

# 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

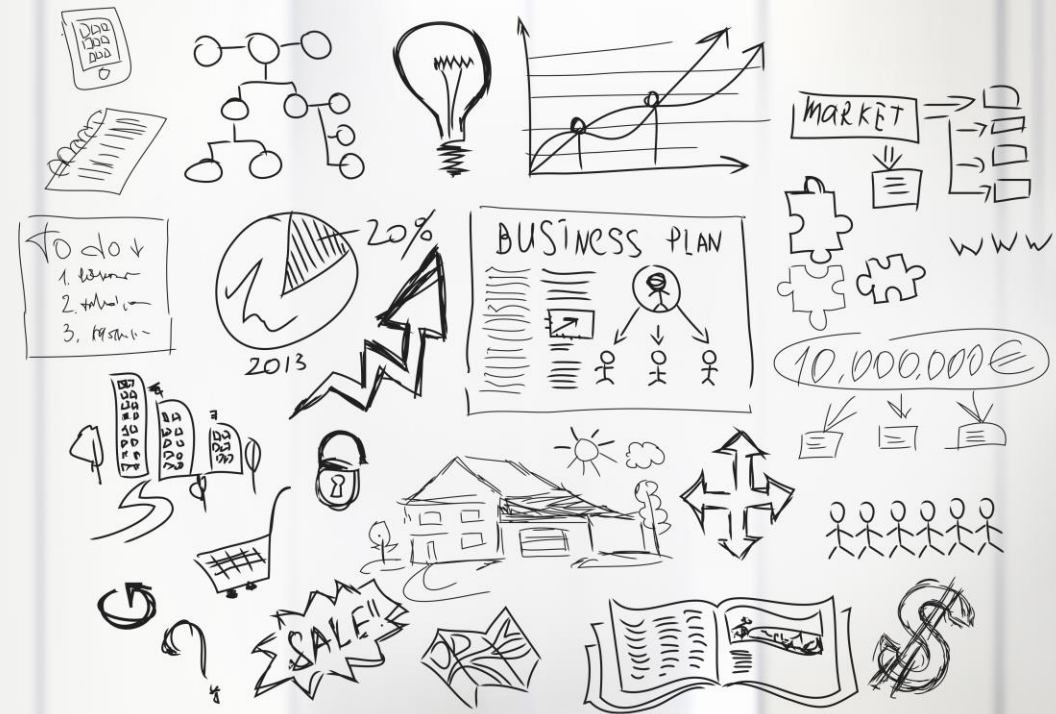
Wstęp do programowania Java dla testerów

Remigiusz Dudek

# AGENDA

*„THE FIRST STEP TOWARDS GETTING SOMEWHERE IS TO DECIDE THAT YOU ARE NOT GOING TO STAY WHERE YOU ARE“* BY ANNONYMOUS

- Basics + Java basics
  - IDE – project structure
  - Class/Object/Package
  - First @Test
  - Basics (variables / methods)
  - Primitive types
  - Assertions
  - Basic Classes
  - Basic inheritance / Object creation
  - Equality
  - Strings
  - Arrays/Collection
  - Steer the flow (conditions/loops)
- OO Design
  - Data driven testing (Parameters & File IO)
  - Design pattern - Singleton
  - Exceptions
  - Inheritance
  - Page Object Pattern
  - *Polymorphism*



# DATA DRIVEN TESTING

„A journey of a thousand miles, begins with a single step”

by Lau-Tzu

Wstęp do programowania Java dla testerów

Remigiusz Dudek

# JUNIT PARAMS

```
<repositories>
  <repository>
    <id>CentralMavenRepository</id>
    <url>http://central.maven.org/maven2/</url>
  </repository>
</repositories>
```

1. How to run the same test with different input parameters?

```
<dependencies>
  (. . .)
  <dependency>
    <groupId>pl.pragmatists</groupId>
    <artifactId>JUnitParams</artifactId>
    <version>1.0.5</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

# JUNIT PARAMS

```
@RunWith(JUnitParamsRunner.class)
public class ParameterizedTests {
    @Test
    @Parameters({
        "1, 2, 3",
        "3, 1, 2",
        "2, 3, 1"})
    public void shouldNotBePossibleToCreateTriangle(int a, int b, int c) {
        TriangleValidator triangleValidator = new TriangleValidator();

        boolean isValid = triangleValidator.validate(a, b, c);

        Assertions.assertThat(isValid).isFalse();
    }

    @Test
    @FileParameters("src/test/resources/org/vistula/dudekre/credentials.csv")
    public void readParametersFromFile(String username, String password) {
        System.out.println(username + "," + password);
    }
}
```

# EXERCISE

1. Redo any of tests already created with usage of JUnitParams (ex. PalindromeChecker)

# DESIGN PATTERN — SINGLETON

```
public class Singleton {  
    private static Singleton instance;  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        if (instance == null) {  
            instance = new Singleton();  
        }  
        return instance;  
    }  
}
```

# EXERCISE

## 1. Make CredentialsValidator to be a Singleton

```
public class CredentialsValidator {  
    boolean validate(String username, String password) {  
        . . .  
    }  
  
    public void addValidCredentials(String username, String password) {  
        . . .  
    }  
}
```



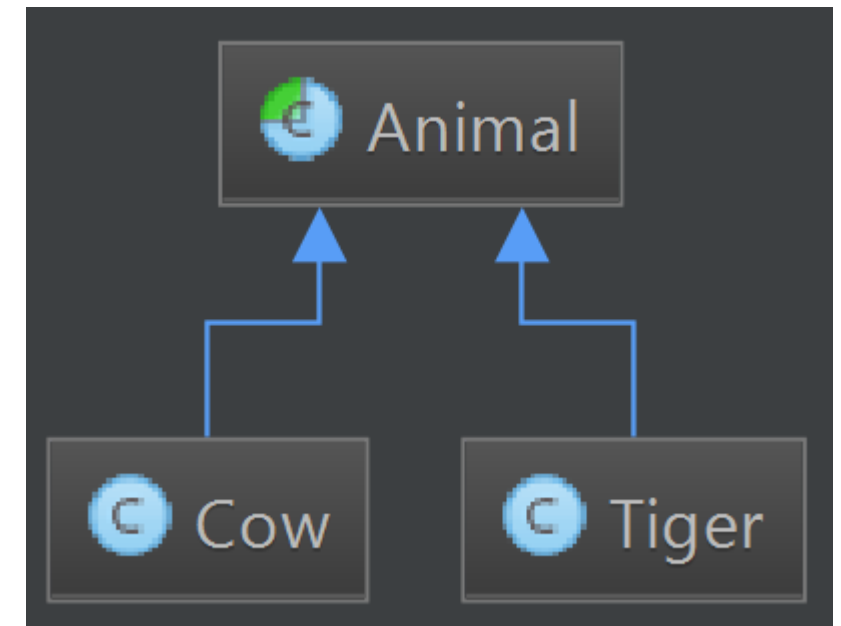
# INHERITANCE/IMPLEMENTATION

## 1. Inheritance

- a) „Is something” relation
  - a) Cow is an Animal
  - b) Tiger is an Animal
- b) **Extends** the base class
- c) Inherit (have access to) all public method
- d) **protected** fields/method visible for children
- e) **abstract** class cannot be instantiated (all abstract methods needs to be implemented)

## 2. Implementation

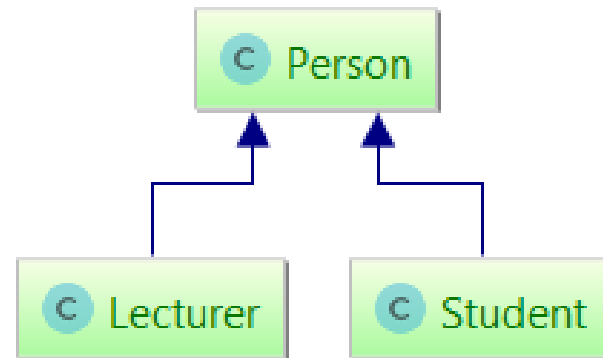
- a) „Behaves like something” relations
  - a) Lamp is Switchable
  - b) Washing machine is Switchable
- b) Interface is a set of methods that needs to be implemented
- c) All methods defined in an interface are public



# INHERITANCE

```
public class Person {  
    private String name;  
  
    public Person(String name) {  
        this.name = name;  
    }  
  
    public void marry(Person p) {  
        this.name = p.name;  
    }  
  
    public String name() {  
        return name;  
    }  
}
```

```
public class Lecturer extends Person {  
    private Date hiringDate;  
  
    public Lecturer(String name, Date hiringDate) {  
        super(name);  
    }  
  
    public NoticePeriod noticePeriod() {  
        return THREE_MONTHS;  
    }  
}
```



# EXERCISE

1. Create AdministratorPage, HomePage, LoginPage (all are WebPages)

```
public class Person {  
    private String name;  
  
    public Person(String name) {  
        this.name = name;  
    }  
  
    public void marry(Person p) {  
        this.name = p.name;  
    }  
  
    public String name() {  
        return name;  
    }  
}
```

```
public class Lecturer extends Person {  
    private Date hiringDate;  
  
    public Lecturer(String name, Date hiringDate) {  
        super(name);  
    }  
  
    public NoticePeriod noticePeriod() {  
        return THREE_MONTHS;  
    }  
}
```

# EXCEPTIONS

1. Used to handle exceptional situations
2. Throw keyword
3. All Exceptions should extend Exception class
  1. Message
  2. Caused by
  3. Stack trace
4. Try/Catch
  1. RuntimeException
  2. Exception

```
public class InvalidLoginException extends RuntimeException {  
    public InvalidLoginException(String message) {  
        super(message);  
    }  
  
    public InvalidLoginException(Throwable cause) {  
        super(cause);  
    }  
}
```

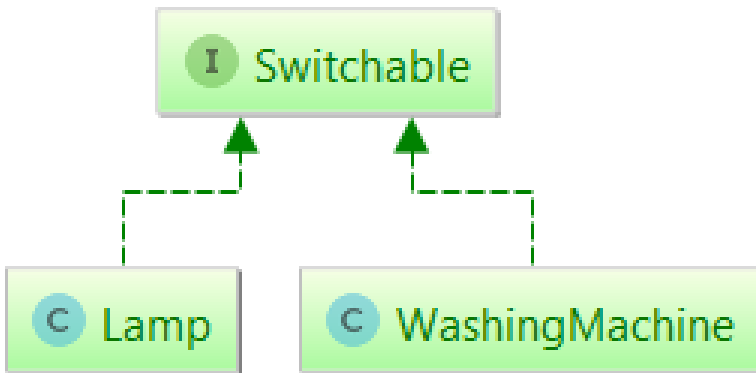
# EXERCISE

1. If TriangleValidator gets invalid input (e.g. negative number) it should throw exception
2. Credentials validator should throw exception if credentials are invalid

# INTERFACES

## 1. Implementation

- a) „Behaves like something” relations
  - a) Lamp is Switchable
  - b) Washing machine is Switchable
- b) Interface is a set of methods that needs to be implemented
- c) All methods defined in an interface are public



```
public interface Switchable {
    void turnOn();
    void turnOff();
}
```

```
public class Lamp implements Switchable {
    public void turnOn() {
        System.out.println("Here is the light");
    }

    public void turnOff() {
        System.out.println("Lamp is turned off");
    }
}
```

```
public class WashingMachine implements Switchable {
    public void turnOn() {
        System.out.println("Started washing");
    }

    public void turnOff() {
        System.out.println("Washing machine is turned off");
    }
}
```

# EXERCISE

1. `CredentialsValidator` **behaves like** `CredentialsStore` **and** *`CredentialsValidator`* (sic!)

# PAGE OBJECT PATTERN

```
public class EmailLoginPage extends WebPage {
    public EmailHomePage validLogin(String email, String password) {
        typeInEmailAndPassword(email, password);
        pressLoginButton();
        return new EmailHomePage();
    }

    public InvalidLoginPage invalidLogin() {
        typeInEmailAndPassword("invalid@invalid.com", "invalid");
        pressLoginButton();
        return new InvalidLoginPage();
    }

    public PasswordRetrievalPage retrievePassword() {
        return new PasswordRetrievalPage();
    }

    public void checkRememberMe() {
        checkRememberMeTickBox();
    }
}
```

Logowanie [? Pomoc](#)

adres e-mail

hasło

**ZALOGUJ SIĘ**

☐ Zapamiętaj mnie [Odzyskaj hasło »](#)



# HOMework

## 1. Basing on Page Object Pattern create framework for testing web pages

### a) Characteristics of an application (blog)

- i. We always start from LoginPage (Administrator account is by default available)
- ii. After logging as administrator, AdministrationPanel is opened and user is able to add another user (username + password)
- iii. Only added users can log to an application (if invalid credentials are provided, Invalid Login Page is displayed)
- iv. Once user is properly logged in, HomePage for given user is displayed
- v. Being on a home page user can add new Article (Article = Title + text)
- vi. Being on a home page user can remove existing Articles
- vii. Being on a home page user can read all articles that has been added

### b) Test scenarios

- i. After logging with Invalid credentials InvalidLoginPage is displayed
- ii. Only added users can log in
- iii. It should be possible to add Article
- iv. It should be possible to remove Article

# POLYMORPHISM

