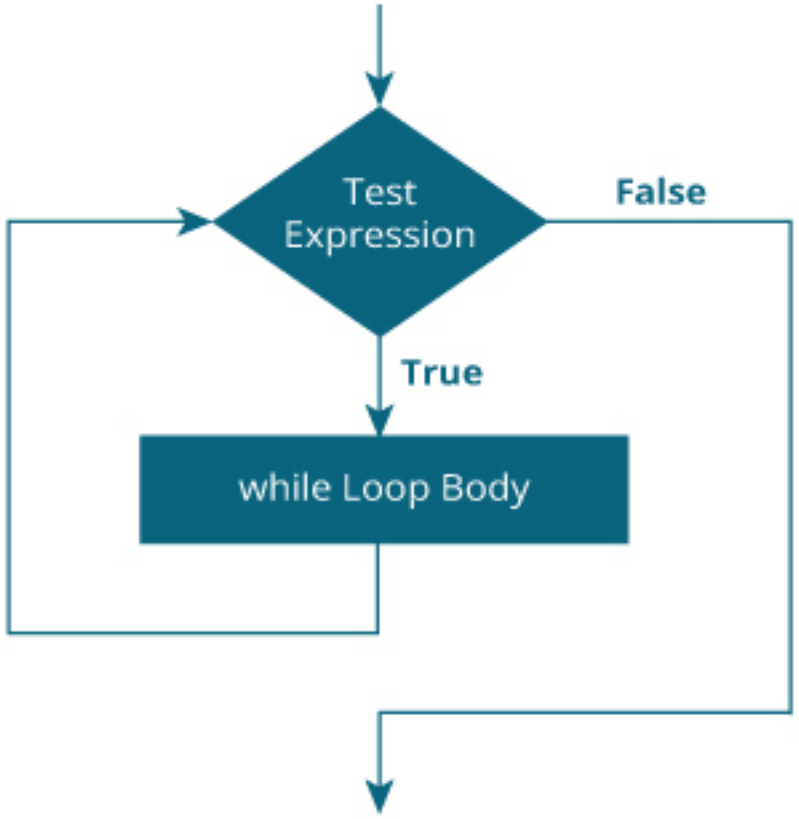
Basic data types understanding



Usage of the operators

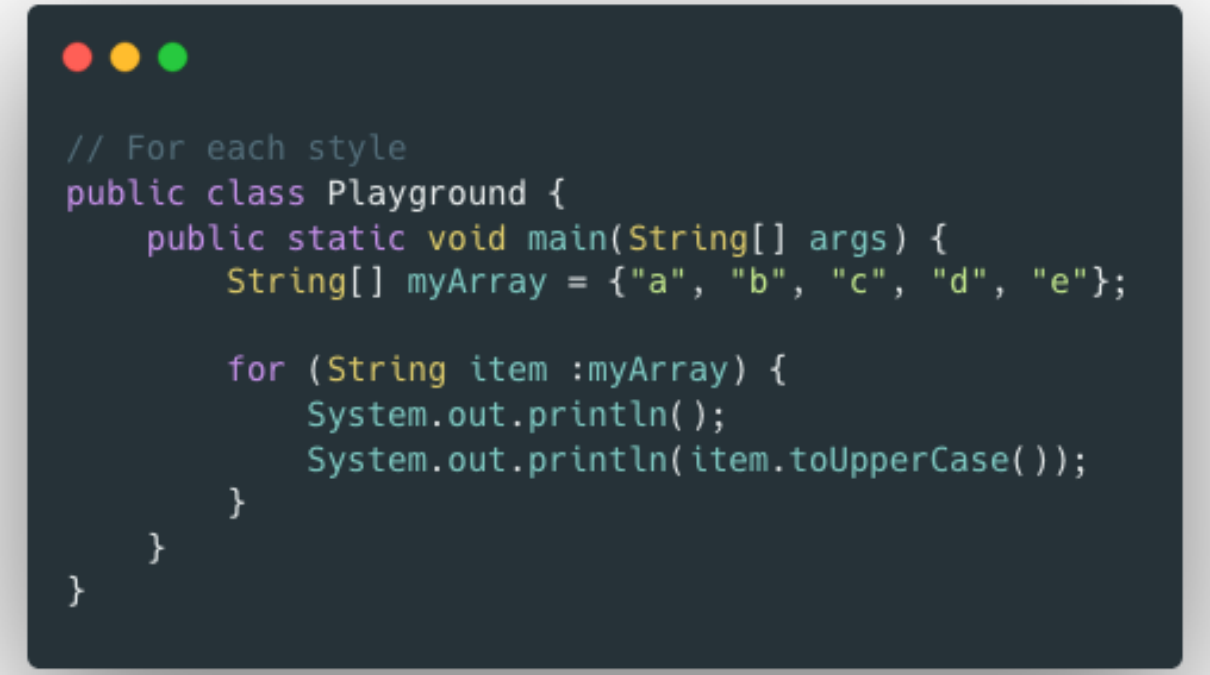
Conditions

Theory – Loop control




```java
// For
for(initialization; condition; update) { // Statements }
// Example
for(int x = 10; x < 20; x++) {
  System.out.print("value of x : " + x );
}
// While
while(condition) { // Statements }
// Example
int x = 10;
while( x < 20 ) {
  System.out.print("value of x : " + x );
  x++;
  System.out.print("\n");
}

// Do while
do { // Statements } while(condition);
// Example
int x = 10;
do {
    System.out.print("value of x : " + x );
    x++;
    System.out.print("\n");
} while( x < 20 );
```

Theory – Loop control



```java
// For each style
public class Playground {
    public static void main(String[] args) {
        String[] myArray = {"a", "b", "c", "d", "e"};

        for (String item :myArray) {
            System.out.println();
            System.out.println(item.toUpperCase());
        }
    }
}
```

Theory – Loop control

```java
import java.util.Arrays;

public class ArrayPlay {
    public static void main(String args[]) {
        String[] classRoom = {"Aurica", "Gigi", "Dorel", "Dorica", "Nicu", "Petre"};
        String[] available = {"Dorel", "Dorica", "Nicu", "Petre"};
        _ nr_of_stud = _;
        for (String student : _) {
            if (Arrays.asList(_).contains(student)) {
                nr_of_stud _= _;
            }
        }

        System.out.println(_);
    }
}
```

Theory – Loop control

```java
import java.util.Arrays;

public class ArrayPlay {
    public static void main(String args[]) {
        String[] classRoom = {"Aurica", "Gigi", "Dorel", "Dorica", "Nicu", "Petre"};
        String[] available = {"Dorel", "Dorica", "Nicu", "Petre"};
        int nr_of_stud = 0;
        for (String student : classRoom) {
            if (Arrays.asList(available).contains(student)) {
                nr_of_stud += 1;
            }
        }

        System.out.println(nr_of_stud);
    }
}
```

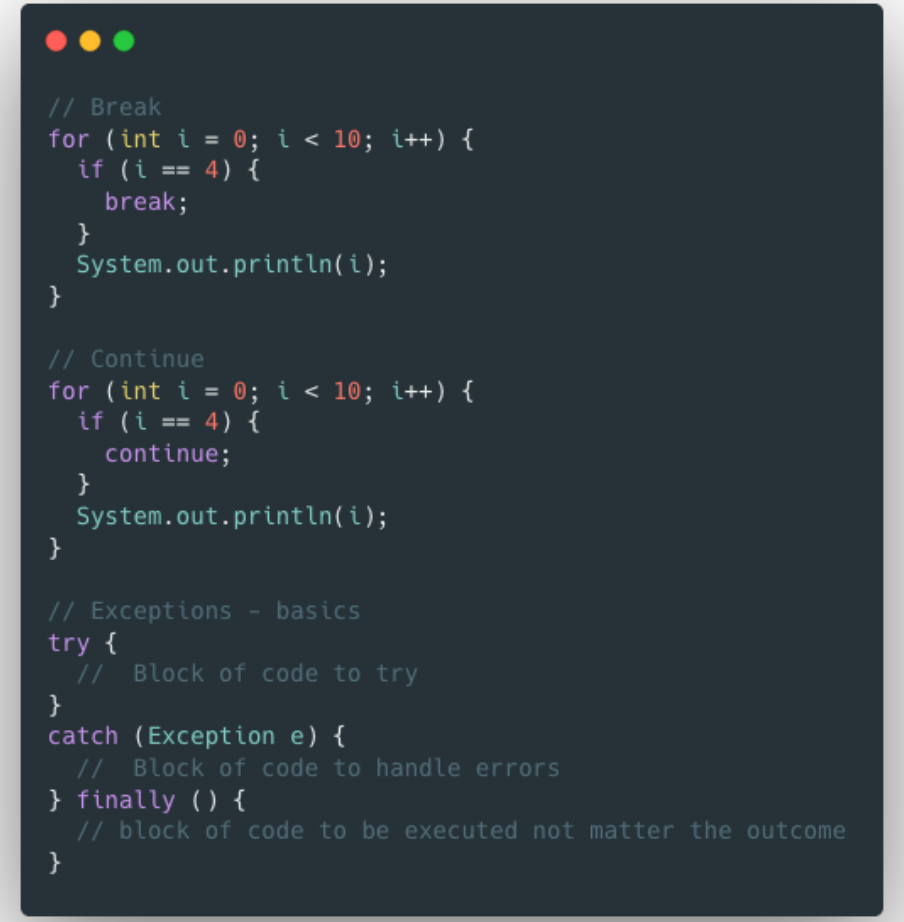
Theory – Exceptions & Break/Continue

Java Break - The break statement can also be used to jump out of a loop.

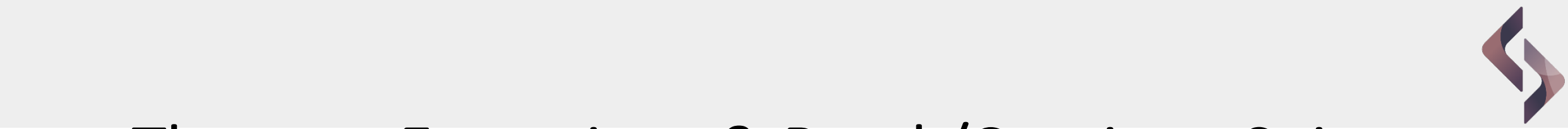
Java Continue – This statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Try – This statement allows you to define a block of code to be tested for errors while it is being executed. The catch statement allows you to define a block of code to be executed, if an error occurs in the try block. The try and catch keywords come in pairs. The finally statement lets you execute code, after try...catch, regardless of the result.

Theory – Exceptions & Break/Continue Snippets



```java
// Break
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  System.out.println(i);
}

// Continue
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}

// Exceptions - basics
try {
  //  Block of code to try
}
catch (Exception e) {
  //  Block of code to handle errors
} finally () {
  // block of code to be executed not matter the outcome
}
```



Theory – Exceptions & Break/Continue Snippets

```java
// Break
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  System.out.println(i);
}

// Continue
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}

// Exceptions - basics
try {
  //  Block of code to try
}
catch (Exception e) {
  //  Block of code to handle errors
} finally () {
  // block of code to be executed not matter the outcome
}
```

```java
public class Try {
    public static void main(String[] args) {
        try {
            int a = 5;
            int b = 0;
            System.out.println(a/b);
        } catch (Exception e) {
            System.out.println(e);
        } finally {
            System.out.println("do this");
        }
    }
}
```

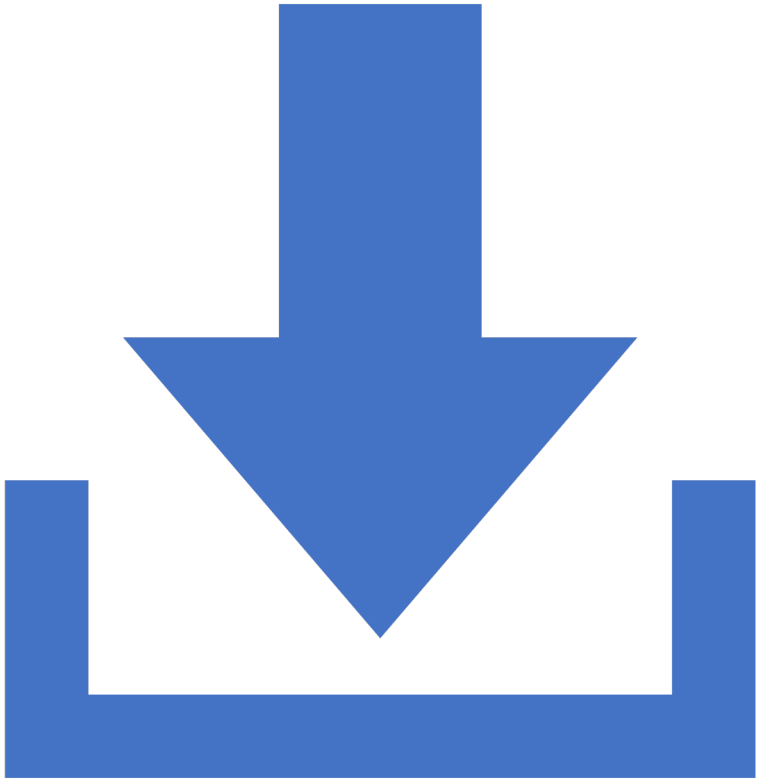
Theory – Scanner

- The Scanner class is used to get user input, and it is found in the java.util package
- To use the Scanner class, create an object of the class

```java
import java.util.Scanner;

public class Scan {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);

        System.out.println("Enter name, age and salary:");

        // String input
        String name = myObj.nextLine();

        // Numerical input
        int age = myObj.nextInt();
        double salary = myObj.nextDouble();

        // Output input by user
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Salary: " + salary);
    }
}
```

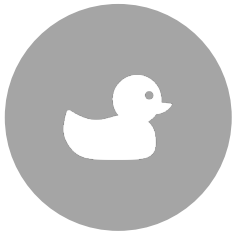
New project

- Go to the repo: https://gitlab.com/sda-international/users/courses/romania/cluj-napoca/tester-cluj-3/module-7---intro-to-programming.git
- Click Clone -> Clone with HTTPS -> Click on the Copy URL icon
- Download, install and open Github desktop
- Click File -> Clone repository -> Select URL -> Paste the content from the second step
- Change the branch to develop -> Select new branch -> select e.g. feature/your_name
- Create a new Java project -> Change project location, where you've downloaded the repo



Let’s play a game - products

CARS



TOYS

FRUITS

CAKES

ANIMALS

GADGETS

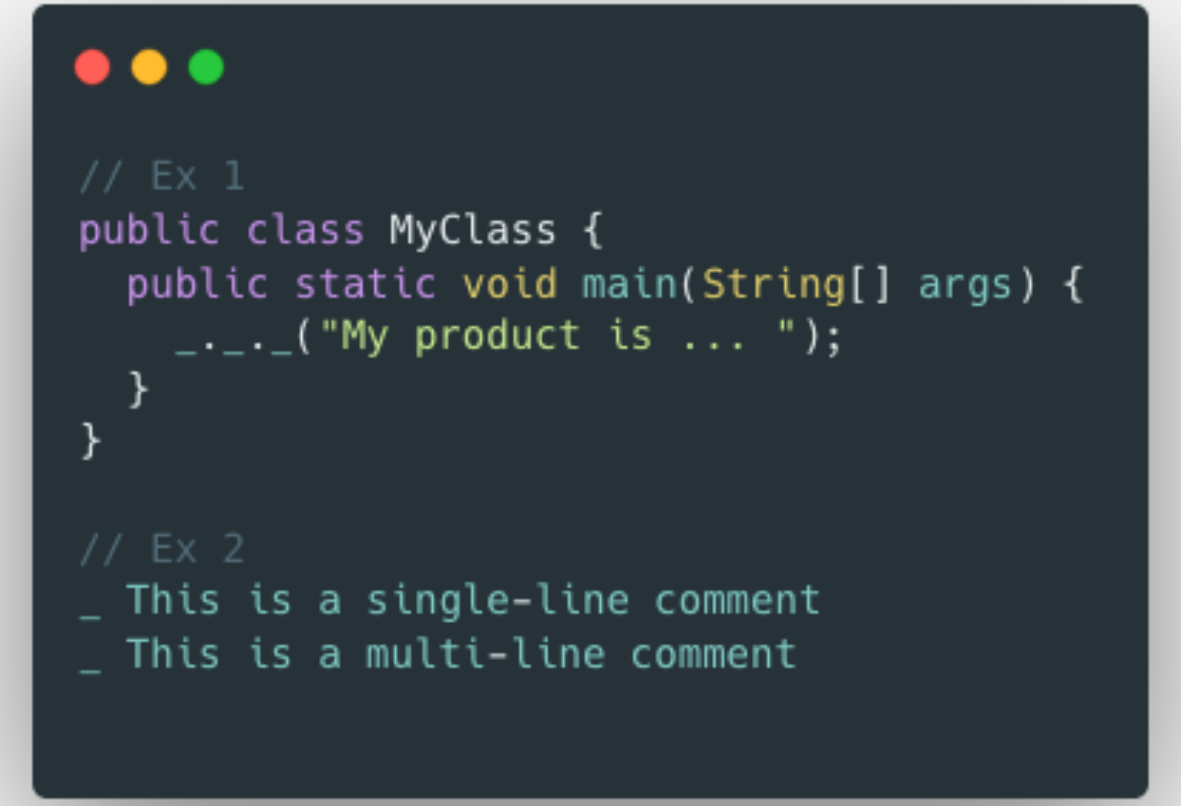
Practice – Basics exercises

Define – a new project

Design - a new class for each chapter to follow

Implement - consider the example from the slide as a reference one, and use every variable, string, int, or other information based on the product defined inside the team

Exercises – Java syntax



```java
// Ex 1
public class MyClass {
  public static void main(String[] args) {
    _._._("My product is ... ");
  }
}

// Ex 2
_ This is a single-line comment
_ This is a multi-line comment
```

Exercises – Java syntax solution

```java
// Ex 1
public class MyClass {
  public static void main(String[] args) {
    System.out.println("My product is Pizza");
  }
}

// Ex 2
// This is a single-line comment
/* This is a multi-line comment
```

Exercises - Variables

```
// Ex 3 Create a variable named "your_product" and assign the value "type"  to it.
_ _ = _;

// Ex 4 Create a variable named "your_variable" and assign the value x (it's an int) to it.
// Example data_type cake = a_number;
_ _ = _;

// Ex 5 Display the sum of 5 + 10, using two variables: x and y.
_ _ = _;
int y = 10;
System.out.println(x + y);

// Ex 6 Create a variable called x, assign x + y to it, and display the result.
int x = 5;
int y = 10;
_ _   = x + y;
System.out.println(_);

// Ex 7 Fill in the missing parts to create three variables of the same type, using a comma-separated
list:
_   x = 5_ y = 6_ z = 50;
System.out.println(x + y + z);
```

Exercises – Variables solution

```java
// Ex 3 Create a variable named "your_product" and assign the value "type"  to it.
String carName = "Dacia";

// Ex 4 Create a variable named "your_variable" and assign the value x (it's an int) to it.
// Example data_type cake = a_number;
int myCake = 12;

// Ex 5 Display the sum of 5 + 10, using two variables: x and y.
int x = 5;
int y = 10;
System.out.println(x + y);

// Ex 6 Create a variable called z, assign x + y to it, and display the result.
int x = 5;
int y = 10;
int z  = x + y;
System.out.println(z);

// Ex 7 Fill in the missing parts to create three variables of the same type, using a comma-separated
list:
int  x = 5, y = 6, z = 50;
System.out.println(x + y + z);
```

Exercises – Data types

```
// Ex 8
// Add the correct data type for the following variables:
// replace the variable names with something related to your product

_ myNum = 9;
_ myFloatNum = 8.99f;
_ myLetter = 'A';
_ myBool = false;
_ myText = "my product";

// Ex 9
// How byte, short, int, long, float, double, boolean and char are called:
_ data types.

// Ex 10
// Type casting - convert the following double type (myDouble) to an int type:

double myDouble = 9.78;
int myInt = _ myDouble;
```

Exercises – Data types solution

```java
// Ex 8
// Add the correct data type for the following variables:
// replace the variable names with something related to your product

int myNum = 9;
float myFloatNum = 8.99f;
char myLetter = 'A';
boolean myBool = false;
String myText = "my product";

// Ex 9
// How byte, short, int, long, float, double, boolean and char are called:
primitive data types.

// Ex 10
// Type casting - convert the following double type (myDouble) to an int type:

double myDouble = 9.78;
int myInt = (int) myDouble;
```

Exercises – Operators

```java
// Ex 11
// Multiply 10 with 5, and print the result.
System.out.println(10 _ 5);

// Ex 12
// Divide 10 by 5, and print the result.
System.out.println(10 _ 5);

// Ex 13
// Use the correct operator to increase the value of the variable x by 1.
int x = 10;
_x;

// Ex 14
// Use the addition assignment operator to add the value 5 to the variable x.
int x = 10;
x _ 5;
```

Exercises – Operators solution

```java
// Ex 11
// Multiply 10 with 5, and print the result.
System.out.println(10 * 5);

// Ex 12
// Divide 10 by 5, and print the result.
System.out.println(10 / 5);

// Ex 13
// Use the correct operator to increase the value of the variable x by 1.
int x = 10;
++x;

// Ex 14
// Use the addition assignment operator to add the value 5 to the variable x.
int x = 10;
x += 5;
```

Exercises – Strings

```
// Ex 15
// Fill in the missing part to create a greeting variable of type String and assign it the value Hello.
_ greeting = _;

// Ex 16
// Use the correct method to print the length of the txt string.
String txt = "Hello";
System.out.println(_._);

// Ex 17
// Convert the value of txt to upper case.
String txt = "Hello";
System.out.println(_._);

// Ex 18
// Use the correct operator to concatenate two strings:
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName _ lastName);

// Ex 19
// Use the correct method to concatenate two strings:
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName._(lastName));

// Ex 20
// Return the index (position) of the first occurrence of "e" in the following string:
String txt = "Hello Everybody";
System.out.println(txt._(_));
```

Exercises – Strings solution

```java
// Ex 15
// Fill in the missing part to create a greeting variable of type String and assign it the value Hello.
String greeting = "Hello";

// Ex 16
// Use the correct method to print the length of the txt string.
String txt = "Hello";
System.out.println(txt.length());

// Ex 17
// Convert the value of txt to upper case.
String txt = "Hello";
System.out.println(txt.toUpperCase());

// Ex 18
// Use the correct operator to concatenate two strings:
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName + lastName);

// Ex 19
// Use the correct method to concatenate two strings:
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName.concat(lastName));

// Ex 20
// Return the index (position) of the first occurrence of "e" in the following string:
String txt = "Hello Everybody";
System.out.println(txt.indexOf("e"));
```

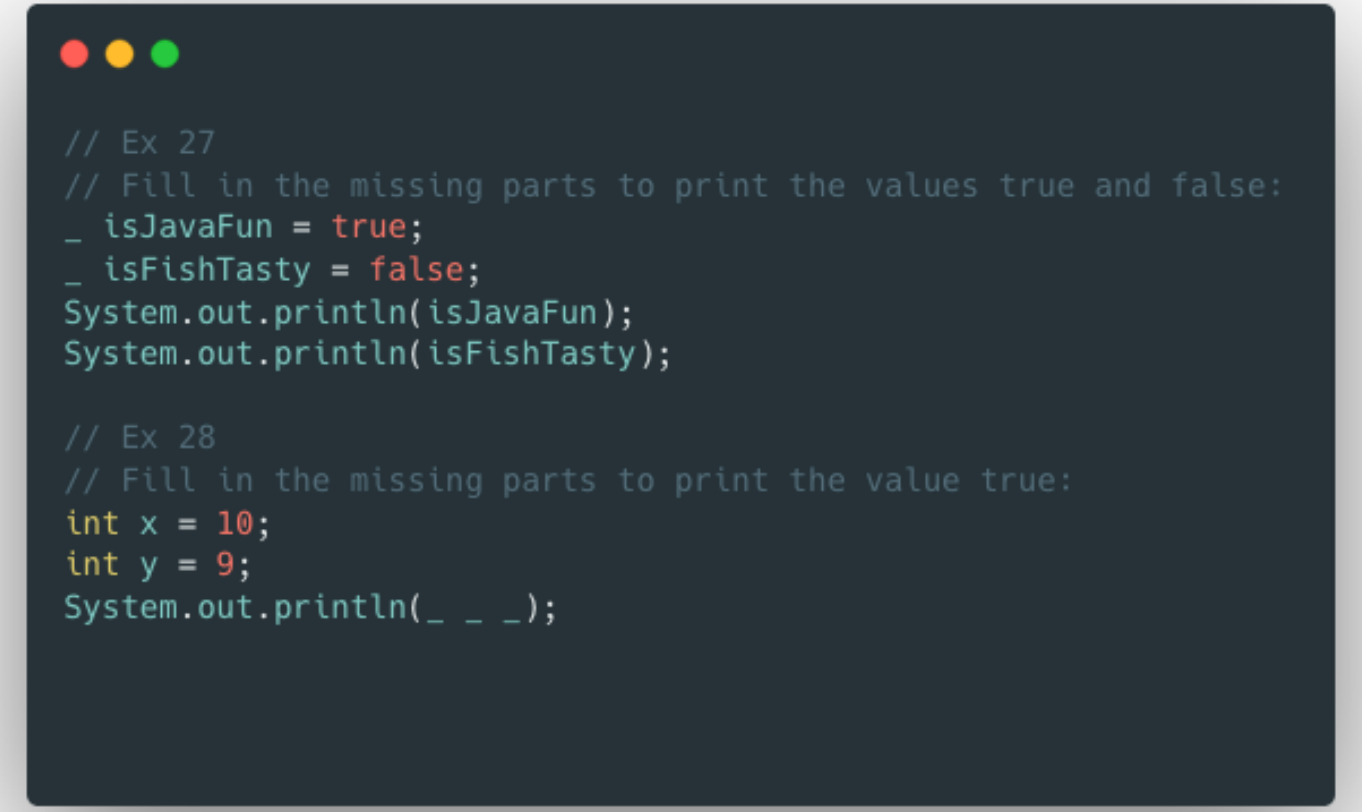
Exercises – Arrays

```
// Ex 21
// Create an array of type String called cars.
_ _  = {"Volvo", "BMW", "Ford"};

// Ex 22
// Print the second item in the cars array.
String[] cars = {"Volvo", "BMW", "Ford"};
System.out.println(_);

// Ex 23
// Change the value from "Volvo" to "Opel", in the cars array.
String[] cars = {"Volvo", "BMW", "Ford"};
_ = _;
System.out.println(cars[0]);

// Ex 24
// Find out how many elements the cars array have.
String[] cars = {"Volvo", "BMW", "Ford"};
System.out.println(_);

// Ex 25
// Loop through the items in the cars array.
String[] cars = {"Volvo", "BMW", "Ford"};
_ (String i : _ ) {
  System.out.println(i);
}

// Ex 26
// Insert the missing parts to create a two-dimensional array.
_ myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

Exercises – Arrays solution

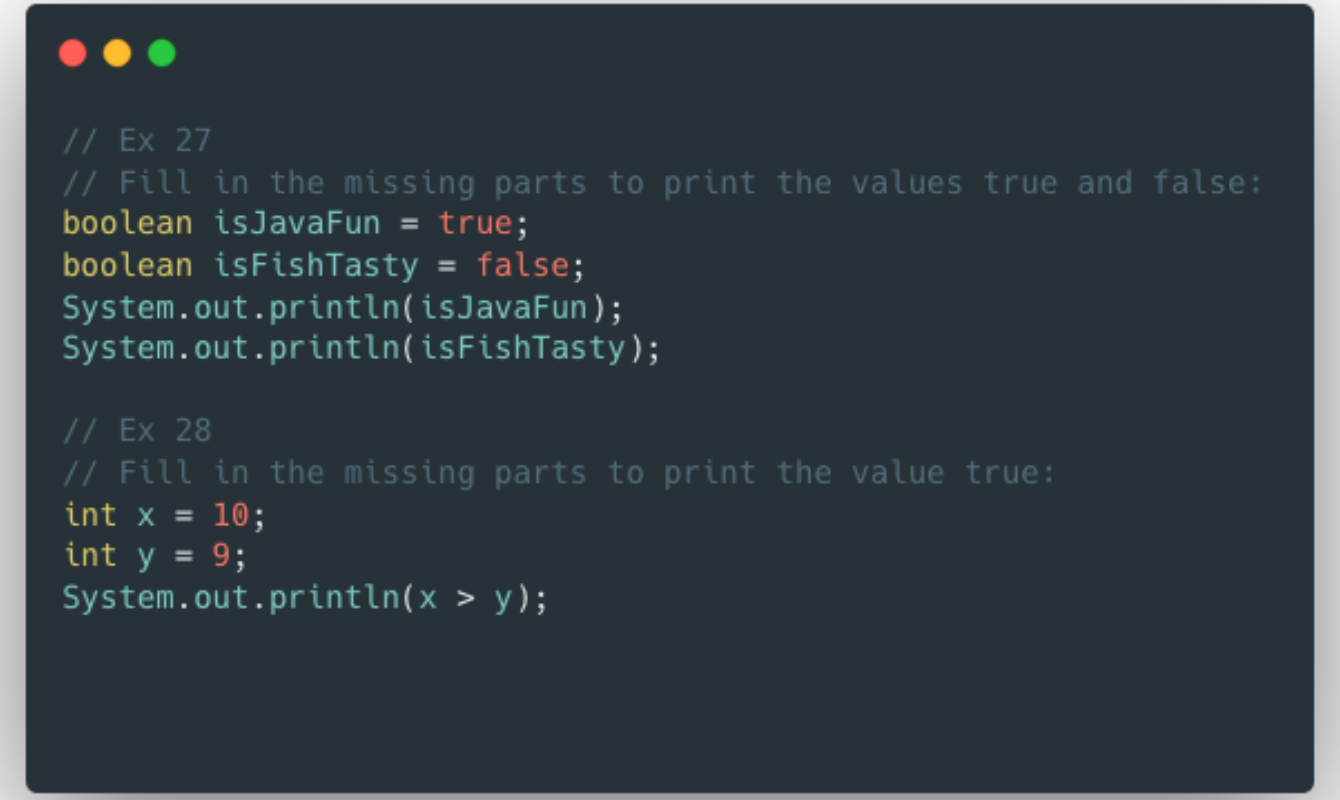
```java
// Ex 21
// Create an array of type String called cars.
String[] cars  = {"Volvo", "BMW", "Ford"};

// Ex 22
// Print the second item in the cars array.
String[] cars = {"Volvo", "BMW", "Ford"};
System.out.println(cars[1]);

// Ex 23
// Change the value from "Volvo" to "Opel", in the cars array.
String[] cars = {"Volvo", "BMW", "Ford"};
cars[0] = "Opel";
System.out.println(cars[0]);

// Ex 24
// Find out how many elements the cars array have.
String[] cars = {"Volvo", "BMW", "Ford"};
System.out.println(cars.length);

// Ex 25
// Loop through the items in the cars array.
String[] cars = {"Volvo", "BMW", "Ford"};
for (String i : cars ) {
  System.out.println(i);
}

// Ex 26
// Insert the missing parts to create a two-dimensional array.
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

Exercises – Booleans



```
// Ex 27
// Fill in the missing parts to print the values true and false:
_ isJavaFun = true;
_ isFishTasty = false;
System.out.println(isJavaFun);
System.out.println(isFishTasty);

// Ex 28
// Fill in the missing parts to print the value true:
int x = 10;
int y = 9;
System.out.println(_ _ _);
```

Exercises – Booleans solution



```java
// Ex 27
// Fill in the missing parts to print the values true and false:
boolean isJavaFun = true;
boolean isFishTasty = false;
System.out.println(isJavaFun);
System.out.println(isFishTasty);

// Ex 28
// Fill in the missing parts to print the value true:
int x = 10;
int y = 9;
System.out.println(x > y);
```

Exercises – if else

```
// Ex 29
// Print "Hello World" if x is greater than y.
int x = 50;
int y = 10;
_ (x _ y) {
  System.out.println("Hello World");
}

// Ex 30
// Print "Hello World" if x is equal to y.
int x = 50;
int y = 50;
_ (x _ y) {
  System.out.println("Hello World");
}

// Ex 31
// Print "Yes" if x is equal to y, otherwise print "No".
int x = 50;
int y = 50;
_ (x _ y) {
  System.out.println("Yes");
} _ {
  System.out.println("No");
}

// Ex 32
// Print "1" if x is equal to y, print "2" if x is greater than y, otherwise print "3".
int x = 50;
int y = 50;
_ (x _ y) {
  System.out.println("1");
} _ (x > y) {
  System.out.println("2");
} _ {
  System.out.println("3");
}
```

Exercises – if else solution

```java
// Ex 29
// Print "Hello World" if x is greater than y.
int x = 50;
int y = 10;
if (x > y) {
  System.out.println("Hello World");
}

// Ex 30
// Print "Hello World" if x is equal to y.
int x = 50;
int y = 50;
if (x == y) {
  System.out.println("Hello World");
}

// Ex 31
// Print "Yes" if x is equal to y, otherwise print "No".
int x = 50;
int y = 50;
if (x > y) {
  System.out.println("Yes");
} else {
  System.out.println("No");
}

// Ex 32
// Print "1" if x is equal to y, print "2" if x is greater than y, otherwise print "3".
int x = 50;
int y = 50;
if (x == y) {
  System.out.println("1");
} else if (x > y) {
  System.out.println("2");
} else {
  System.out.println("3");
}
```

Exercises – switch

```java
// Ex 33
// Insert the missing parts to complete the following switch statement.
int day = 4;
switch (_) {
  _ 1:
    System.out.println("Saturday");
    break;
  _ 2:
    System.out.println("Sunday");
    _;
}

// Ex 34
// Complete the switch statement, and add the correct keyword at the end to specify some code to run if
there is no case match in the switch statement.
int day = 4;
switch (_) {
  _ 1:
    System.out.println("Saturday");
    break;
  _ 2:
    System.out.println("Sunday");
   _;
  _:
    System.out.println("Weekend");
}
```

Exercises – switch solution

```java
// Ex 33
// Insert the missing parts to complete the following switch statement.
int day = 4;
switch (day) {
  case 1:
    System.out.println("Saturday");
    break;
  case 2:
    System.out.println("Sunday");
    break;
}

// Ex 34
// Complete the switch statement, and add the correct keyword at the end to specify some code to run if
there is no case match in the switch statement.
int day = 4;
switch (day) {
  case 1:
    System.out.println("Saturday");
    break;
  case 2:
    System.out.println("Sunday");
   break;
  default:
    System.out.println("Weekend");
}
```

Exercises – loops

```java
// Ex 35
// Print i as long as i is less than 6.
int i = 1;
_ (i < 6) {
  System.out.println(i);
  _;
}

// Ex 36
// Use the do/while loop to print i as long as i is less than 6.
int i = 1;
_ {
  System.out.println(i);
  i++;
}
_ (i < 6);

// Ex 37
// Use a for loop to print "Yes" 5 times:
_ (int i = 0; i < 5; _) {
  System.out.println(_);
}

// Ex 38
// Loop through the items in the cars array.
String[] cars = {"Volvo", "BMW", "Ford"};
_ (String i : _) {
  System.out.println(i);
}

// Ex 39
// Stop the loop if i is 5.
for (int i = 0; i < 10; i++) {
  if (i == 5) {
    _ ;
  }
  System.out.println(i);
}

// Ex 40
// In the loop, when the value is "4", jump directly to the next value.
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    _;
  }
  System.out.println(i);
}
```

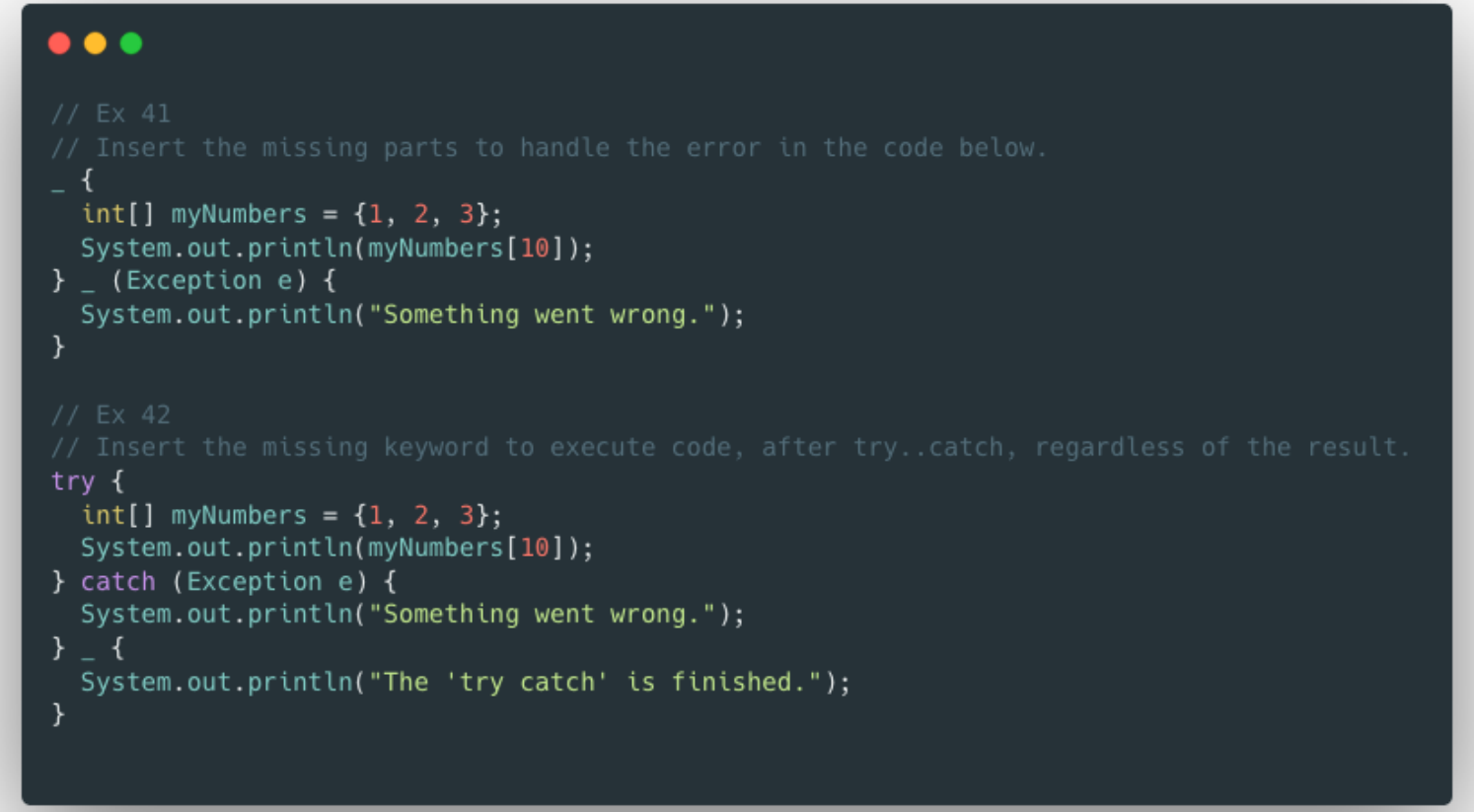
Exercises – loops solution

```java
// Ex 35
// Print i as long as i is less than 6.
int i = 1;
while (i < 6) {
  System.out.println(i);
  i++;
}

// Ex 36
// Use the do/while loop to print i as long as i is less than 6.
int i = 1;
do {
  System.out.println(i);
  i++;
}
while (i < 6);

// Ex 37
// Use a for loop to print "Yes" 5 times:
for (int i = 0; i < 5; i++) {
  System.out.println("Yes");
}

// Ex 38
// Loop through the items in the cars array.
String[] cars = {"Volvo", "BMW", "Ford"};
for (String i : cars) {
  System.out.println(i);
}

// Ex 39
// Stop the loop if i is 5.
for (int i = 0; i < 10; i++) {
  if (i == 5) {
   break;
  }
  System.out.println(i);
}

// Ex 40
// In the loop, when the value is "4", jump directly to the next value.
for (int i = 0; i < 10; i++) {
  if (i == 4) {
   continue;
  }
  System.out.println(i);
}
```

Exercises – loops solution

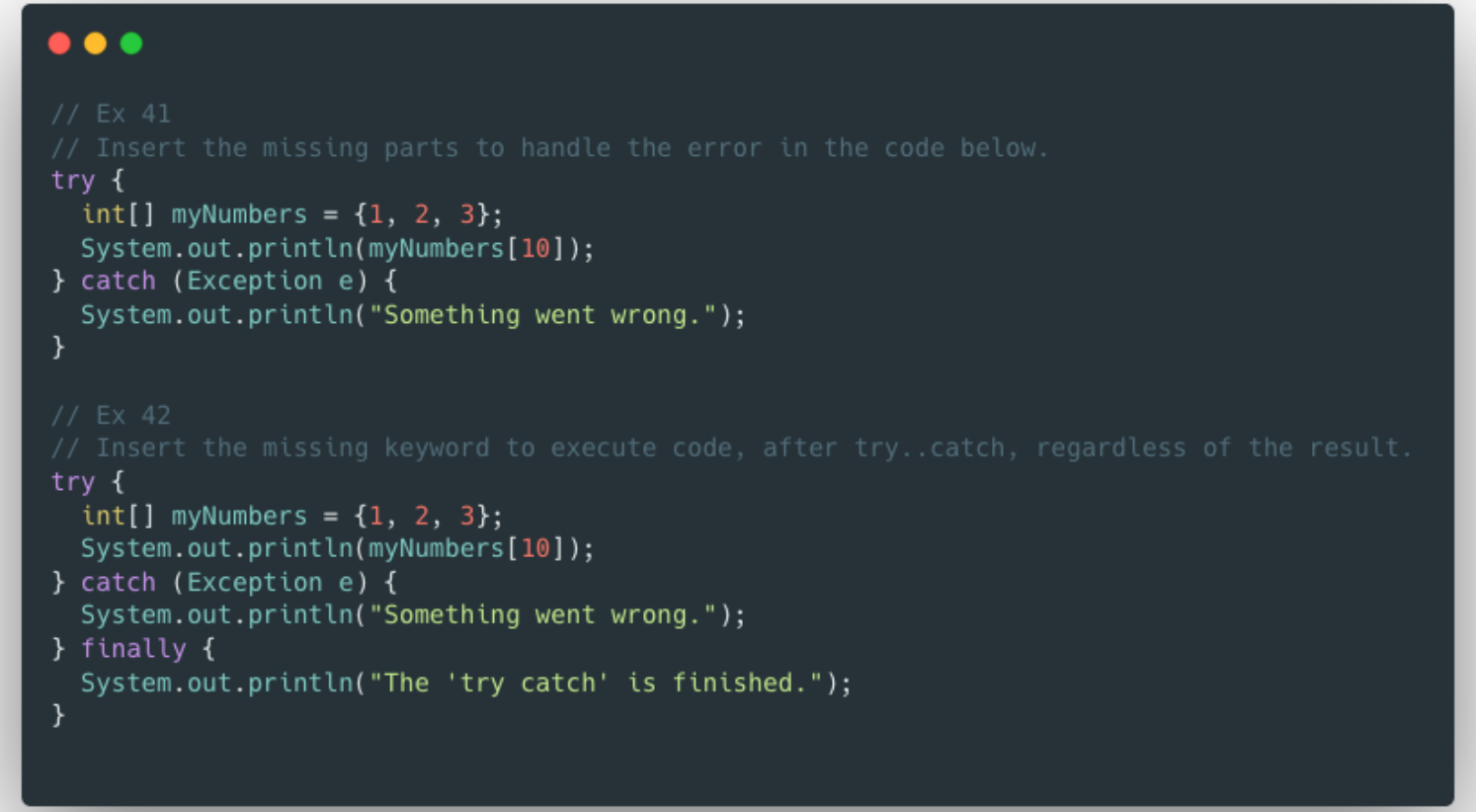
```java
// Ex 35
// Print i as long as i is less than 6.
int i = 1;
while (i < 6) {
  System.out.println(i);
  i++;
}

// Ex 36
// Use the do/while loop to print i as long as i is less than 6.
int i = 1;
do {
  System.out.println(i);
  i++;
}
while (i < 6);

// Ex 37
// Use a for loop to print "Yes" 5 times:
for (int i = 0; i < 5; i++) {
  System.out.println("Yes");
}

// Ex 38
// Loop through the items in the cars array.
String[] cars = {"Volvo", "BMW", "Ford"};
for (String i : cars) {
  System.out.println(i);
}

// Ex 39
// Stop the loop if i is 5.
for (int i = 0; i < 10; i++) {
  if (i == 5) {
   break;
  }
  System.out.println(i);
}

// Ex 40
// In the loop, when the value is "4", jump directly to the next value.
for (int i = 0; i < 10; i++) {
  if (i == 4) {
   continue;
  }
  System.out.println(i);
}
```

Exercises – exceptions



```
// Ex 41
// Insert the missing parts to handle the error in the code below.
_ {
  int[] myNumbers = {1, 2, 3};
  System.out.println(myNumbers[10]);
} _ (Exception e) {
  System.out.println("Something went wrong.");
}

// Ex 42
// Insert the missing keyword to execute code, after try..catch, regardless of the result.
try {
  int[] myNumbers = {1, 2, 3};
  System.out.println(myNumbers[10]);
} catch (Exception e) {
  System.out.println("Something went wrong.");
} _ {
  System.out.println("The 'try catch' is finished.");
}
```

Exercises – exceptions solution



```java
// Ex 41
// Insert the missing parts to handle the error in the code below.
try {
  int[] myNumbers = {1, 2, 3};
  System.out.println(myNumbers[10]);
} catch (Exception e) {
  System.out.println("Something went wrong.");
}

// Ex 42
// Insert the missing keyword to execute code, after try..catch, regardless of the result.
try {
  int[] myNumbers = {1, 2, 3};
  System.out.println(myNumbers[10]);
} catch (Exception e) {
  System.out.println("Something went wrong.");
} finally {
  System.out.println("The 'try catch' is finished.");
}
```

Theory - what you need to know in the initial stage

- Classes – a template that describes the behavior of an object
- Describes the behavior/state that the object of its type support.
- Object - an instance of a class

```
BasicOperators bo = new BasicOperators();
int output = bo.add(2, 3);
System.out.println(output);
```

```
class BasicOperators {
    int add(int a, int b) {
        return a + b;
    }
    int dif(int a, int b) {
        return a - b;
    }

    int div(int a, int b) {
        return a/b;
    }

    int modulo(int a, int b) {
        return a%b;
    }
}
```

Theory - what you need to know in the initial stage

- Constructor – a special method which initializes an object of a given type

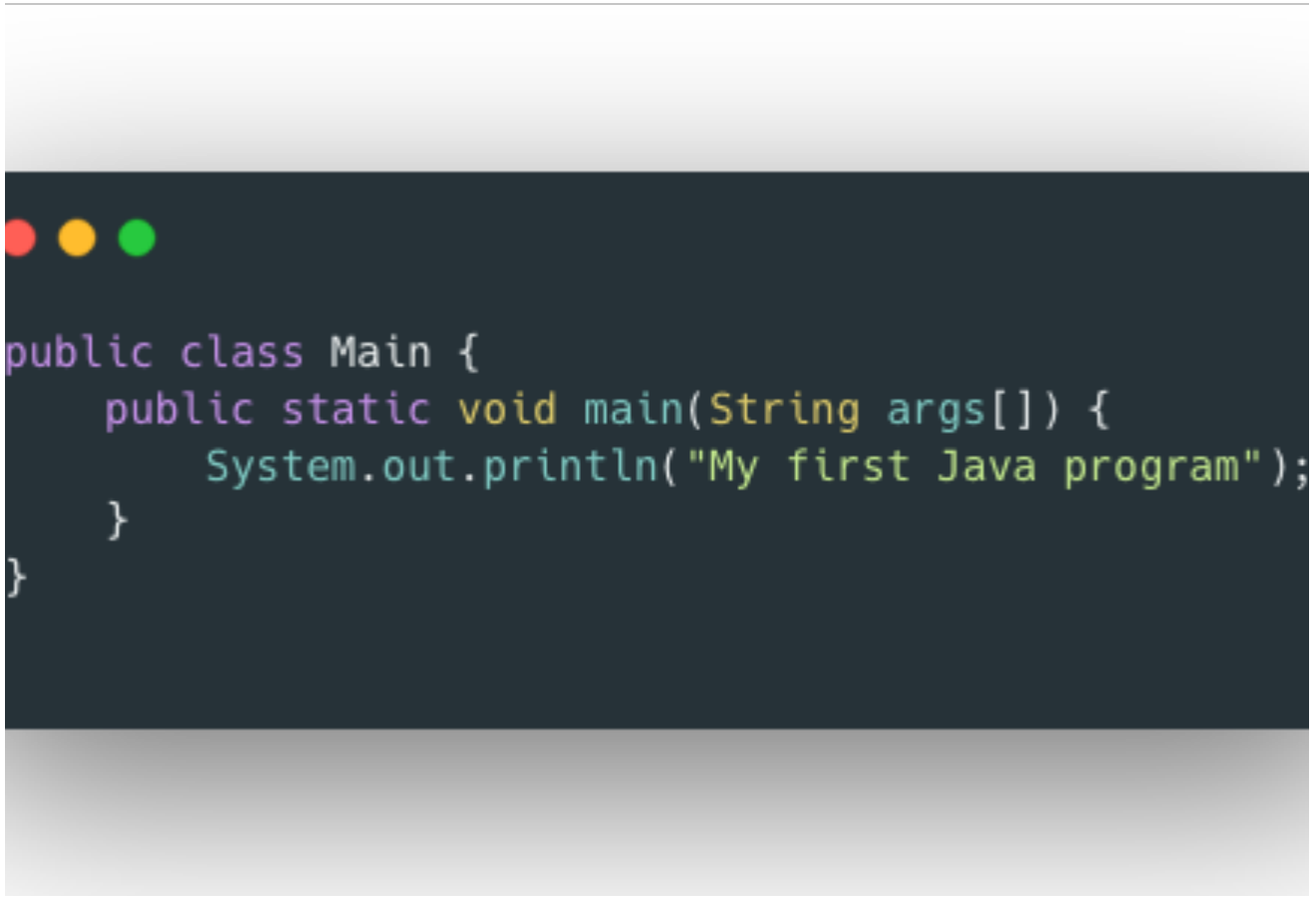
```
BasicOperators bo = new BasicOperators(5, 3);
int output = bo.modulo();
System.out.println(output);
```

```
class BasicOperators {
    private int a;
    private int b;

    BasicOperators(int a, int b) {
        this.a = a;
        this.b = b;
    }

    int add(int a, int b) {
        return a + b;
    }

    int dif(int a, int b) {
        return a - b;
    }

    int div(int a, int b) {
        return a / b;
    }

    int modulo() {
        return this.a % this.b;
    }
}
```

Theory – OOP

What you need to know in the initial stage

- Main method – the entry point of an application
  - public - access modifier
  - static – no instance needed to call it
  - void – return nothing
  - main – name of the method
  - String args[] - stores Java *command line arguments* and is an array of type *java.lang.String* class



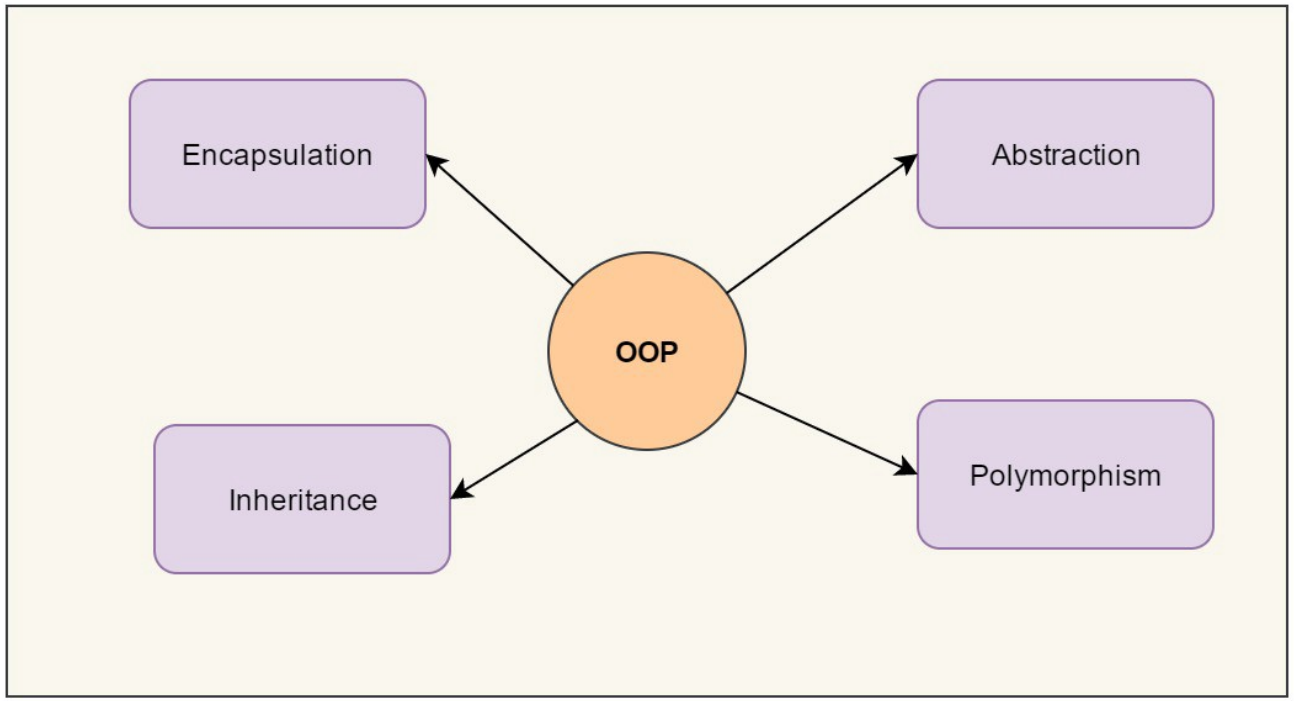
Theory – OOP
Package

Package is a mechanism for combining logically related classes and interfaces into a whole. They provide access control to classes and interfaces and their components. This helps to avoid name collisions. Java libraries are organized into packages. The affiliation of classes and interfaces defined in a given file to a particular package can be determined by providing the package declaration as the first entry in the source file.package package-name; further-source-file-part

Packages are used in Java in order:

- to prevent naming conflicts
- to control access
- to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

Theory - OOP Principles



Four Pillars of Object Oriented Programming

InstanceOf

```java
// Implement an empty class MyModel
public class MyModel {
}

// Create M1 and M2 classes by extending MyModel class
class M1 extends MyModel {
    void doThis() {
        System.out.println("doThis");
    }
}

class M2 extends MyModel {
    void doHere() {
        System.out.println("doHere");
    }
}
```

```java
// Create a new class e.g. ClassInstance and define the main method

public class ClassInstance {
    public static void main(String[] args){
        System.out.println(!("" instanceof String));
        M1 m1 = new M1();
        System.out.println(m1 instanceof M1);
        System.out.println(m1 instanceof MyModel);
        M2 m2 = new M2();
        System.out.println(m2 instanceof M2);

        // Below line causes compile time error:-
        // Incompatible conditional operand types
        // Integer and String
        Integer i = 5;
        System.out.println(i instanceof String);
    }
}
```

Theory – OOP Principles

Encapsulation

Information hiding

There are four access controls (levels of access) but only three access modifiers:

public - visible to the world

protected - visible to the package and all subclasses

default - visible to the package

private - visible to the class only

Theory – OOP Principles

Encapsulation

SAY WE HAVE A PROGRAM. IT HAS A FEW LOGICALLY DIFFERENT OBJECTS WHICH COMMUNICATE WITH EACH OTHER — ACCORDING TO THE RULES DEFINED IN THE PROGRAM.

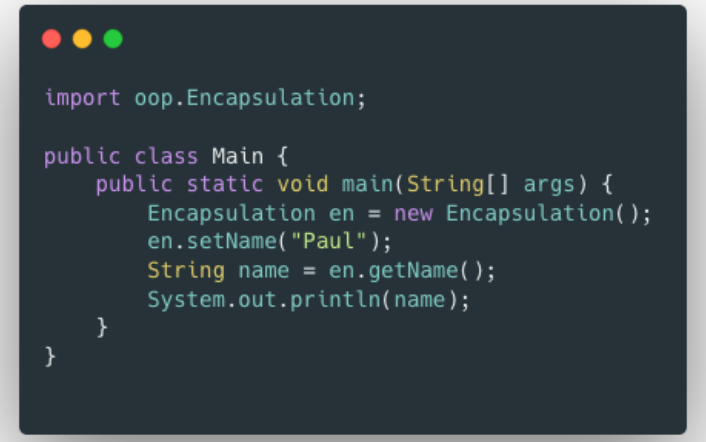
ENCAPSULATION IS ACHIEVED WHEN EACH OBJECT KEEPS ITS STATE **PRIVATE**, INSIDE A CLASS. OTHER OBJECTS DON'T HAVE DIRECT ACCESS TO THIS STATE. INSTEAD, THEY CAN ONLY CALL A LIST OF PUBLIC FUNCTIONS — CALLED METHODS.

SO, THE OBJECT MANAGES ITS OWN STATE VIA METHODS — AND NO OTHER CLASS CAN TOUCH IT UNLESS EXPLICITLY ALLOWED. IF YOU WANT TO COMMUNICATE WITH THE OBJECT, YOU SHOULD USE THE METHODS PROVIDED. BUT (BY DEFAULT), YOU CAN'T CHANGE THE STATE.

Theory – OOP Principles Encapsulation

- Create a Java package e.g. “oop”
- Create a new class called “Encapsulation”
- Insert the code from the code snippet listed in the left side
- Create two new Main classes
  - One inside the package
  - One under the src

```java
package oop;

public class Encapsulation {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    protected int getAge(int myAge) {
        return age = myAge;
    }

    protected void concatIdAndName(String myName, int myId) {
        String newString = myName + " " + myId;
        System.out.println(newString);
    }

    void tryDefault() {
        System.out.println("default");
    }

}
```



```java
import oop.Encapsulation;

public class Main {
    public static void main(String[] args) {
        Encapsulation en = new Encapsulation();
        en.setName("Paul");
        String name = en.getName();
        System.out.println(name);
    }
}
```

```java
package oop;

public class Main {
    public static void main(String[] args) {
        Encapsulation en = new Encapsulation();
        en.concatIdAndName("Paul", 31);
        en.tryDefault();
    }
}
```

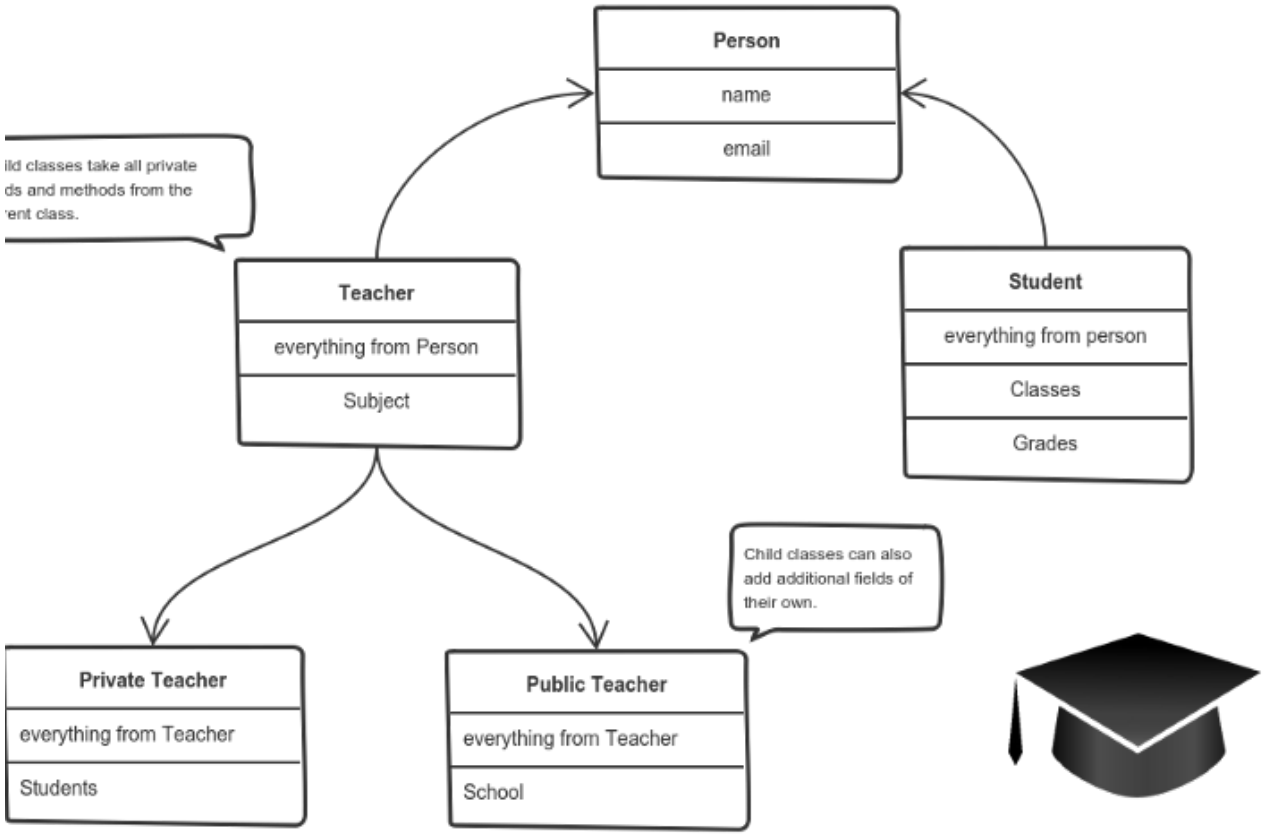
Theory – OOP Principles Encapsulation

- Go to the class available outside of your package
- Instantiate the Encapsulation class. Code available in the first image
- Set name using the setter ”setName”
- Get name using the getter “getName”
- Secondly go the the main class inside the package and instantiate again the Encapsulation.
- Call concatIdAndName method
- Call default method also “tryDefault”

Theory – OOP Principles

Inheritance

- Objects are often very similar. They share common logic. But they're not **entirely** the same. Ugh…
- So how do we reuse the common logic and extract the unique logic into a separate class? One way to achieve this is inheritance.
- It means that you create a (child) class by deriving from another (parent) class. This way, we form a hierarchy.
- The child class reuses all fields and methods of the parent class (common part) and can implement its own (unique part).



Theory – OOP Principles Inheritance

- Create a class named Inheritance, this extending the Encapsulation one
- Implement all the listed methods
- Apart of the already in place method create a new one: public, return a String and the string returned should be “my_string”

```java
package oop;

public class Inheritance extends Encapsulation {
    public String getName() {
        return "overriden name";
    }

    public void encapsulationUsage() {
        this.concatIdAndName("Loczi", 5);
    }

    void tryDefault() {
        System.out.println("inherited");
    }

    public void newMethod() {
        System.out.println("a new method was created");
    }
}
```

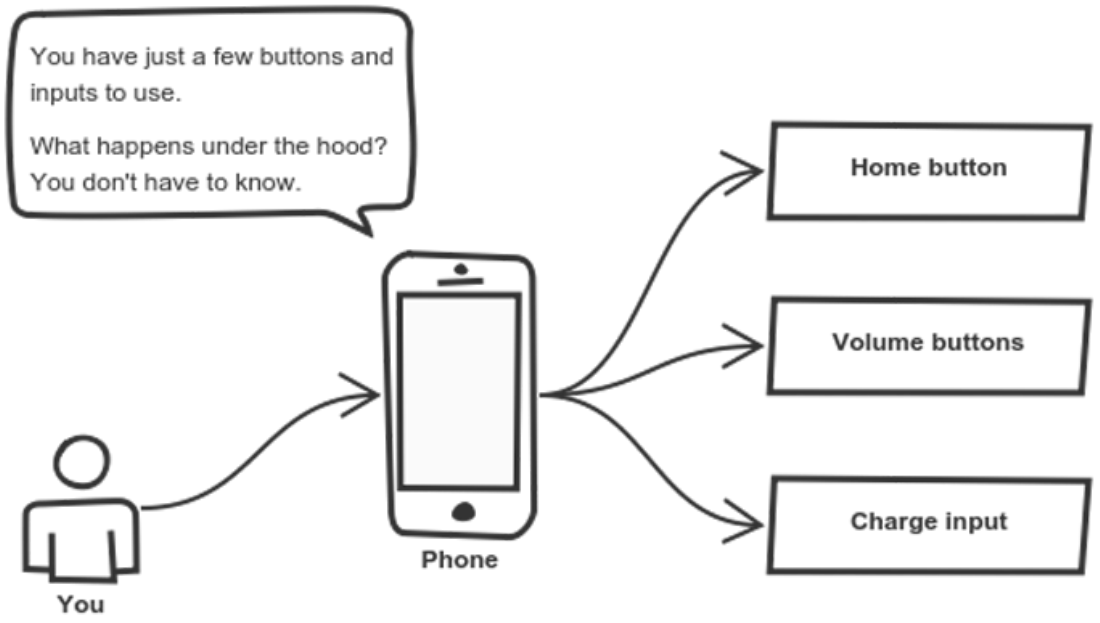
Theory – OOP Principles Inheritance

- Next step: use those methods
- In both Main classes use the code snippet from the image

```
Inheritance inh = new Inheritance();
inh.getName();
inh.encapsulationUsage();
inh.tryDefault();
inh.newMethod();
```

Theory – OOP Principles
Abstraction

- Problem: In object-oriented design, programs are often extremely large. And separate objects communicate with each other a lot. So maintaining a large codebase like this for years — with changes along the way — is difficult.
- Solution: Abstraction is a concept aiming to ease this problem. Applying abstraction means that each object should **only** expose a high-level mechanism for using it.
- Details: This mechanism should hide internal implementation details. It should only reveal operations relevant for the other objects.
- Example: Cell Phone



Theory – OOP Principles
Abstraction

- Create a new abstract class Abstraction
- Create another one extending the Abstraction one e.g. MyModel
- Override the name method from the Abstraction one

```java
// Create an Abstraction class
package oop;

abstract class Abstraction {
    public abstract void modelType();

    public static void name() {
        System.out.println("Abstract class");
    }
}


// Create the class which implement the abstract methods
package oop;

public class MyModel extends Abstraction {
    public void modelType() {
        System.out.println("Abstract method implemented");
    }
}
```

Theory – OOP Principles

Main class

Refactor the Main class from the src

Define a new String variable named principle initialized with e.g. “inheritance”

Implement a switch statement checking the state of the “principle” variable

If the principle is “inheritance”

Create an object based on the Inheritance class

Call a method by your own which print something

If the principle is “abstraction”

Create an object based on the MyModel class

Call a method by your own which print something

If the principle is “encapsulation”

Create an object based on the Encapsulation class

Call doThis method

By default

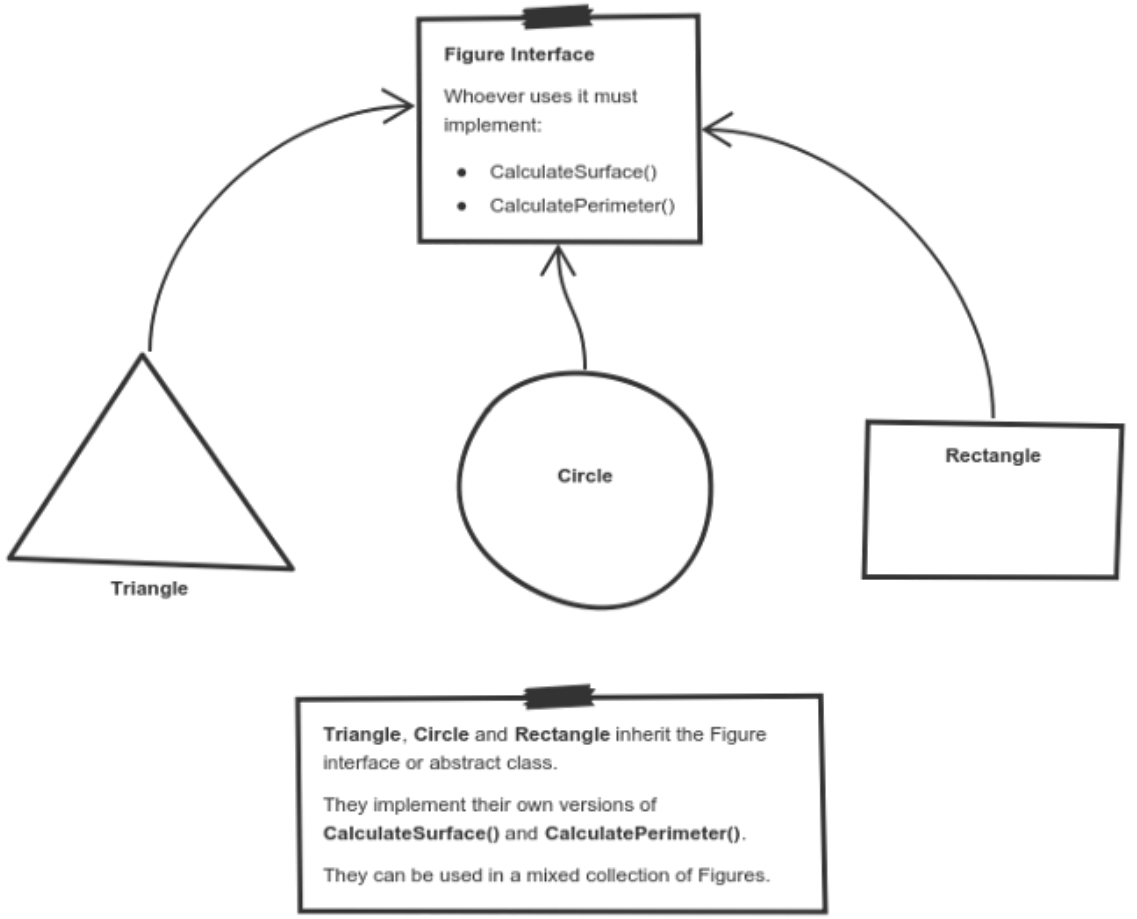
Just print a message

Theory – OOP Principles Main class - solution

```java
import oop.Encapsulation;
import oop.Inheritance;
import oop.MyModel;

public class Main {
    public static void main(String[] args) {
        String principe = "abstraction";
        switch (principe) {
            case "inheritance":
                Inheritance inh = new Inheritance();
                inh.getName();
                inh.encapsulationUsage();
                inh.newMethod();
                break;
            case "encapsulation":
                Encapsulation en = new Encapsulation();
                en.setName("Paul");
                String name = en.getName();
                System.out.println(name);
                break;
            case "abstraction":
                MyModel model = new MyModel();
                model.modelType();
                model.name();
                break;
            default:
                System.out.println("not available");
        }
    }
}
```

Theory – OOP Principles
Polymorphism

- Polymorphism means “many shapes” in Greek
- Inheritance based
- Image a parent class with multiple children
- Polymorphism gives a way to use a class exactly like its parent so there’s no confusion with mixing types. But each child class keeps its own methods as they are.

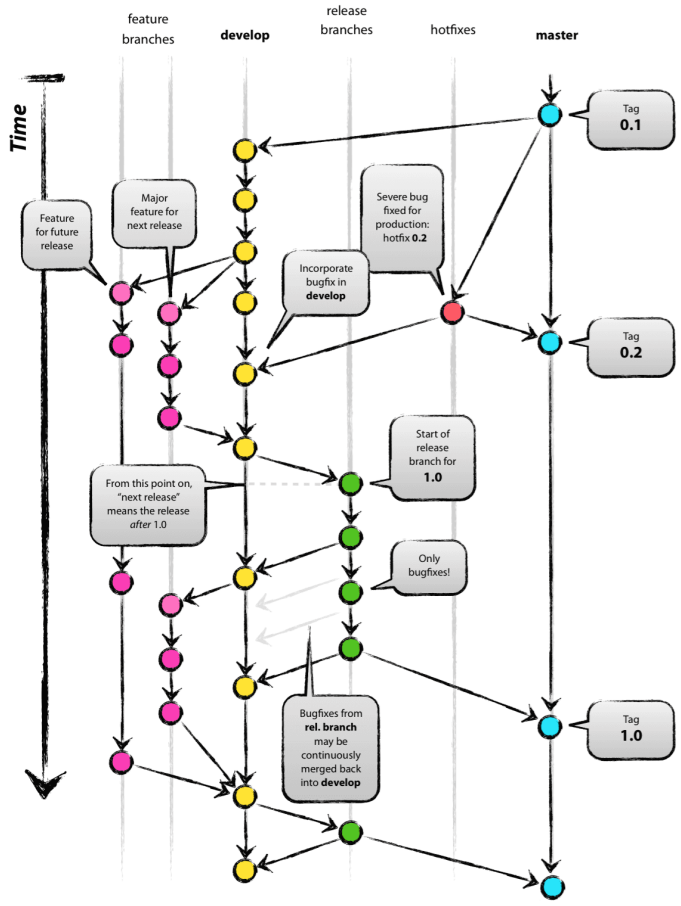


Theory – OOP Principles
Polymorphism

- Create a new java class Polymorphism
- Define all the listed classes from this snippet
- In the end play around with those classes inside the Polymorhism

```java
package oop;

class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}

class Polymophism {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();  // Create a Animal object
        Animal myPig = new Pig();  // Create a Pig object
        Animal myDog = new Dog();  // Create a Dog object
        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
}
```

Practice – Local setup
Git basics



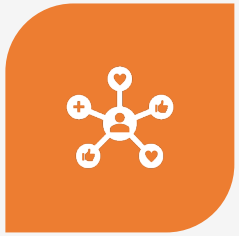
Git

- Manage software development projects and its files, as they are changing over time. Git stores this information in a data structure called a repository.
- Such a git repository contains a set of commit objects and a set of references to commit objects.
- A git repository is a central place where developers store, share, test and collaborate on web projects.

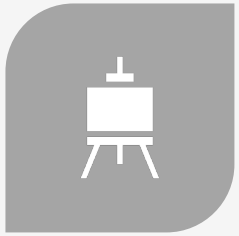
GitLab

- Is a repository manager which lets teams collaborate on code. Written in Ruby and Go, **GitLab** offers some features for issue tracking

Practice – Local setup
Git clone



IF YOU DON'T HAVE AN ACCOUNT, CREATE A NEW ONE



EACH GROUP TO CREATE A NEW PROJECT BASED ON THE PRODUCT NAME

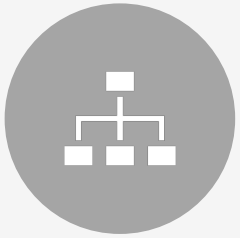
MAKE IT PUBLIC AND INCLUDE A README

DOWNLOAD GITHUB DESTKOP

CLONE OR ADD LOCAL REPOSITORY - > FROM URL OR FROM LOCAL PATH

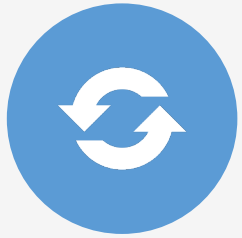
Commit your local changes

OPEN GITHUB DESKTOP APP



REVIEW IF LOCAL CHANGES ARE REFLECTED INTO YOUR STRUCTURE

CREATE A NEW BRANCH NAMED DEVELOP_YOUR_NAME



COMMIT CHANGES TO THE REPOSITORY

PUSH REMOTE BRANCH

Practice – Local setup Maven

PROJECT OBJECT MODEL - CONTAINS INFORMATION OF PROJECT AND CONFIGURATION INFORMATION FOR THE MAVEN TO BUILD THE PROJECT SUCH AS DEPENDENCIES, BUILD DIRECTORY, SOURCE DIRECTORY, TEST SOURCE DIRECTORY, PLUGIN, GOALS ETC.

PROJECT - IT IS THE ROOT ELEMENT OF POM.XML FILE.

MODELVERSION - IT IS THE SUB ELEMENT OF PROJECT. IT SPECIFIES THE MODELVERSION. IT SHOULD BE SET TO 4.0.0.

GROUPID - IT IS THE SUB ELEMENT OF PROJECT. IT SPECIFIES THE ID FOR THE PROJECT GROUP.

ARTIFACTID - IT IS THE SUB ELEMENT OF PROJECT. IT SPECIFIES THE ID FOR THE ARTIFACT (PROJECT). AN ARTIFACT IS SOMETHING THAT IS EITHER PRODUCED OR USED BY A PROJECT. EXAMPLES OF ARTIFACTS PRODUCED BY MAVEN FOR A PROJECT INCLUDE: JARS, SOURCE AND BINARY DISTRIBUTIONS, AND WARS.

VERSION - IT IS THE SUB ELEMENT OF PROJECT. IT SPECIFIES THE VERSION OF THE ARTIFACT UNDER GIVEN GROUP.

Enough for today – Maven
If it works, we are ready, else try until the next session

Next step: Maven

Create a new Java project: Choose Maven

Select some generic names for the group and artifact ids

Create a Main class with a main method and print to the console ”I’m ready for the next chapter”

Practice – Local setup
Install IntelliJ IDE

GroupId - uniquely identifies your project across all projects. A group ID should follow Java's package name rules. This means it starts with a reversed domain name you control. For example,org.apache.maven, org.apache.commons

ArtifactId - **artifactId** is the name of the jar without version. If you created it, then you can choose whatever name you want with lowercase letters and no strange symbols. If it's a third-party jar, you have to take the name of the jar as it's distributed. eg. maven, commons-math

Version - if you distribute it, then you can choose any typical version with numbers and dots (1.0, 1.1, 1.0.1, ...). Don't use dates as they are usually associated with SNAPSHOT (nightly) builds. If it's a third-party artifact, you have to use their version number whatever it is, and as strange as it can look. For example

More Practice – More Exercises 2nd part

Print this is “my second program developed by” + your_name

Print the console input. Tip Scanner (Google it)

Declare and print the following data types: bool, int, double, string, array

Write a program that perform following operations

- Add two number
- Extract a number from another
- Divide two numbers
- Multiply

Define an array of integers and loop through it using a loop

Write a while and a do while

Write an if statement that check if a number is even or odd

More Practice - More Exercise 3rd part

Create a new clean class with the main method. Inside of it get the input from the user using a scanner

Define four variables of different types.

Declare a string e.g. “myType”

Use an if statement to check the string stored by myType

Replace if statement with a switch

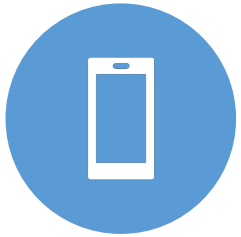
Define the following array: {1, 2, 3, 2, 1, 5, 4, 3, 7, 7, 4, 9, 1, 5}

Get all the elements unique from the array

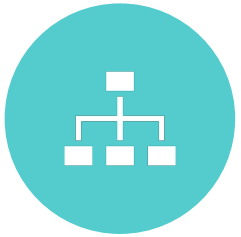
Remove the duplicates from the array

How many elements equal to 5 are in the array

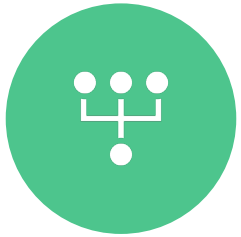
Commit your local changes



OPEN GITHUB DESKTOP APP



REVIEW IF LOCAL CHANGES ARE REFLECTED INTO YOUR STRUCTURE



CREATE A NEW BRANCH NAMED DEVELOP_YOUR_NAME

COMMIT CHANGES TO THE REPOSITORY

PUSH REMOTE BRANCH

Homework

- Implement a simple calculator application (sum (addition), multiply, subtract, divide, modulo)
- Swap two numbers using a temporary variable
- Swap two numbers without using a temporary variable
- Write a program that display a Xmas tree to the console
- Write a method that calculates the factorial of a given number.
  Factorial is the product of all positive integers less than or equal to n. For example, factorial(4) = 4x3x2x1 = 24.
- Write a Java program to calculate the area of a triangle
- Write a Java program that calculate the area of a polygon with 5 sides

Homework

- Write a program that converts km to miles and vice versa
- Write a program that converts grades Celsius grades to Fahrenheit
- Write a program that check if today is your b-day
- Write a program that will predict the first Xmas day on next year
- Write a Java program that takes a year from user and print whether that year is a leap year or not
- Find out how to create a new maven project and create a simple Hello world class
- Java program to check whether a string is a Palindrome

Homework

- Design a calculator using abstraction and inheritance. It should contain all the possible functionalities of a simple calculator (sum, diff, multiply, divide, modulo, equal)
- Using the polymorphism design multiple objects of a given product (a car, a toy, a fruit or some sweets)
- Swap two numbers using a temporary variable
- Swap two numbers without using a temporary variable
- Write a method that calculates the factorial of a given number.
  Factorial is the product of all positive integers less than or equal to n. For example, factorial(4) = 4x3x2x1 = 24.
- Write a method that returns the largest integer in the list.
  You can assume that the list has at least one element.
- Others: https://www.w3resource.com/java-exercises/

Thank you