



## Wydział Telekomunikacji, Informatyki i Elektrotechniki

Laboratorium z przedmiotu: Zaawansowane Programowanie Obiektowe

Imię i Nazwisko:	Paweł Pauszek	Ćwiczenie nr:	5
Temat Ćwiczenia:	Walidacja z wykorzystaniem własnej adnotacji		
Kierunek:	Informatyka Stosowana	Grupa:	2
Data wykonania ćwiczenia:	16.05.2022	Data oddania Sprawozdania:	30.05.2022

Napisać aplikację okienkową (z wykorzystaniem biblioteki Swing lub JavaFX), która umożliwi walidację wybranego pola dowolnej klasy w konwencji JavaBean, oznaczonego własną adnotacją. Zadania do wykonania i wymagania:

Utworzyć nową adnotację do walidacji oznaczonych pól klasowych. Adnotacja powinna bazować na wyrażeniach regularnych, które posłużą do sprawdzania poprawności wprowadzanych danych.

- Dla dowolnej klasy utworzonej w konwencji JavaBean wybrać i oznaczyć za pomocą stworzonej adnotacji pole, które podlegać będzie walidacji, np.  
`@MyPattern(regex="correct value", message = "This value is not correct!")`  
`String myValidatedField=""`;

- Stworzyć dwa proste obrazki, np. „0.png” oraz „1.png” o rozdzielczości 20x20 pikseli, które reprezentować będą wynik walidacji (1, 3).

- Skorzystać z klasy HBox (JavaFX) lub BoxLayout (Swing) w celu utworzenia klasy o nazwie „VinputText”, wykorzystującej kontrolkę TextInputControl (JavaFX) lub komponent JTextComponent (Swing). Takie rozwiązanie pozwoli zastosować pole lub obszar tekstowy w zależności od potrzeby.

- Klasa VinputText powinna zawierać metodę registerValidator(Validator v), aby umożliwić zarejestrowanie obiektu walidatora. Po jego zarejestrowaniu, komponent VinputText będzie wykorzystywał ten obiekt do sprawdzania wprowadzanych danych (w przypadku zadania dodatkowego poniżej będzie to wiele walidatorów rejestrowanych i przechowywanych w stosownej kolekcji).

- Wykorzystać mechanizm refleksji w celu wykrycia i zastosowania stworzonej adnotacji.

- Utworzyć odpowiednią klasę walidatora. Przyjmijmy następującą konwencję: nazwa klasy walidatora tworzona jest przez dodanie do nazwy adnotacji słowa „Validator”, np. dla adnotacji @MyPattern klasa walidująca będzie nosić nazwę MyPatternValidator.

- Klasa walidatora powinna pozwalać na validację zgodnie z parametrami przekazanymi w adnotacji (np. wyrażenie regularne przekazane jako parametr regex). Natomiast parametr message ma umożliwiać przekazywanie wiadomości, która wyświetlana będzie jako Tooltip po najechaniu kursorem na x.

- Klasa walidatora powinna implementować interfejs Validator. Metoda validate(String value) na bazie parametrów adnotacji dokonuje validacji wartości wprowadzonej za pomocą komponentu VinputText. Wynik validacji przechowywany jest w prywatnym polu valid typu boolean, do którego dostęp mamy za pomocą publicznej metody isValid(), zwracającej wartość true jeśli validacja powiedzie się albo false w przeciwnym przypadku. Metoda getMessage() zwraca wiadomość zawartą w parametrze message utworzonej adnotacji.

```
public interface Validator {  
    void validate(String value);  
    boolean isValid();  
    String getMessage();  
}
```

- Przycisk Confirm powinien być aktywny tylko wtedy, gdy wprowadzona wartość jest poprawna. Dzięki temu, nie będzie możliwości wprowadzenia niepoprawnych danych.

- Validacja powinna odbywać się automatycznie w czasie rzeczywistym w trakcie wprowadzania danych

Klasa rozruchowa programu:

```
HelloApplication.java x
1 package com.example.walidacja;
2
3 import javafx.application.Application;
4 import javafx.fxml.FXMLLoader;
5 import javafx.scene.Scene;
6 import javafx.stage.Stage;
7 import java.io.IOException;
8
9 2 usages no.body
10 public class HelloApplication extends Application {
11     no.body
12     @Override
13     public void start(Stage stage) throws IOException {
14         FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("GUI.fxml"));
15         Scene scene = new Scene(fxmlLoader.load());
16         stage.setTitle("Hello!");
17         stage.setScene(scene);
18         stage.show();
19     }
20
21     no.body
22     public static void main(String[] args) { launch(); }
```

Klasa Pattern, która zawiera główny pattern do sprawdzania pola (email)

```
Pattern.java x
1 package com.example.walidacja;
2
3 no.body*
4 public class Pattern {
5     2 usages
6     @MyPattern(regex = "(?:[a-z0-9!#$%&'*/=?^_`{|}~-]+(?:\\.([a-z0-9!#$%&'*/=?^_`{|}~-]+)*)" +
7         "\\|"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01- +
8         "\\x09\\x0b\\x0c\\x0e-\\x7f])*")@(?:([a-z0-9](?:[a-z0-9-]*[a-z0-9]))?" +
9         "\\.|)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\\|([?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)" +
10         "\\.|){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\\x01-\\x08\\x +
11         "x0b\\x0c\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f]))+" +
12         "\\|")", message = "Niepoprawny adres email") //pattern do regexa
13     private String email;
14
15     no.body
16     public String getEmail() { return email; }
17
18     no.body
19     public void setEmail(String email) { this.email = email; }
20 }
```

## Interfejs Validator

```
Validator.java x
1 package com.example.walidacja;
2
3 public interface Validator {
4     void validate(String value);
5     boolean isValid();
6     String getMessage();
7 }
```

Interfejs MyPattern, pozwalająca na odwoływania się w czasie rzeczywistym dzięki adnotacjom

```
MyPattern.java x
1 package com.example.walidacja;
2
3 import java.lang.annotation.ElementType;
4 import java.lang.annotation.Retention;
5 import java.lang.annotation.Target;
6
7 @Target({ElementType.FIELD, java.lang.annotation.ElementType.TYPE}) //możemy użyć adnotacji tylko do pola lub do klasy
8 @Retention(java.lang.annotation.RetentionPolicy.RUNTIME) //dzięki tej adnotacji możemy się odwoływać w czasie działania programu
9 public @interface MyPattern {
10     String regex();
11     String message();
12 }
```

Klasa MyPatternValidator, implementująca interfejs Validator

```
MyPatternValidator.java x
1 package com.example.walidacja;
2
3 public class MyPatternValidator implements Validator{
4     private final MyPattern pattern;
5     private boolean isValid = false;
6     public MyPatternValidator(MyPattern pattern) { this.pattern = pattern; }
7
8     @Override
9     public void validate(String value) { isValid = value.matches(pattern.regex()); }
10
11     @Override
12     public boolean isValid() { return isValid; }
13
14     @Override
15     public String getMessage() { return pattern.message(); }
16 }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

## Klasa VinputText

```
VinputText.java
12 public class VinputText {
13     1 usage
14     Image error = new Image( url: "P:/Firefox/sign-error-icon.png"); // obrazek błędu
15     1 usage
16     Image valid = new Image( url: "P:/Firefox/istockphoto-1145805108-170667a.jpg"); // obrazek poprawności
17     3 usages
18     private final Tooltip imagetooltip = new Tooltip(); // tooltip odpowiedzialny za wyswietlenie informacji po najechaniu kursorem na obrazek
19     5 usages
20     private final TextInputControl textInput;
21     4 usages
22     private Validator validator;
23     3 usages
24     static private final List<Validator> validatorsList = new ArrayList<>(); // lista wszystkich walidatorów
25     2 usages
26     private int validatorsNumber; // liczba walidatorów
27     7 usages
28     private final ImageView image; // odpowiada za wyswietlenie obrazka
29     2 usages
30     private final Button button;
```

```

31 22
32     1 usage 1 no.body*
33     @ public VinputText(VBox vbox, Field field, Button button){
34         24
35         HBox hbox = new HBox(); // tworzy nowy HBox
36         hbox.setSpacing(30); // ustawia odstep miedzy elementami HBox
37         25
38         this.textInput = new TextField(); // tworzy nowy TextField
39         26
40         textInput.setPrefSize( prefWidth: 200, prefHeight: 40); // ustawia wielkość TextField
41         27
42         this.image = new ImageView(); // tworzy nowy ImageView
43         28
44         this.button = button; // przypisuje button
45         29
46         hbox.getChildren().add(image); // dodaje obrazek do HBox
47         30
48         hbox.getChildren().add(textInput); // dodaje TextField do HBox
49         31
50         hbox.getChildren().add(new Label(field.getName())); // dodaje Label do HBox
51         32
52         vbox.getChildren().add(hbox); // dodaje HBox do VBox
53         33
54     }
55     1 usage 1 no.body*
56     void addValidator(Validator v){ // dodaje walidator do listy walidatorów
57         35
58         this.validator = v;
59         36
60         validatorsList.add(v);
61         37
62     }
63     38
64 39
```

```

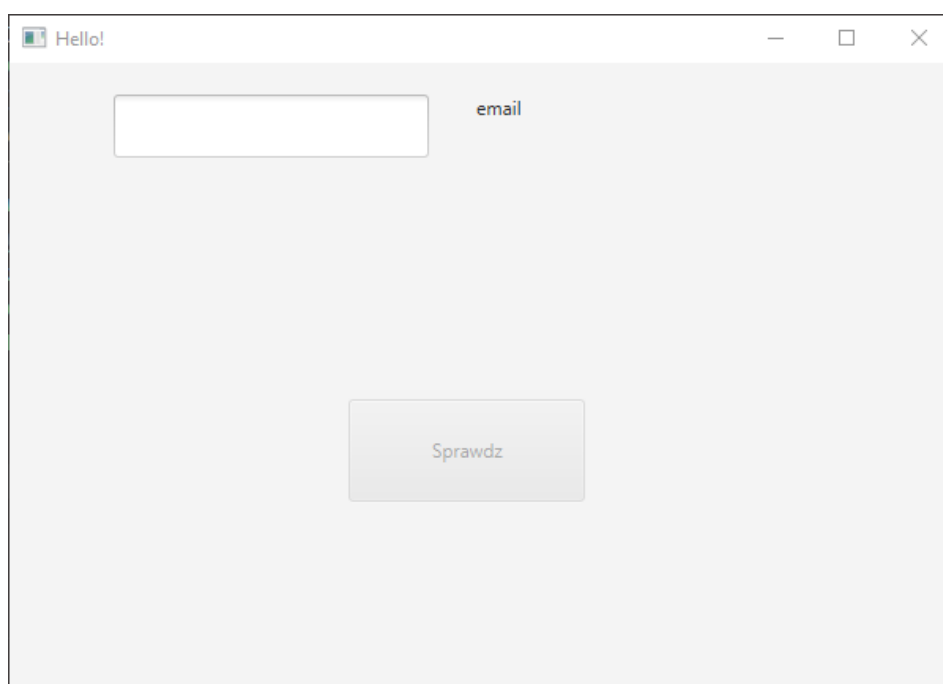
65 39
66     1 usage 1 no.body*
67     void checkField(){
68         40
69         imagetooltip.setText(validator.getMessage()); // ustawia tekst tooltipa
70         41
71         textInput.textProperty().addListener(((listener) -> { // dodaje listener do textProperty
72             42
73             validator.validate(textInput.getText()); // sprawdza czy pole jest poprawne
74             43
75             if (validator.isValid()) { // jeśli pole jest poprawne
76                 44
77                 image.setImage(valid); // ustawia obrazek poprawności
78                 45
79                 image.setOnMouseEntered(null); // usuwa listenera na obrazek
80                 46
81                 Tooltip.uninstall(image, imagetooltip); // usuwa tooltip
82                 47
83             }else{ // jeśli pole jest niepoprawne
84                 48
85                 image.setImage(error); // ustawia obrazek błędu
86                 49
87                 Tooltip.install(image, imagetooltip); // dodaje tooltip
88                 50
89             }
90             51
91         });
92         52
93         validatorsNumber = validatorsList.stream().filter(Validator::isValid).toList().size(); // liczba poprawnych walidatorów
94         53
95         button.setDisable(!(validatorsList.size() == validatorsNumber)); //jeżeli liczba poprawnych walidatorów jest
96         54
97         // równa liczbie wszystkich walidatorów to przycisk sprawdz jest aktywny
98         55
99     });
100     56
101 }
102 57
103 }
104 58
105 59
```

## Klasa główna programu - Controller

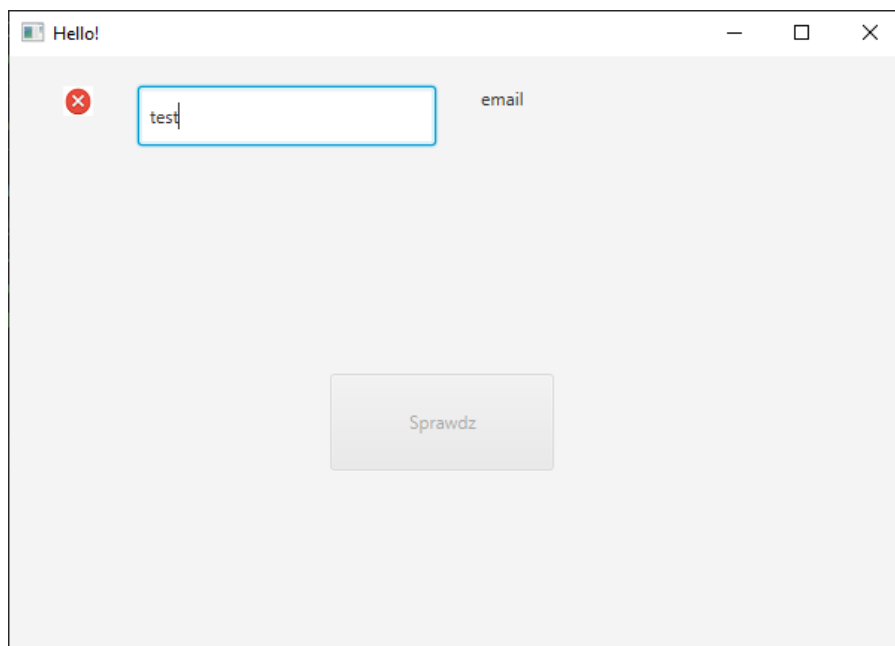
```
Controller.java x
1 usage  no.body *
9  public class Controller {
10
11      3 usages
12      @FXML
13      private Button sprawdz;
14
15      2 usages
16      @FXML
17      private VBox mainVBox;
18
19      no.body *
20      @FXML
21      void initialize() throws ClassNotFoundException {
22          sprawdz.setDisable(true); // przycisk sprawdz jest wyłączony
23          Class<?> cl = Class.forName("com.example.walidacja.Pattern"); // wczytuje klasę Pattern
24          Field[] fields = cl.getDeclaredFields(); // wczytuje wszystkie pola klasy Pattern
25          Arrays.stream(fields)
26              .filter(field -> field.isAnnotationPresent(MyPattern.class))
27              .forEach(field -> { // dla każdego pola znajdujące się w klasie Pattern
28                  MyPatternValidator mpv = new MyPatternValidator(field.getAnnotation(MyPattern.class));
29                  // tworzy nowy obiekt MyPatternValidator
30                  VinputText vinput = new VinputText(mainVBox, field, sprawdz); // tworzy nowy obiekt VinputText
31                  vinput.addValidator(mpv); // dodaje nowy obiekt MyPatternValidator do listy walidatorów
32                  vinput.checkField(); // sprawdza czy pole jest poprawne
33              });
34      }
35  }
```

Działanie aplikacji:

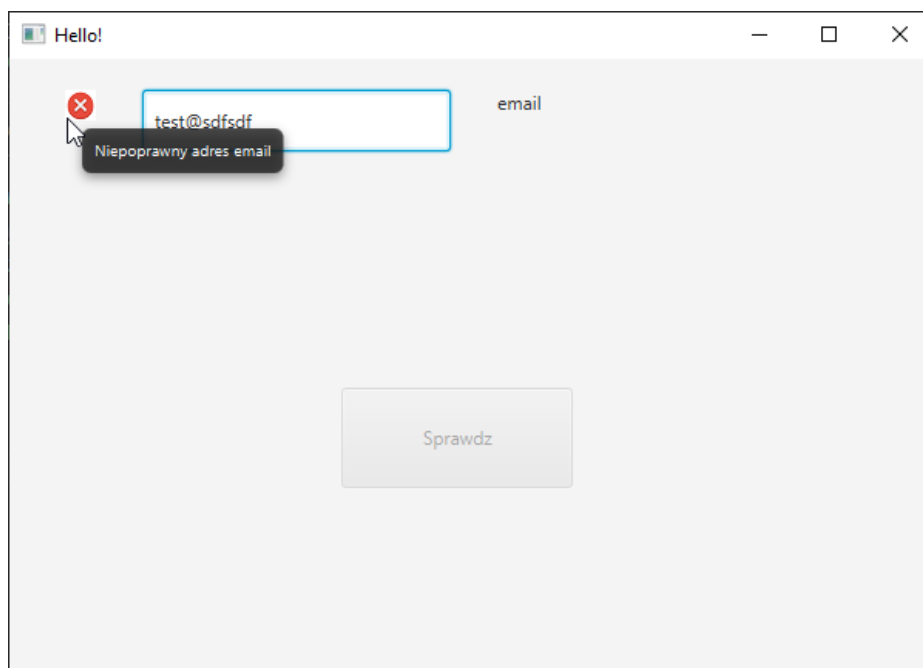
Uruchomienie:



Niepoprawny email:

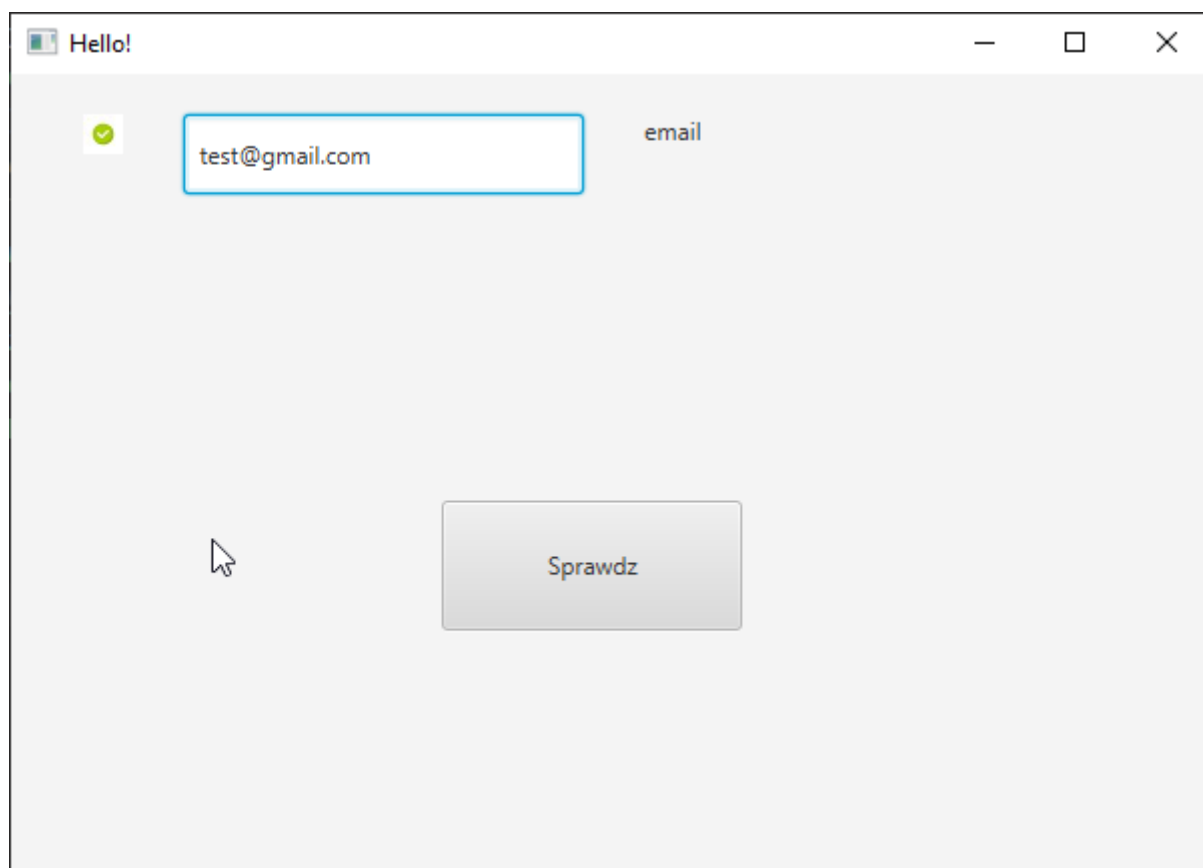


A screenshot of a web application window titled "Hello!". The window has a light gray background and standard window controls (minimize, maximize, close) in the top right corner. On the left side, there is a red circle with a white 'x' icon. To its right is a text input field with a blue border containing the text "test". Further to the right is the label "email". Below these elements, centered in the window, is a gray button with the text "Sprawdz".



A screenshot of the same web application window, but now the text input field contains the text "test@sdfsdf". A tooltip message "Niepoprawny adres email" (Incorrect email address) is displayed over the input field. The red 'x' icon is still present. The "Sprawdz" button remains centered at the bottom.

Poprawny email:



A web form titled "Hello!" is displayed. It features a green checkmark icon on the left. In the center, there is an input field containing the text "test@gmail.com". To the right of the input field is a label "email". Below the input field, there is a button labeled "Sprawdz". A mouse cursor is visible near the button.