

## **Programowanie Zdarzeniowe**

Praca domowa – Powoływanie obiektów klas za pomocą refleksji z wykorzystaniem graficznego interfejsu użytkownika

Prowadzący: \*\*\*

Grupa: \*\*\*

Wykonał: **Mateusz Malec**

# **OPIS GRAFICZNEGO INTERFEJSU UŻYTKOWNIKA**

The image shows a screenshot of a Windows application window titled "Obiektowywoływacz 3000". The window has a blue title bar with standard minimize, maximize, and close buttons. The main interface is light gray and contains several sections:

- Top section:** A label "Podaj ścieżkę do folderu:" followed by a text input field and a button labeled "Ustaw ścieżkę".
- Lista klas:** A large, empty rectangular box with a thin black border.
- Konstruktory:** Another large, empty rectangular box with a thin black border.
- Parameters:** A label "Parametry rozdzielone średnikiem" followed by a text input field and a button labeled "Wywołaj".
- Pola utworzonego obiektu:** A large, empty rectangular box with a thin black border.

**Okno aplikacji po uruchomieniu.**

1. Podajemy poprawną ścieżkę do katalogu nad startowym pakietem klas, które chcemy załadować.

Obiektowywoławcz 3000

Podaj ścieżkę do folderu:

Lista klas

```
class pl.edu.wat.wcy.pz.przyklad.Aplikacja  
class pl.edu.wat.wcy.pz.przyklad.Klasa1  
class pl.edu.wat.wcy.pz.przyklad.pakiet1.Klasa2  
class pl.edu.wat.wcy.pz.przyklad.pakiet2.Klasa4
```

Konstruktory

Parametry rozdzielone średnikiem

Pola utworzonego obiektu

2. Klikamy przycisk

W przypadku podania błędnej ścieżki...

The screenshot shows a window titled "Obiektowywoływacz 3000". It has a text input field labeled "Podaj ścieżkę do folderu:" containing the path "C:\Projekty\PrzykladKlas\bledna\_sciezka". To the right of the input field is a button labeled "Ustaw ścieżkę". Below the input field are two empty rectangular boxes labeled "Lista klas" and "Konstruktory". Below these is a text input field labeled "Parametry rozdzielone średnikiem". To the right of this field is a button labeled "Wywołaj". At the bottom, there is a section titled "Pola utworzonego obiektu" which contains the error message: "Nie udało się załadować plików z podanej ścieżki!".

...otrzymujemy komunikat o błędzie.

To pole tekstowe, oprócz wyświetlania pól utworzonego obiektu, służy jako konsola do wyświetlania błędów

Obiektowywoływacz 3000

Podaj ścieżkę do folderu:

Lista klas

- class pl.edu.wat.wcy.pz.przyklad.Aplikacja
- class pl.edu.wat.wcy.pz.przyklad.Klasa1
- class pl.edu.wat.wcy.pz.przyklad.pakiet1.Klasa2
- class pl.edu.wat.wcy.pz.przyklad.pakiet2.Klasa4

Konstruktory

```
private pl.edu.wat.wcy.pz.przyklad.Klasa1(int)
private pl.edu.wat.wcy.pz.przyklad.Klasa1(double)
public pl.edu.wat.wcy.pz.przyklad.Klasa1(int,double)
```

Parametry rozdzielone średnikiem

Pola utworzonego obiektu

W przypadku podania poprawnej ścieżki...

1. Wybranie klasy z listy, ładuje jej konstruktory.

Obiektowywoływacz 3000

Podaj ścieżkę do folderu:

Lista klas

```
class pl.edu.wat.wcy.pz.przyklad.Aplikacja
class pl.edu.wat.wcy.pz.przyklad.Klasa1
class pl.edu.wat.wcy.pz.przyklad.pakiet1.Klasa2
class pl.edu.wat.wcy.pz.przyklad.pakiet2.Klasa4
```

Konstruktory

```
private pl.edu.wat.wcy.pz.przyklad.Klasa1(int)
private pl.edu.wat.wcy.pz.przyklad.Klasa1(double)
public pl.edu.wat.wcy.pz.przyklad.Klasa1(int,double)
```

Parametry rozdzielone średnikiem

Pola utworzonego obiektu

Wybranie konstruktora, pozwala na utworzenie za jego pomocą instancji zaznaczonej klasy. Przycisk „Wywołaj” staje się aktywny.

Obiektowywoływacz 3000

Podaj ścieżkę do folderu:

Lista klas

```
class pl.edu.wat.wcy.pz.przyklad.Aplikacja
class pl.edu.wat.wcy.pz.przyklad.Klasa1
class pl.edu.wat.wcy.pz.przyklad.pakiet1.Klasa2
class pl.edu.wat.wcy.pz.przyklad.pakiet2.Klasa4
```

Konstruktory

```
private pl.edu.wat.wcy.pz.przyklad.Klasa1(int)
private pl.edu.wat.wcy.pz.przyklad.Klasa1(double)
public pl.edu.wat.wcy.pz.przyklad.Klasa1(int,double)
```

Parametry rozdzielone średnikiem

 

Pola utworzonego obiektu

Nie udało się utworzyć obiektu!  
Błędny format parametrów.

Zaznaczony konstruktor przyjmuje jeden parametr prymitywnego typu double.

Pole tekstowe pozwala nam na podanie wartości parametrów, które zostaną przekazane do konstruktora.

Przy próbie powołania instancji obiektu z błędnymi parametrami, otrzymamy komunikat o błędzie.



Obiektowywoływacz 3000

Podaj ścieżkę do folderu:

**Lista klas**

- class pl.edu.wat.wcy.pz.przyklad.Aplikacja
- class pl.edu.wat.wcy.pz.przyklad.Klasa1
- class pl.edu.wat.wcy.pz.przyklad.pakiet1.Klasa2
- class pl.edu.wat.wcy.pz.przyklad.pakiet2.Klasa4

**Konstruktory**

- private pl.edu.wat.wcy.pz.przyklad.Klasa1(int)
- private pl.edu.wat.wcy.pz.przyklad.Klasa1(double)
- public pl.edu.wat.wcy.pz.przyklad.Klasa1(int,double)

**Parametry rozdzielone średnikiem**

**Pola utworzonego obiektu**

Obiekt klasy pl.edu.wat.wcy.pz.przyklad.Klasa1

- private int liczba : 0
- private double liczba\_pp : 123.456

Gdy format parametrów będzie zgodny z typami parametrów oczekiwanymi przez konstruktor...

...jeśli nie wystąpią inne wyjątki związane z powoływaniem obiektu za pomocą Java reflection API, zostaną wyświetlone wszystkie pola obiektu, wraz z ich wartościami.

# Przykład poprawnego powołania obiektu

Obiektowywoływacz 3000

Podaj ścieżkę do folderu:

Lista klas

- class pl.edu.wat.wcy.pz.przyklad.Aplikacja
- class pl.edu.wat.wcy.pz.przyklad.Klasa1
- class pl.edu.wat.wcy.pz.przyklad.pakiet1.Klasa2
- class pl.edu.wat.wcy.pz.przyklad.pakiet2.Klasa4

Konstruktory

- private pl.edu.wat.wcy.pz.przyklad.pakiet2.Klasa4()
- private pl.edu.wat.wcy.pz.przyklad.pakiet2.Klasa4(java.lang.String,int)

Parametry rozdzielone średnikiem

Pola utworzonego obiektu

Obiekt klasy pl.edu.wat.wcy.pz.przyklad.pakiet2.Klasa4

private static int statycznaLiczba : 50

private java.lang.String opisObiektu : Przykładowy tekst

Kolejność wartości parametrów musi być zgodna z kolejnością występowania parametrów w konstruktorze

np. Dla zaznaczonego konstruktora -> parametry(String , int)

Wartości ->

(tekst;10) ✓

(10; tekst) ✗

**OPIS KODU ŹRÓDŁOWEGO**

## Widok – główne okno AppFrame

```
public class App {  
  
    public static void main( String[] args ) {  
  
        EventQueue.invokeLater(new Runnable() {  
            @Override  
            public void run() {  
                new AppFrame();  
            }  
        });  
    }  
}
```

Główne okno aplikacji  
Pobiera ono instancję kontrolera odpowiedzialnego za komunikację z modelem, czyli klasą FolderClassLoaderSingleton, która powołuje UMLClassLoader'a oraz odpowiada za współpracę z Java reflection API  
AppFrame tworzy także dwa panele i zapisuje je jako słuchaczy notyfikacji kontrolera.  
header – odpowiedzialny za widok ścieżki  
center – odpowiedzialny za widok list klas i konstruktorów, oraz pól powołanego obiektu

Przekazanie głównego okna aplikacji do wywołania w EDT.

```
public class AppFrame extends JFrame {  
    private HeaderPanel header;  
    private CenterPanel center;  
    private ClassLoaderController clc;  
    public AppFrame() {  
        super("Obiektowywoływacz 3000");  
        clc = ClassLoaderController.getInstance();  
  
        header = new HeaderPanel(clc);  
        center = new CenterPanel(clc);  
        clc.addClassListener(center);  
        clc.addConstructorListener(center);  
        clc.addObjectListener(center);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new BorderLayout());  
        setSize(525, 550);  
        setMinimumSize(new Dimension(525, 350));  
        setLocation(100, 100);  
  
        setResizable(true);  
        setVisible(true);  
        this.add(header, BorderLayout.NORTH);  
        this.add(center, BorderLayout.CENTER);  
        //this.pack();  
    }  
}
```

## Widok – HeaderPanel header

```
public class HeaderPanel extends JPanel implements ActionListener{

    private ClassLoaderController clc;

    private JTextField classPathField;
    private JButton classPathButton;
    private JLabel label;

    public HeaderPanel(ClassLoaderController clc){
        this.clc = clc;

        classPathField = new JTextField(20);
        classPathButton = new JButton("Ustaw ścieżkę");
        label = new JLabel("Podaj ścieżkę do folderu: ");
        setLayout(new FlowLayout(FlowLayout.LEFT));
        setMinimumSize(new Dimension(525,10));
        this.add(label);
        this.add(classPathField);
        this.add(classPathButton);

        classPathButton.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        clc.loadClasses(classPathField.getText());
    }

}
```

Jest on słuchaczem własnego przycisku odpowiedzialnego za załadowanie ścieżki. Po odnotowaniu akcji wciśnięcia przycisku, wywołuje on metodę kontrolera odpowiedzialną za załadowanie klas, przekazując ścieżkę podaną przez użytkownika.

## Widok – CenterPanel center

```
public class CenterPanel extends JPanel implements ClassListener, ConstructorListener, ObjectListener{
```

...

Implementuje on interfejsy słuchaczy zdarzeń kontrolera, które komunikują odpowiednio o załadowaniu klas, konstruktorów oraz próbie powołania instancji klasy.

W przypadku otrzymania jednej z tych notyfikacji pobiera on przekazane przez odpowiedni obiekt zdarzenia klasy, konstruktory, lub obiekt klasy, a także ewentualny komunikat o błędzie i po zaktualizowaniu swojego stanu odrysowuje się.

```
    public void objectReceived(ObjectEvent event){
        objDetails.setText(event.toString());
        repaint();
    }

    public void constructorsReceived(ConstructorEvent event){
        DefaultListModel constructorNamesModel = new DefaultListModel();
        for (Constructor name : event.getConstructors()) {
            constructorNamesModel.addElement(name);
        }
        constructorsList.setModel(constructorNamesModel);
    }

    public void classesReceived(ClassEvent event) {
        DefaultListModel classNamesModel = new DefaultListModel();
        if(event.getOpCode() == ClassEvent.CLEAR_LIST){
            classesList.setModel(classNamesModel);
        }
        else if(event.getOpCode() == ClassEvent.LOAD_LIST) {
            for (Class clazz : event.getClasses()) {
                classNamesModel.addElement(clazz);
            }
            classesList.setModel(classNamesModel);
            objDetails.setText("");
        }
        else if(event.getOpCode() == ClassEvent.ERROR){
            classesList.setModel(classNamesModel);
            constructorsList.setModel(classNamesModel);
            objDetails.setText(event.getErrorMessage());
        }
        repaint();
    }
}
```

## Widok – CenterPanel center cd.

```
class NewInstanceListener implements ActionListener{  
    public void actionPerformed(ActionEvent e) {  
        clc.createInstanceWithParameters(constructor, params.getText());  
    }  
}
```

Słuchacz przycisku powołania instancji klasy

```
class ConsListListener implements ListSelectionListener{  
    public void valueChanged(ListSelectionEvent e) {  
        if (e.getValueIsAdjusting() == false) {  
  
            if (constructorsList.getSelectedIndex() == -1) {  
                invokeButton.setEnabled(false);  
                System.out.println("Nie zaznaczono zadnego konstruktora");  
                repaint();  
            } else {  
                System.out.println("Zaznaczono konstruktork " + constructorsList.getSelectedValue().toString());  
                constructor = (Constructor) constructorsList.getSelectedValue();  
                invokeButton.setEnabled(true);  
                it.repaint();  
            }  
        }  
    }  
}
```

Słuchacz wybrania konstruktora z listy

```
class ClassListListener implements ListSelectionListener{  
    public void valueChanged(ListSelectionEvent e) {  
        if (e.getValueIsAdjusting() == false) {  
  
            if (classesList.getSelectedIndex() == -1) {  
                System.out.println("Nie zaznaczono zadnej klasy");  
            } else {  
                Class clazz = (Class)classesList.getSelectedValue();  
                System.out.println("Zaznaczono klasę " + clazz.toString());  
                clc.loadConstructors(clazz);  
            }  
        }  
    }  
}
```

Słuchacz wybrania klasy z listy

Zawiera on także klasy wewnętrzne implementujące słuchaczy zdarzeń kliknięcia przycisku, powołania nowej instancji klasy oraz zaznaczenia elementu na jednej z dwóch list – klas i kontrolerów

## Kontroler – ClassLoaderController clc

```
public class ClassLoaderController {
    static FolderClassLoaderSingleton fcl = FolderClassLoaderSingleton.getInstance();
    private static ClassLoaderController instance;
    private List<ClassListener> classListeners = new ArrayList();
    private List<ConstructorListener> constructorListeners = new ArrayList();
    private List<ObjectListener> objectListeners = new ArrayList();
    private ClassLoaderController() { }

    public static ClassLoaderController getInstance(){
        if(instance == null) instance = new ClassLoaderController();
        return instance;
    }

    public void loadClasses(String path){
        try {
            fcl.loadClasses(path);
            fireClassEvent(this, ClassEvent.LOAD_LIST, fcl.getLoadedClasses());
        } catch (ClassLoaderException e) {
            fireClassEvent(this, ClassEvent.ERROR, e.getErrorMessage());
        }
    }

    public void loadConstructors(Class clazz){
        fireConstructorEvent(this, fcl.getConstructors(clazz));
    }
}
```

wywołuje w modelu ładowanie klas z podanej ścieżki, notyfikuje słuchaczy o załadowaniu klas, lub o błędzie

wywołuje w modelu ładowanie konstruktorów, notyfikuje słuchaczy konstruktorów

Jest on warstwą komunikacji widoku aplikacji z modelem, implementuje wzorzec Singleton, chyba trochę niepotrzebnie, ponieważ model również implementuje ten wzorzec. W zamyśle chciałem zadbać aby model operował tylko na jednej instancji UMLClassLoader'a, a później wyciągając z niego kontroler, pomyślałem, że jeśli będzie singletonem, wszyscy słuchacze będą automatycznie przypisywani do jednej instancji kontrolera.



## Kontroler – ClassLoaderController clc cd.

Wywołuje w modelu powołanie nowej instancji klasy

```
public void createInstanceWithParameters(Constructor constructor, String parameters){  
    Object object = null;  
    try {  
        object = fcl.createInstanceWithParameters(constructor, parameters);  
        fireObjectEvent(this, object);  
    } catch (ClassLoaderException e) {  
        fireObjectEvent(this, e.getErrMessage());  
    }  
}
```

Wyjątek dziedziczący po RuntimeException pozwalający na wyrzucanie spersonalizowanych komunikatów o błędach

## Wyjątek – ClassLoaderException

Pozwala na zastąpienie wyjątku wyrzuconego w modelu, na spersonalizowaną wiadomość, która może być przekazana poprzez notyfikację słuchacza do wyświetlenia w widoku.

```
public class ClassLoaderException extends RuntimeException{  
    private String errMessage;  
  
    public ClassLoaderException(String errMessage){  
        super();  
        this.errMessage = errMessage;  
    }  
    public String getErrMessage(){return errMessage;}  
}
```

# Kontroler – ClassLoaderController clc cd.

## Metody obsługujące słuchaczy

```
public synchronized void addClassListener( ClassListener l ) {
    classListeners.add( l );
}
public synchronized void removeClassListener( ClassListener l ) {
    classListeners.remove( l );
}
private synchronized void fireClassEvent(Object source,final int opCode, List<Class> classes) {
    ClassEvent ce = new ClassEvent(source,opCode, classes);
    for(ClassListener l : classListeners) l.classesReceived(ce);
}
private synchronized void fireClassEvent(Object source,final int opCode, String errMessage) {
    ClassEvent ce = new ClassEvent(source,opCode, errMessage);
    for(ClassListener l : classListeners) l.classesReceived(ce);
}
public synchronized void addConstructorListener( ConstructorListener l ) {
    constructorListeners.add( l );
}
public synchronized void removeConstructorListener( ConstructorListener l ) {
    constructorListeners.remove( l );
}
private synchronized void fireConstructorEvent(Object source, List<Constructor> constructors) {
    ConstructorEvent ce = new ConstructorEvent(source, constructors);
    for(ConstructorListener l : constructorListeners) l.constructorsReceived(ce);
}
public synchronized void addObjectListener( ObjectListener l ) {
    objectListeners.add( l );
}
public synchronized void removeObjectListener( ObjectListener l ) {
    objectListeners.remove( l );
}
private synchronized void fireObjectEvent(Object source, Object instance) {
    ObjectEvent oe = new ObjectEvent(source, instance);
    for(ObjectListener l : objectListeners) l.objectReceived(oe);
}
private synchronized void fireObjectEvent(Object source, String errorMessage) {
    ObjectEvent oe = new ObjectEvent(source, errorMessage);
    for(ObjectListener l : objectListeners) l.objectReceived(oe);
}
```

## Model – FolderClassLoaderSingleton fcl

```
private void listFilesForFolder(final File folder, String prefix) {
    for (final File fileEntry : folder.listFiles()) {
        if (fileEntry.isDirectory()) {
            listFilesForFolder(fileEntry, prefix + fileEntry.getName() + ".");
            System.out.println("Nazwa folderu " + fileEntry.getName());
        } else if (fileEntry.getName().endsWith(".class")) {
            String entryName = prefix + fileEntry.getName();
            System.out.println("Odnaleziono " + entryName);
            classNames.add(entryName);
        }
    }
}

public void loadClasses(String path) throws ClassLoaderException {
    classes.clear();
    classNames.clear();
    try {
        final File folder = new File(path);
        listFilesForFolder(folder, "");
        URL[] urls = {folder.toURI().toURL()};
        URLClassLoader cl = URLClassLoader.newInstance(urls);
        for (String className : classNames) {
            className = className.substring(0, className.length() - 6);
            Class c = cl.loadClass(className);
            classes.add(c);
            System.out.println("Załadowano klasę " + className);
        }
    } catch (MalformedURLException | ClassNotFoundException | NullPointerException e) {
        Logger.getGlobal().severe(e.getMessage());
        throw new ClassLoaderException("Nie udało się załadować plików z podanej ścieżki!");
    }
}
```

Metoda przechodząca rekurencyjnie po folderach podanej ścieżki, automatycznie tworzy przedrostki z nazwami pakietów dla znajdowanych klas

Metoda powołująca instancję UMLClassLoader'a, ładuje znalezione klasy, w przypadku niepowodzenia wyrzuca spersonalizowany komunikat

Metody odczytujące pliki \*.class z pakietów znajdujących się w podanym przez użytkownika folderze, oraz ładujące znalezione klasy.

## Model – FolderClassLoaderSingleton fcl cd.

```
public List<Constructor> getConstructors(Class clazz){
    Constructor[] constructors = clazz.getDeclaredConstructors();
    List<Constructor> conList = new ArrayList<Constructor>();
    for(Constructor c : constructors) conList.add(c);
    return conList;
}
```

Metoda zwracająca listę konstruktorów podanej klasy

Metoda powołująca nową instancję klasy, do której należy przekazany jako parametr konstruktor, rozdziela wartości dla parametrów konstruktora przekazane jako String. Zgodnie z założeniami może powoływać tylko typy prymitywne lub typy opakowujące prymitywne, oraz String, na potrzeby prezentacji pracy domowej zaimplementowałem obsługę jedynie dwóch typów liczbowych, ale można dodać tutaj dowolny typ przyjmujący jako parametry konstruktora typy prymitywne lub String.

```
public Object createInstanceWithParameters(Constructor constructor, String parameters) throws ClassNotFoundException{
    Object newObject = null;
    String[] values = parameters.split(";");
    Class[] paramClasses = constructor.getParameterTypes();
    Object[] objs = new Object[paramClasses.length];
    for(int i=0;i<paramClasses.length;i++){
        try {
            if (paramClasses[i].equals(String.class)) objs[i] = values[i];
            else if (paramClasses[i].equals(int.class) || paramClasses[i].equals(Integer.class)) objs[i] = new Integer(values[i]);
            else if (paramClasses[i].equals(double.class) || paramClasses[i].equals(Double.class)) objs[i] = new Double(values[i]);
        } catch (Exception e2){
            Logger.getGlobal().severe(e2.getMessage());
            throw new ClassNotFoundException("Nie udało się utworzyć obiektu!\nBłędny format parametrów.");
        }
    }
    constructor.setAccessible(true);
    try {
        newObject = constructor.newInstance(objs);
    } catch (Exception e1){
        Logger.getGlobal().severe(e1.getMessage());
        throw new ClassNotFoundException("Nie udało się utworzyć obiektu!\nBłąd przy wywoływaniu konstruktora.");
    }
    return newObject;
}
```

# Interfejsy słuchaczy

```
public interface ClassListener {  
    void classesReceived(ClassEvent event);  
}  
  
public interface ConstructorListener {  
    void constructorsReceived(ConstructorEvent event);  
}  
  
public interface ObjectListener {  
    void objectReceived(ObjectEvent event);  
}
```

Na początku napisałem jeden interfejs słuchacza i zdarzenie generyczne, którego typ pola określany był w momencie tworzenia, ale takie zdarzenie nie dało się w łatwy sposób przeiterować pętlą `forEach`, w wielu miejscach w kodzie programu musiałbym wykonywać rzutowanie na odpowiedni typ, więc zrezygnowałem z tego pomysłu.

## Klasa obiektu zdarzenia - ClassEvent

```
public class ClassEvent extends EventObject {
    public static final int LOAD_LIST = 0;
    public static final int CLEAR_LIST = 1;
    public static final int ERROR = 2;

    private final int opCode;
    private List<Class> classes;
    private String errMessage = "";

    public ClassEvent(Object source, final int opCode, List<Class> classes) {
        super(source);
        this.opCode = opCode;
        this.classes = classes;
    }

    public ClassEvent(Object source, final int opCode, String errMessage) {
        super(source);
        this.opCode = opCode;
        this.classes = null;
        this.errMessage = errMessage;
    }

    public List<Class> getClasses() {
        return classes;
    }

    public int getOpCode() {return opCode; }
    public String getErrMessage() {return errMessage;}
}
```

Stałe informują o stanie modelu po próbie załadowania klas, w zasadzie są one zbędne, stała CLEAR\_LIST nigdy nie jest używana, jest ona pozostałością, z resztą jak wszystkie, po pierwszej wersji kodu, którą napisałem. Teraz myślę, że mógłbym po prostu wydzielić osobną klasę słuchacza do nasłuchiwanie błędów.

## Klasa obiektu zdarzenia - ConstructorEvent

```
public class ConstructorEvent extends EventObject {  
    private List<Constructor> constructors;  
    public ConstructorEvent(Object source, List<Constructor> constructors) {  
        super(source);  
        this.constructors = constructors;  
    }  
    public List<Constructor> getConstructors() {  
        return constructors;  
    }  
}
```

Przechowuje listę konstruktorów pobranych z klasy zaznaczonej na liście klas widoku.

## Klasa obiektu zdarzenia - ObjectEvent

```
public class ObjectEvent extends EventObject {
    private String errorMessage;
    private Object instance;

    public ObjectEvent(Object source, Object instance) {
        super(source);
        this.instance = instance;
    }
    public ObjectEvent(Object source, String errorMessage) {
        super(source);
        this.instance = null;
        this.errorMessage = errorMessage;
    }

    public Object getInstance() {return instance;}
    public String toString(){
        if(instance == null) return errorMessage;
        Class clazz = instance.getClass();
        Field[] fields = clazz.getDeclaredFields();
        StringBuilder sb = new StringBuilder();
        sb.append("Obiekt klasy ").append(clazz.getName()).append("\n");
        for (Field f : fields) {
            f.setAccessible(true);
            sb.append(Modifier.toString(f.getModifiers())).append(" ").append(f.getType().getName()).append(" ");

            sb.append(f.getName()).append(" : ");
            try {
                Object v = f.get(instance);
                sb.append(v.toString()).append("\n");
            } catch (IllegalAccessException el) {
                Logger.getGlobal().severe(el.getMessage());
            }
        }
        return sb.toString();
    }
}
```

Metoda toString tej klasy zwraca sformatowany tekst, który jest gotowy do wyświetlenia w polu tekstowym widoku z opisem powołanego obiektu, jeśli wystąpił błąd w wywołaniu obiektu, zwraca ona treść komunikatu o błędzie.