WSTĘP DO PROGRAMOWANIA DLA TESTERÓW

„Everything I was I carry with me, everything I will be lies, waiting on the road ahead"
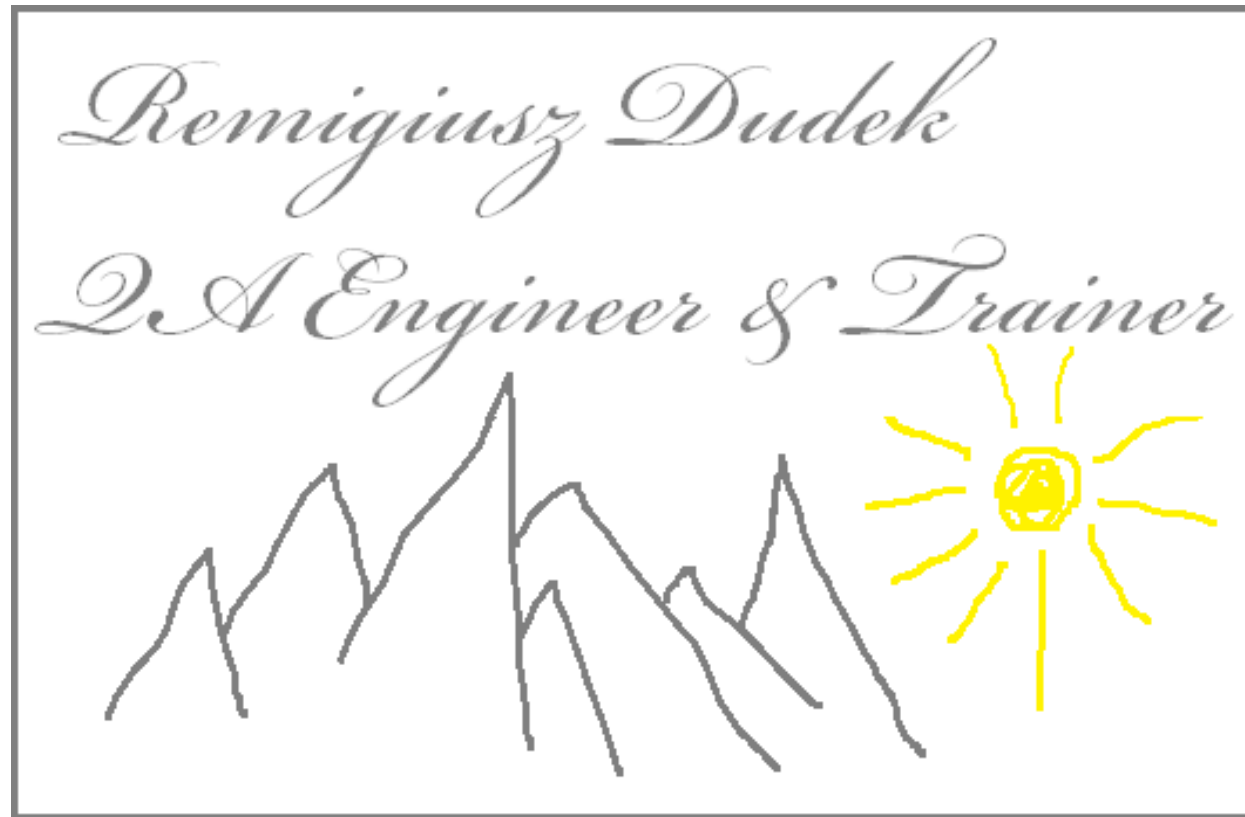
by Ma Jian

Wstęp do programowania Java dla testerów
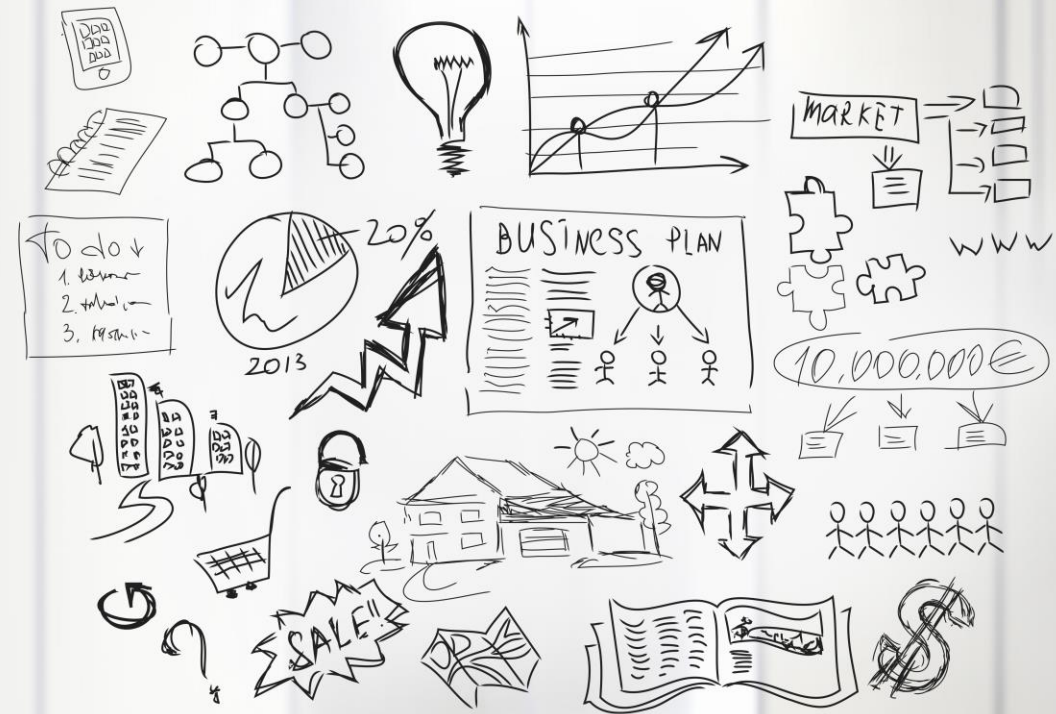
Remigiusz Dudek

GET TO KNOW

„LEARNING IS A JOURNEY NOT A DESTINATION" BY RALPH WALDO EMERSON (ALMOST)

AGENDA

- Day 1 (Basics + Java basics)
  - IDE – project structure
  - Class/Object/Package
  - First @Test
  - Basics (variables / methods)
  - Primitive types
  - Assetions
  - Basic Classes
  - Basic inheritance / Object creation
  - Equality
  - Strings
  - Arrays/Collection
  - Steer the flow (conditions/loops)

- Day 2 (OO Design)
  - Data driven testing (Parameters & File IO)
  - Inheritance
  - Polymorphism
  - Page Object Pattern
  - Data driven testing
  - Exceptions

- Day 3 (Advanced concepts)
  - F.I.R.S.T.
  - Clean code
  - S.O.L.I.D.
  - Design patterns
  - Test frameworks

JAVA BASICS

„A journey of a thousand miles, begins with a single step"
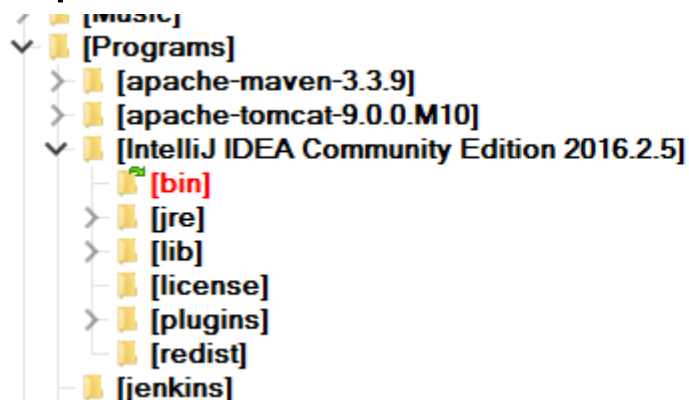
by Lau-Tzu

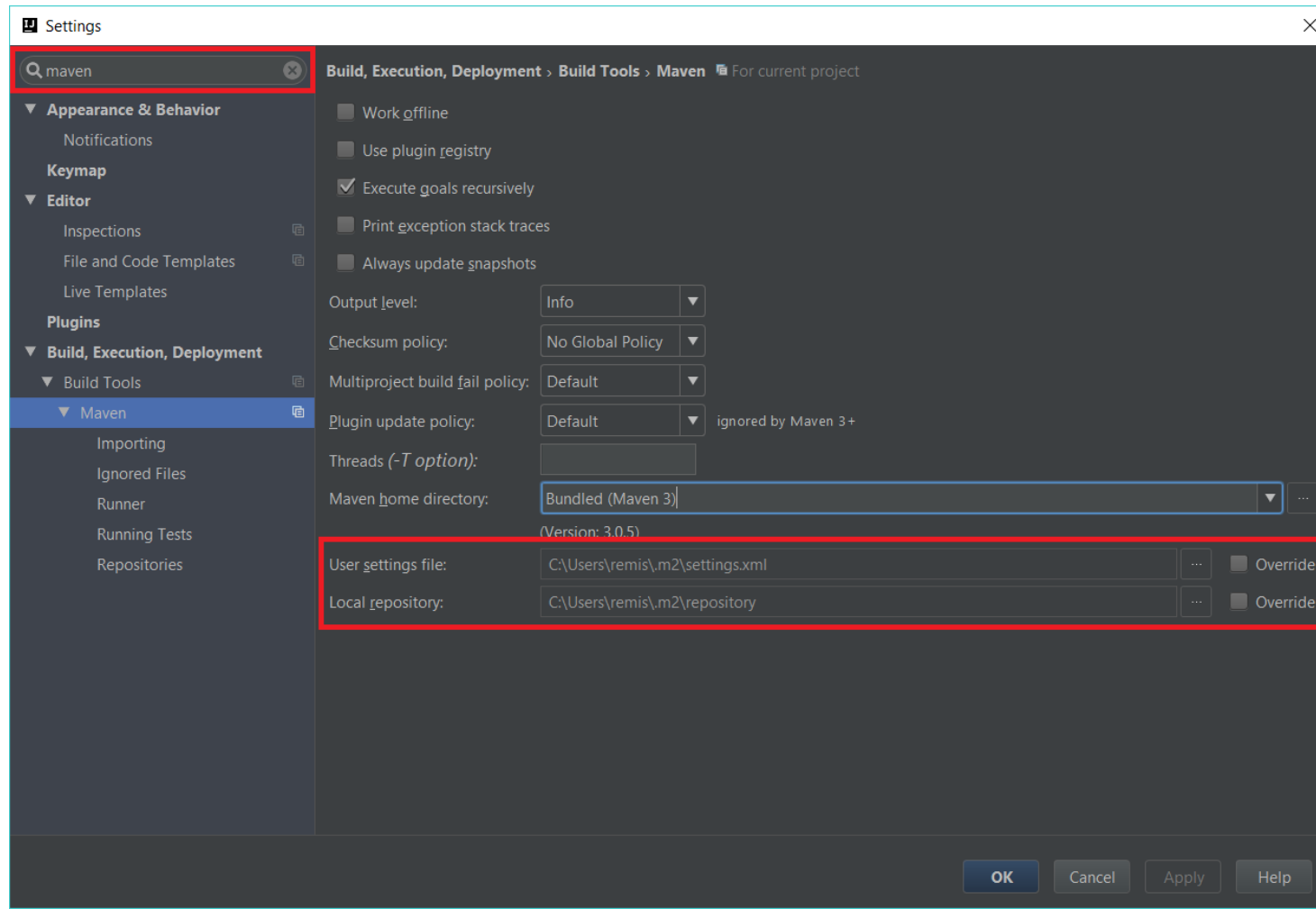Wstęp do programowania Java dla testerów

Remigiusz Dudek

SETUP INTELLIJ

- Check if you have java JDK installed `C:\Windows\System32>jconsole`

- Copy `Vistula-Programs.zip` and unzip it

- Run Intellij

SETUP MAVEN

PROJECT STRUCTURE

„THE KEY TO SUCCESSFULL LEARNING ENVIRONMENT IS STRUCTURE" BY CARA CAROLL
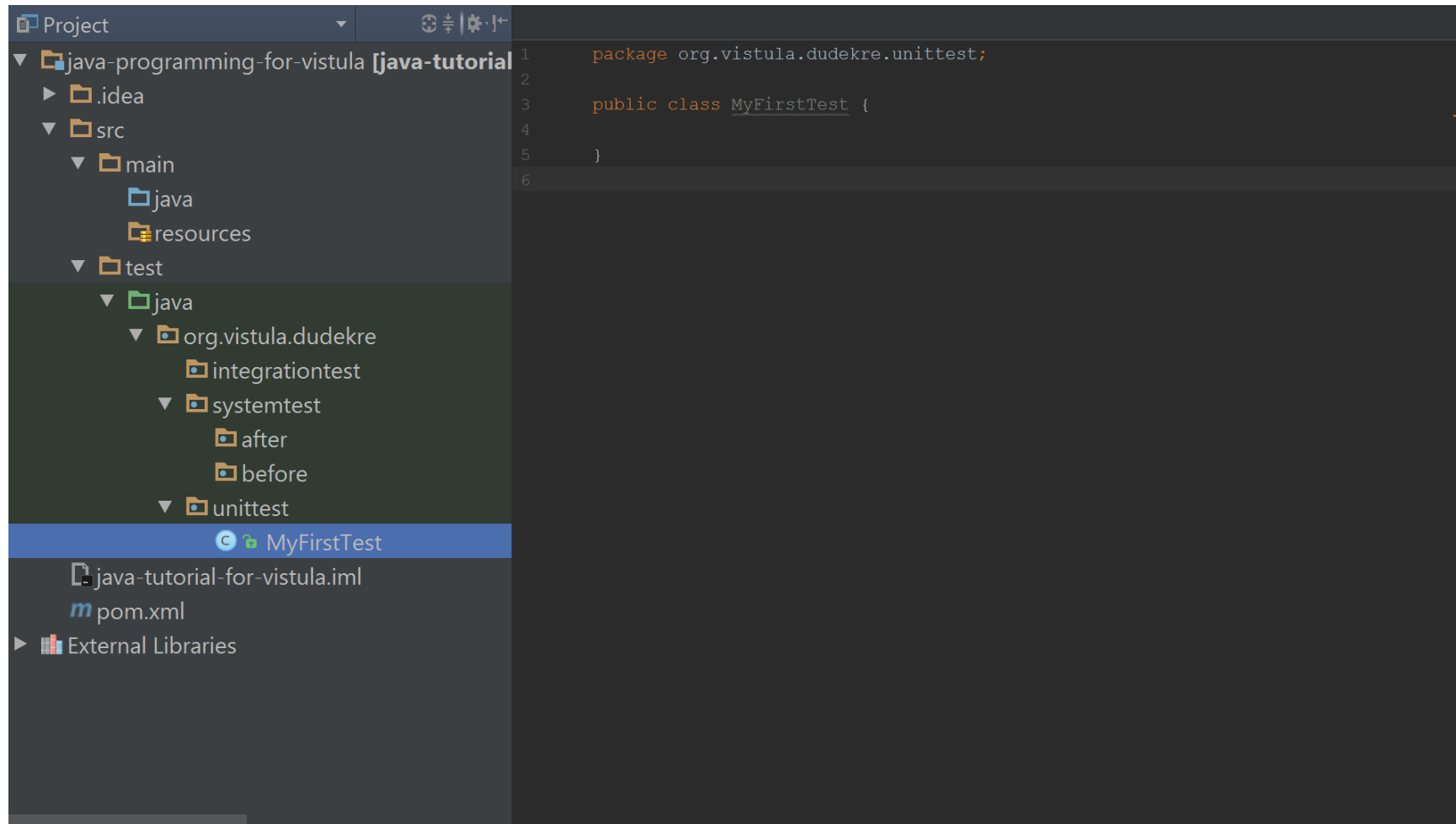
BIG PICTURE

1. What we need to build Java software?
   1. Something that would change text into runnable code – Java Software Development Kit (SDK)
   2. Something that would give as parts that we can use, so we do not need to create everything from scratch – Maven/Gradle
   3. Something that would build a deployable package – Maven/Gradle

2. How can we tell Maven/Gradle what are the parts that we need or how to build the package? – pom.xml/build.gradle

3. What do we need to run Java software? – Java Runtime Environment (JRE)
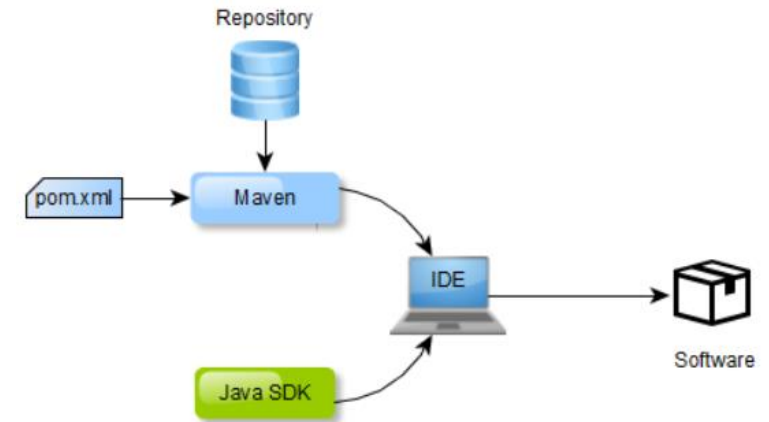
EXCERCISE (P)

1. Create your own project

2. Create following package structure (test source directory)
   a) `org.vistula.<your name>.systemtest.before`
   b) `org.vistula.<your name>.systemtest.after`

3. **Create class** `VistulaTest`

PACKAGE/CLASS/OBJECT
„IF YOU CORRECT YOUR MIND, THE REST OF YOUR LIFE WILL FALL INTO PLACE" BY LAO-TZU

- What is class?

- What is object?

- Naming conventions
  - Package
  - Class
  - Variables
  - Methods

```java
package org.vistula.dudekre.unittest;

import org.junit.Test;

public class MyFirstTest {
    @Test
    public void intentRevealingMethodName() {

    }
}
```

EXCERCISE (L)

1. Create @Test printing out „Hello <your name>"

2. Run the test in *Run Window*

3. Run the test in *Debug Window*

VARIABLES

- Why do we need them?

- Type

- Class/Method variable

```java
public class MyFirstTest {
    String messageSeparator = ",";

    @Test
    public void intentRevealingMethodName() {
        String message = "This is first part of a very important message";
        String otherMessage = "This is even more important";
        System.out.println(message + messageSeparator + otherMessage);
    }
}
```

EXCERCISE (P)

1. Print the same greeting as previously but this time assign your name to variable

2. Run the test in *Run Window*

3. Run the test in *Debug Window* *(do you see any difference?)*

METHODS

- Encapsulation

- Method signature
  - Parameters
  - Return value

```java
public class MyFirstTest {
    String messageSeparator = ",";

    @Test
    public void intentRevealingMethodName() {
        String message = "This is first part of a very important message";
        String otherMessage = "This is even more important";
        concatenateMessages(message, otherMessage);
    }


    private void concatenateMessages(String message, String otherMessage) {
        System.out.println(message + messageSeparator + otherMessage);
    }
}
```

EXCERCISE (L)

1. Extract greeting method taking your name as a parameter

2. Run the test in **Run Window**

3. Run the test in **Debug Window** *(do you see any difference?)*

NUMBER VARIABLES

- Number types (int, long, float, double)

- Boxing types

- Mathematical operators (=, +, -, *, /)
  - Dividing integers!
  - MAX_VALUE/MIN_VALUE

- Other operators (++, --, +=, -=)

- Basic assertions

```java
@Test
public void mathematicalOperations() {
    int integerNumber = 5;
    long largeIntegerNumber = 5L;
    float floatingPointNumber = 1.3f;
    double largeFloatingPointNumber = 1.3d;
    System.out.println(2 * integerNumber);
}
```

EXCERCISE (P)

1. Create a method that takes two parameters and:
    1. adds them (what is the result of adding different variables types)
    2. Play with MAX_VALUE, MIN_VALUE
    3. Divide them (divide different types)

2. Create a test that ensures that method works

3. Run the test in *Run Window*

4. Run the test in *Debug Window* *(do you see any difference?)*

CREATING OBJECTS

- Encapsulation

- Constructor (`new` keyword, `this` keyword)
  - Default
  - Non-default

- Private/Public access modifier
  - Default values of fields

- Class methods

- Basic inheritance - each object in Java is `Object`

```java
public class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int changeXCoordinate(int newX) {
        return x;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}
```

EXCERCISE (L)

1. Create `WebPage` class that has `title` attribute

2. The `title` attribute should be set at construction time

3. Assert that `title` is correct
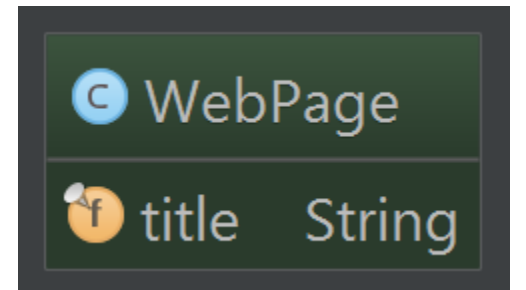
STEER THE FLOW

1. Conditionals
   a) If
   b) If / else
   c) If /else if / else

2. Operators
   a) ==, !=
   b) <, >
   c) <=, >=
   d) &&, ||

```java
@Test
public void twoPointsShouldBeEqual() {
    Point a = new Point(1,2);
    Point b = new Point(2,3);
    Point c = new Point(3,4);

    if (a.getX() == b.getX()) {
        System.out.println("a & b have equal x");
    }

    if (a.getX() != b.getX() && a.getY() == b.getY()) {
        System.out.println("a has different x and the same y
coordinate as point b");
    }

    if (a.equals(b)) {
        System.out.println("a equals to b");
    } else if (a.equals(c)) {
        System.out.println("a equals to c");
    } else {
        System.out.println("a is unique");
    }
}
```

EXCERCISE (P)

1. Create class `Person` that has attribute `age`

2. `Person` should be able to give an answer to a question whether it is working/non-working
   a) nonWorking (age below 18 and above 67)
   b) Working (age between 18 and 67)

```java
@Test
public void personIsWorkingIfAgeBetween18And67() {
}

@Test
public void personIsNotWorkingIfAgeBelow18() {
}

@Test
public void personIsNotWorkingIfAgeAbove67() {
}
```

EQUALITY

- Boolean : true / false

- operator == (what a reference is?)

- Equality
  - Any object can never be equal to null (**what is null !!!**)
  - Every object is always equal to itself
  - When objects are equal their hash is equal
  - Two objects with the same hash does not need to be equal

```java
public class Point {
    private int x;
    private int y;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null) return false;
        if (getClass() != o.getClass())
            return false;

        Point point = (Point) o;

        if (x != point.x) return false;
        return y == point.y;
    }

    @Override
    public int hashCode() {
        int result = x;
        result = 31 * result + y;
        return result;
    }
}
```

EXCERCISE (P)

1. Make two `WebPages` **equal when their** `titles` **are equal**
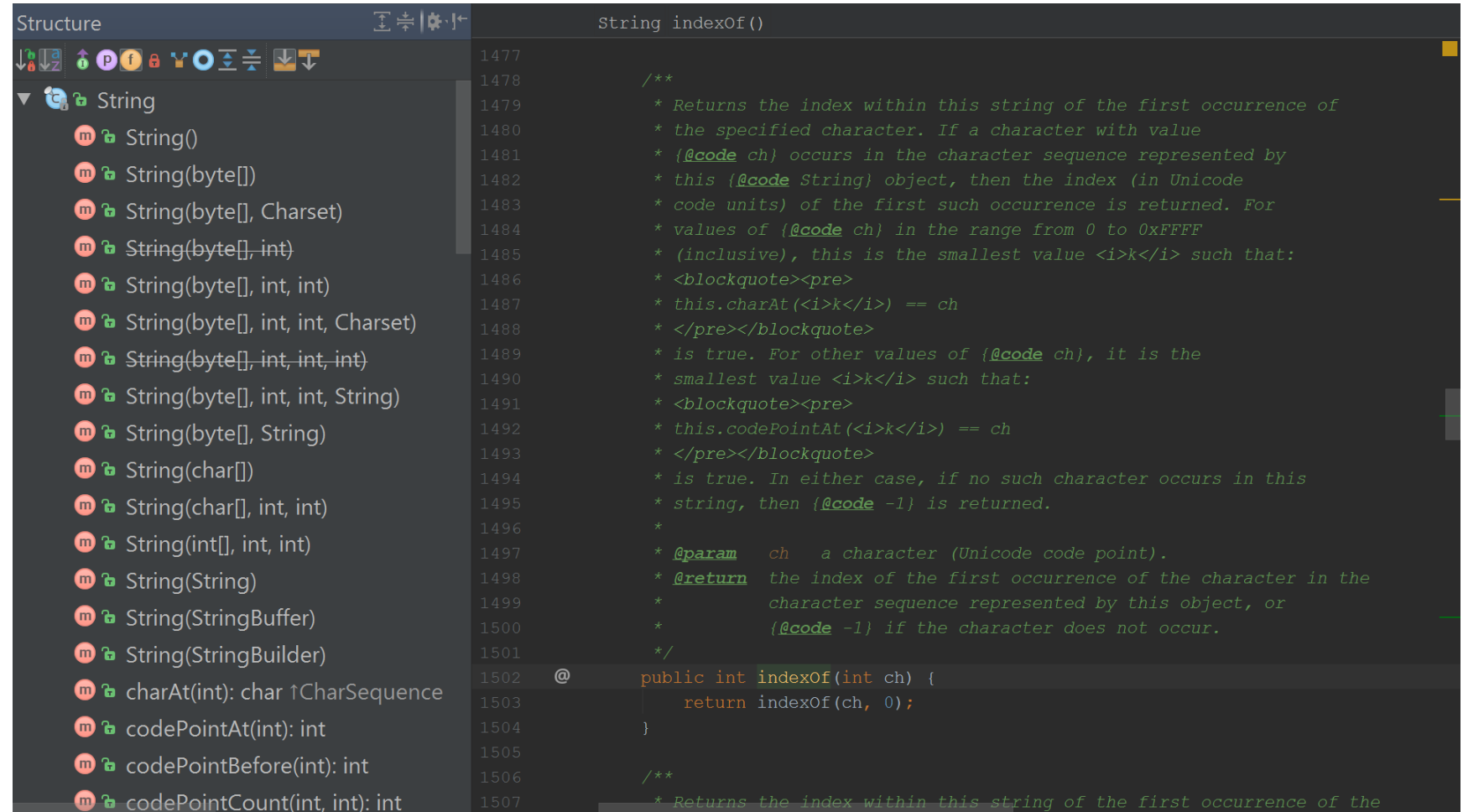
2. Play with == operator
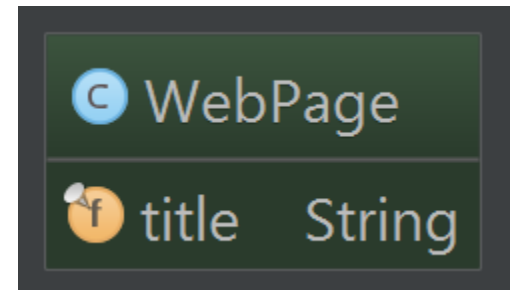
STRING VARIABLES

1. String API

2. Convertions
   a) `toString()`
   b) From string

EXCERCISE (L)

1. Create a method that is able to extract numer of animals from text

2. Create a test for the method

```
@Test
public void shouldExtractNumber() {
    String text = "There are 4 animals in the room";
    int numberOfAnimals = extractNumberOfAnimals(text);
    //assert that numberOfAnimals == 4
}

private int extractNumberOfAnimals(String text) {

    return 0;
}
```

EXCERCISE (P)

1. Print `WebPage` nicely [WebPage {title: <title>}]

2. Replace $tradeId to some id in string `"tradeId: ${tradeId}"`
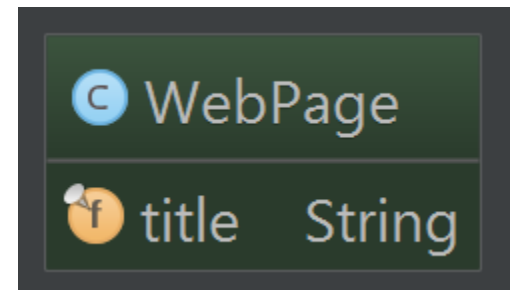
JAVA COLLECTIONS – ARRAY LIST

1. Collections
   a) ArrayList
   b) HashSet
   c) HashMap

2. Passing variables trough copy of reference

EXCERCISE (L)

1. Create ArrayList of WebPages

2. Assert that list contains added WebPages

3. Print this list out
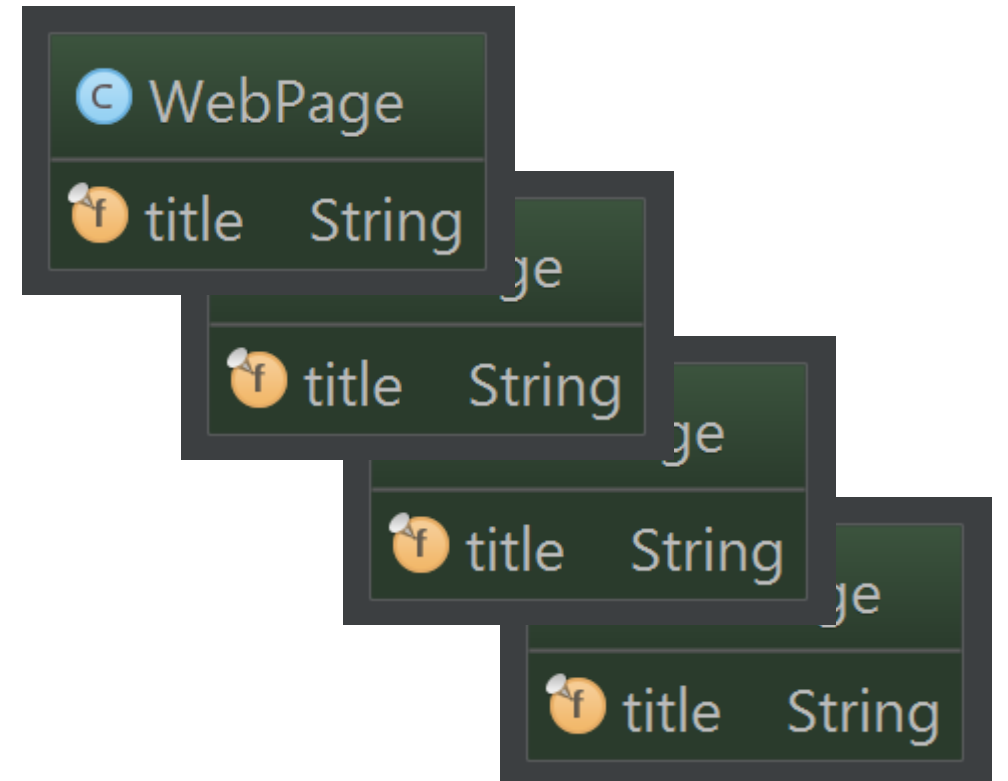
JAVA COLLECTIONS – HASH SET

1. Collections
   a) ArrayList
   b) HashSet
   c) HashMap

2. Passing variables trough copy of reference
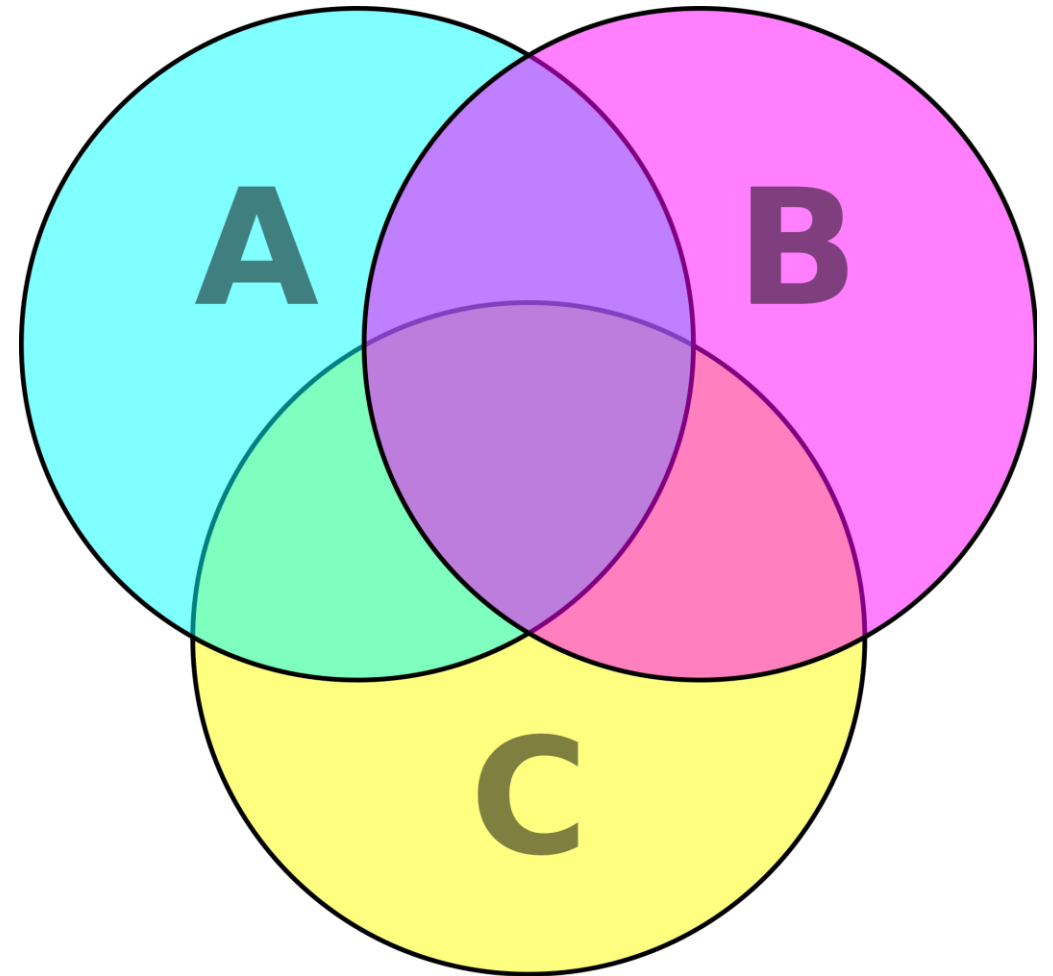
EXCERCISE (P)

1. Create HashSet of WebPages – try to add two equal WebPages
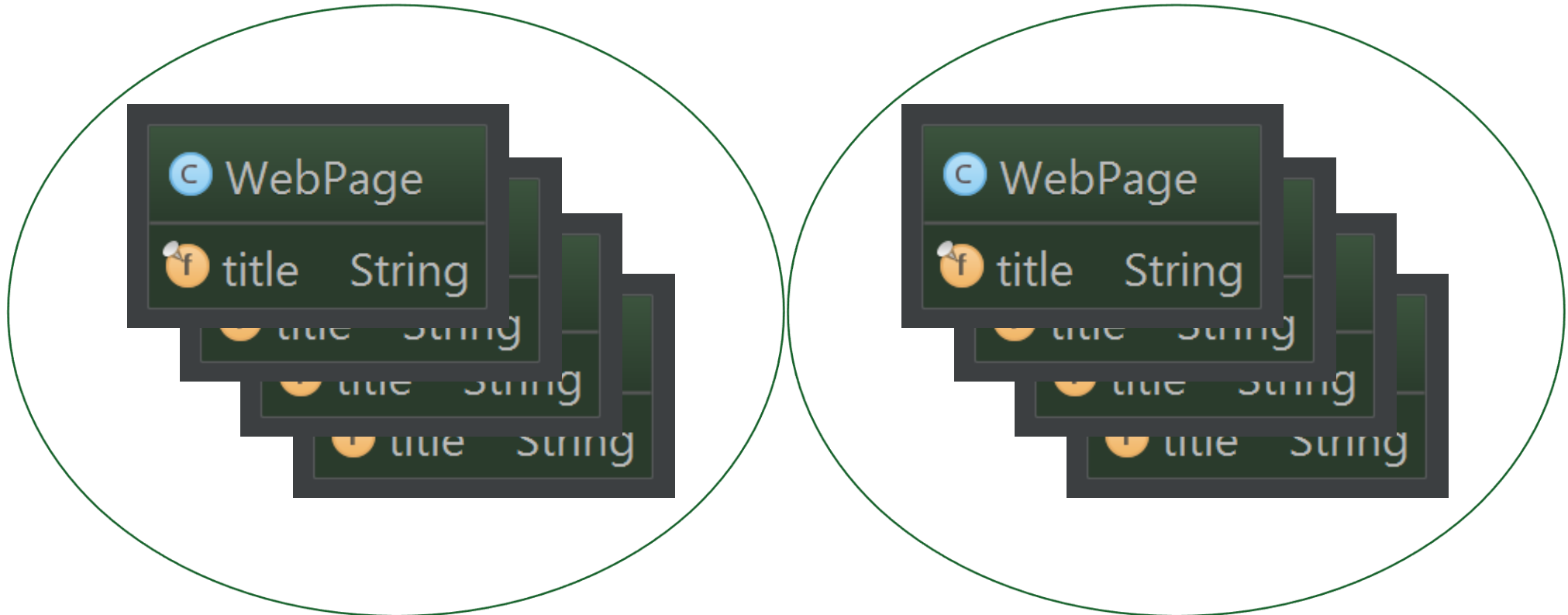
JAVA COLLECTIONS – HASH MAP

1. Collections
   a) ArrayList
   b) HashSet
   c) HashMap

2. Passing variables trough copy of reference
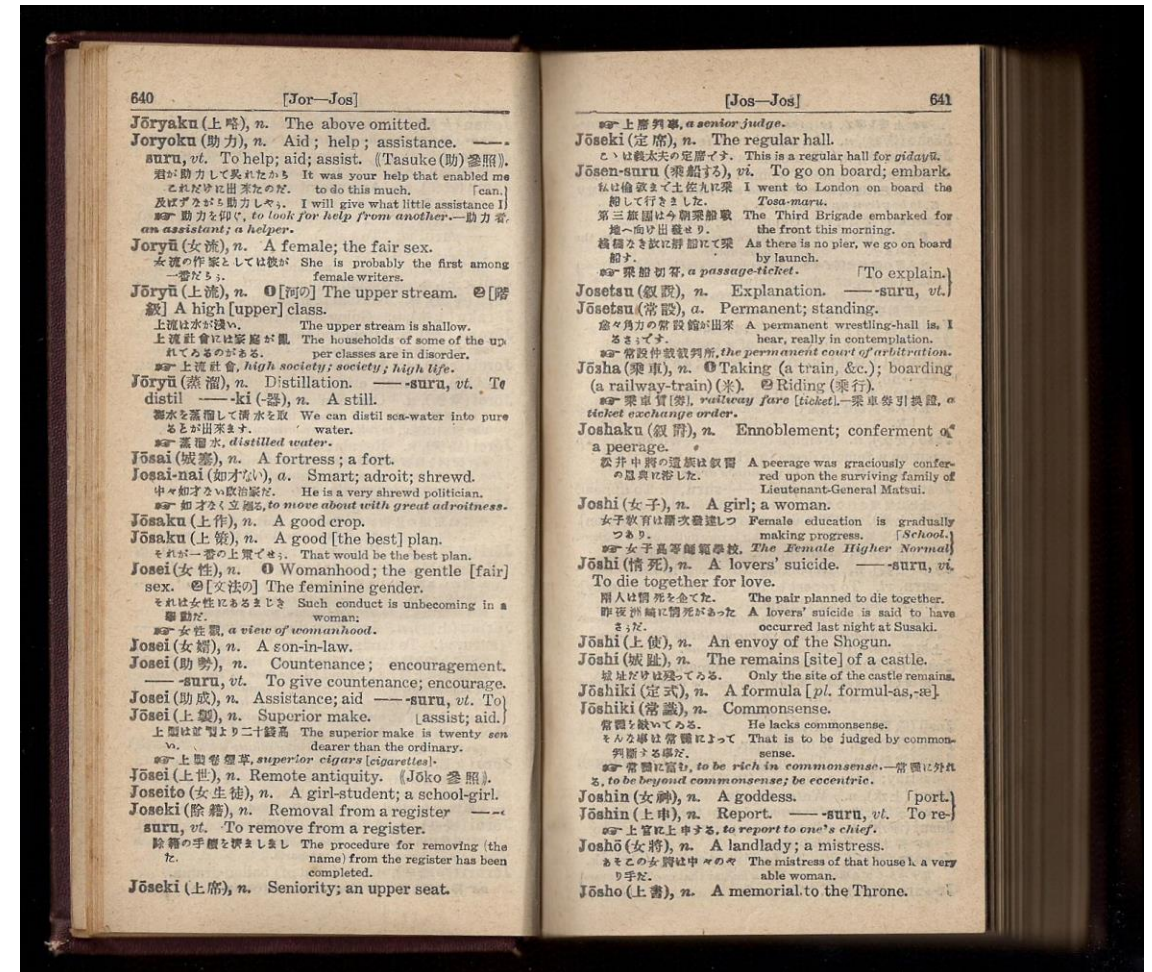
EXCERCISE (L)

1. Create and test CredentialsValidator using HashMap

STEER THE FLOW - LOOPS

1. Loops
   a) For
   b) While

```java
@Test
public void forLoopOverCollection() {
    List<Integer> grades = new ArrayList<Integer>();
    for (Integer grade : grades) {
        System.out.println(grade);
    }
}
@Test
public void forLoopClasic() {
    for (int i = 0; i < 10; i++) {
        System.out.println(i);
    }
}
@Test
public void whileLoopClasic() {
    int idx = 0;
    while (idx < 10) {
        System.out.println(i);
        idx++;
    }
}
```

EXCERCISE (P)

1. Create list of `Persons` with different `age (for)`

2. Divide this list into three lists (`for +if`)
   a) nonWorking (age below 18)
   b) Working (age between 18 and 67)
   c) Retired (age above 67)

3. Find first `Person` with `age` above 18 (`while`)

```java
@Test
public void workStatusDivision() {
    List<Person> people = new ArrayList<Person>();
    people.add(new Person(10));
    people.add(new Person(17));
    people.add(new Person(18));
    people.add(new Person(45));
    people.add(new Person(66));
    people.add(new Person(67));
    people.add(new Person(120));

    List<Person> juniors = getJuniors(people);
    List<Person> workinClass = getWorkingClass(people);
    List<Person> seniors = getSeniors(people);
}

private List<Person> getSeniors(List<Person> people) {
    return null;
}

private List<Person> getWorkingClass(List<Person> people) {
    return null;
}

private List<Person> getJuniors(List<Person> people) {
    return null;
}
```

HOMEWORK

1. Create a method that returns n'th element of a fibonacci series (1, 1, 2, 3, 5, 8, …)

2. Write a `Triangle` class which has method `field()` returning the field of an triangle

3. Write a `TriangleValidator` that checks if it is possible to create a triangle using three sides of given length

```java
public class TriangleValidator {
    public boolean validate(int a, int b, int c) {
        // a + b > c
        // a + c > b
        // b + c > a
        // return true if all inequalities are met
        // return false if at least one inequality is not met
    }
}
```

35

HOMEWORK

4. Create a program that would keep titles of webpages and amount of visits each webpage received

```java
public class WebPageVisitCounter {
    private HashMap<String, Integer> visits = new HashMap<String, Integer>();

    public void visit(String title) {

    }

    public int getNumberOfVisits(String title) {
        return 0;
    }
}
```

5. Write a program that checks if given word is a palindrome (ex. Kajak, Ala)

HOMEWORK

6. Create a `Rectangle` class that is able to calculate field and perimeter

7. Create a `PercentGrader` class that translates percentage to a grade
   a) 95% - 100% - 6
   b) 85% - 95% - 5
   c) 75% - 85% - 4
   d) 65% - 75% - 3
   e) 55% - 65% - 2
   f) 0% - 55% - 1

8. Write a program that draws multiplication table – no test required

X	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
2	0	2	4	6	8	10	12	14	16	18	20
3	0	3	6	9	12	15	18	21	24	27	30
4	0	4	8	12	16	20	24	28	32	36	40
5	0	5	10	15	20	25	30	35	40	45	50
6	0	6	12	18	24	30	36	42	48	54	60
7	0	7	14	21	28	35	42	49	56	63	70
8	0	8	16	24	32	40	48	56	64	72	80
9	0	9	18	27	36	45	54	63	72	81	90
10	0	10	20	30	40	50	60	70	80	90	100