Programowanie Zdarzeniowe

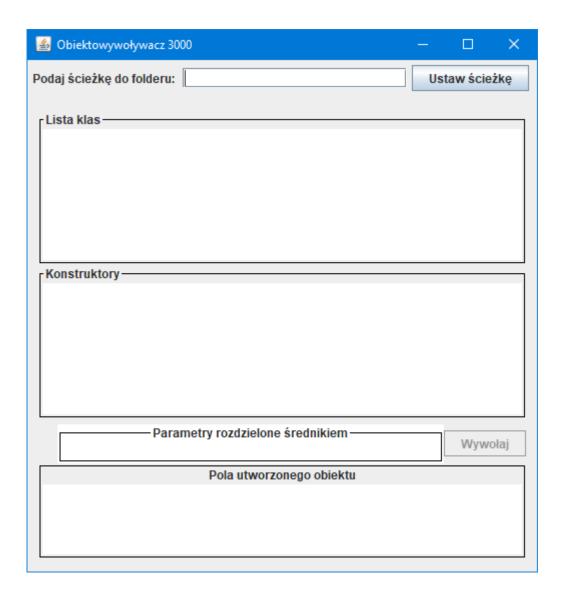
Praca domowa – Powoływanie obiektów klas za pomocą refleksji z wykorzystaniem graficznego interfejsu użytkownika

Prowadzący: ***

Grupa: ***

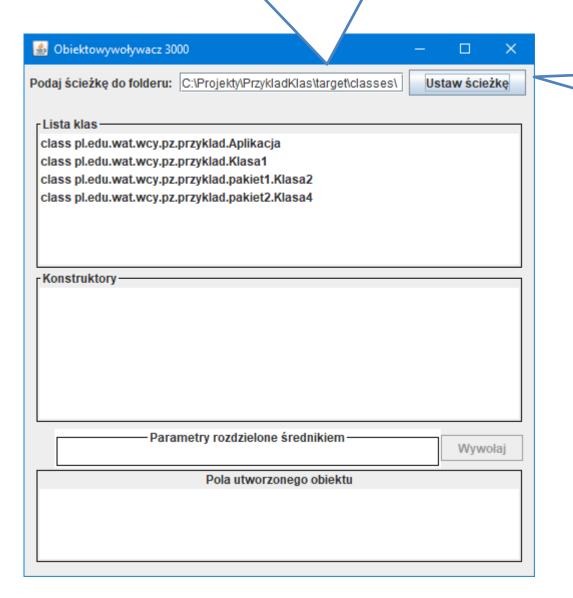
Wykonał: Mateusz Malec

OPIS GRAFICZNEGO INTERFEJSU UŻYTKOWNIKA



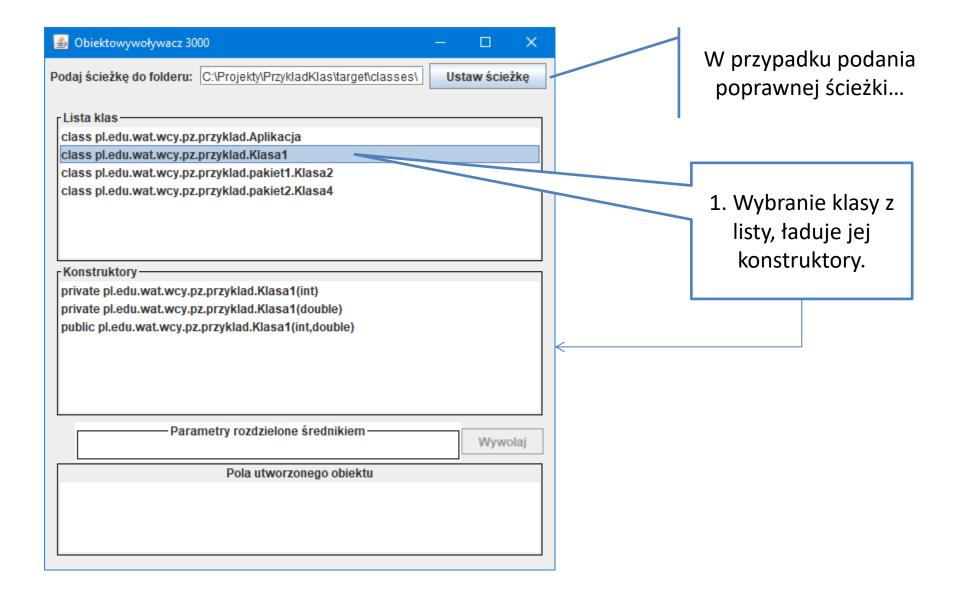
Okno aplikacji po uruchomieniu.

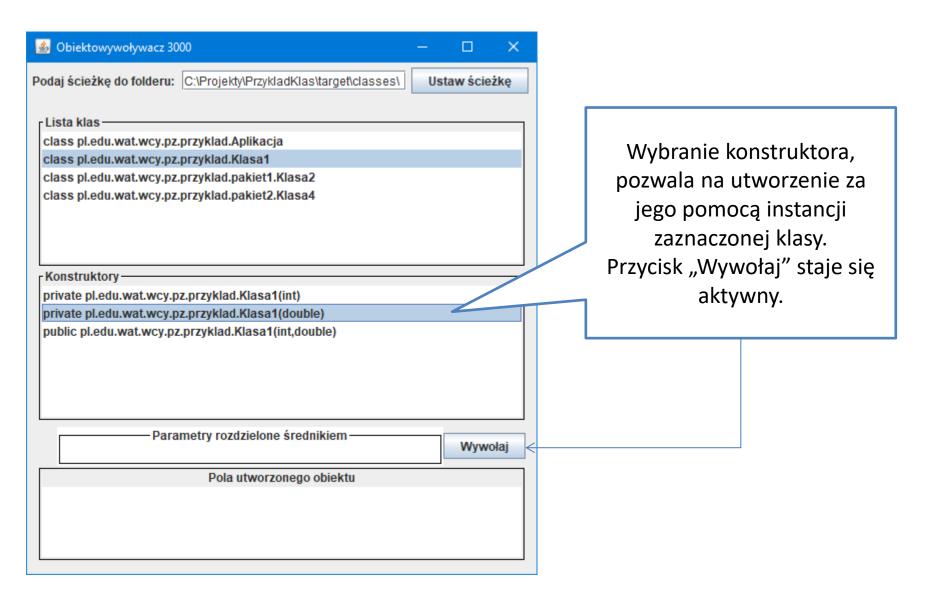
1. Podajemy poprawną ścieżkę do katalogu nad startowym pakietem klas, które chcemy załadować.

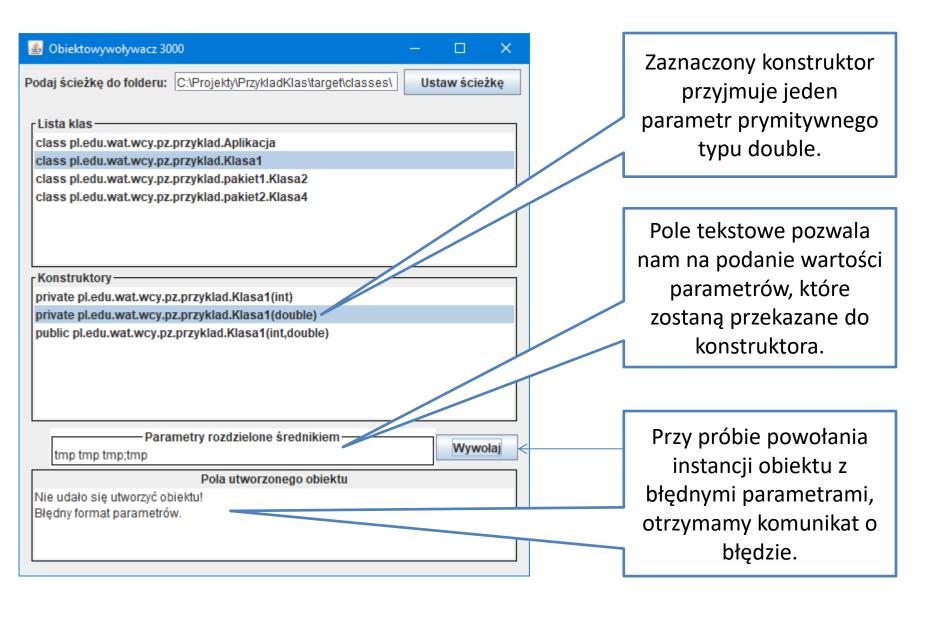


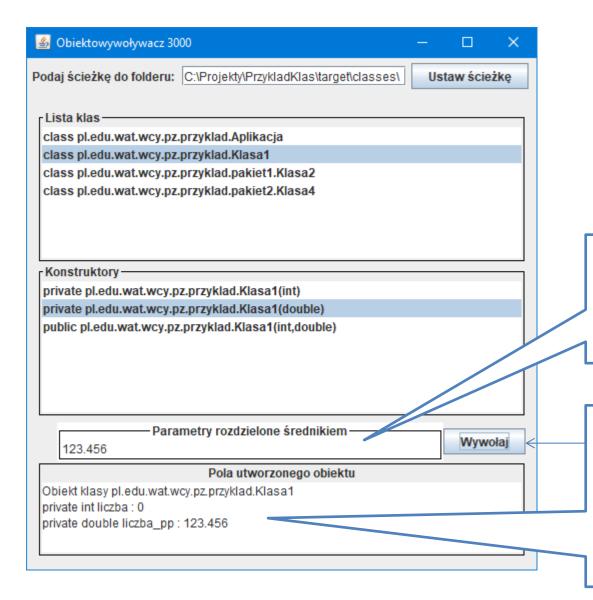
2. Klikamy przycisk

W przypadku podania błędnej ścieżki... 🙆 Obiektowywoływacz 3000 Podaj ścieżkę do folderu: C:\Projekty\PrzykladKlas\bledna_sciezka Ustaw ścieżkę - Lista klas-...otrzymujemy komunikat o błędzie. -Konstruktory-To pole tekstowe, Parametry rozdzielone średnikiem oprócz wyświetlania Wywołaj pól utworzonego Pola utworzonego obiektu obiektu, służy jako Nie udało się załadować plików z podanej ścieżki! konsola do wyświetlania błędów





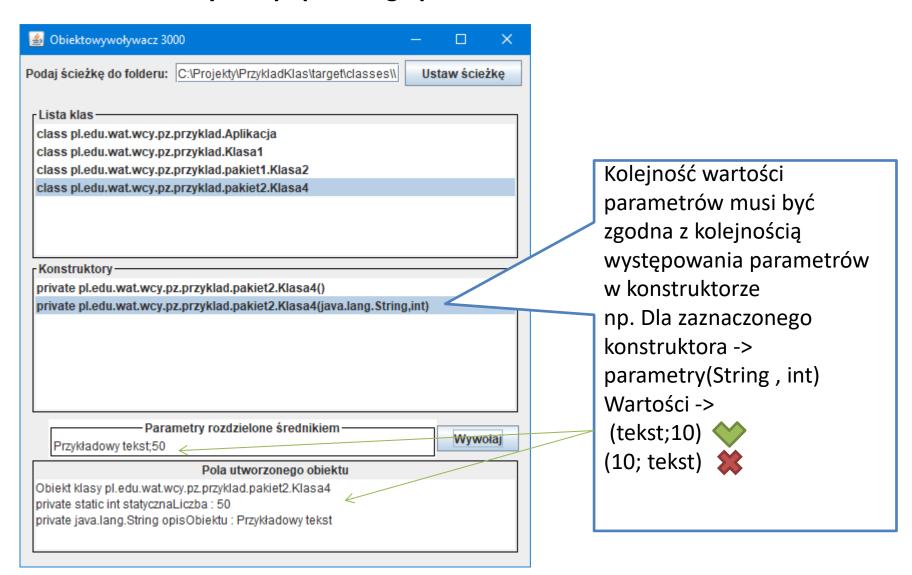




Gdy format parametrów będzie zgodny z typami parametrów oczekiwanymi przez konstruktor...

...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



OPIS KODU ŹRÓDŁOWEGO

Widok – główne okno AppFrame

Przekazanie głównego okna aplikacji do wywołania w EDT.

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

```
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 ścieżke");
        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.getErrMessage());
    repaint();
```

Widok - CenterPanel center cd.

```
class NewInstanceListener implements ActionListener{
                                                       Słuchacz przycisku powołania instancji klasy
    public void actionPerformed(ActionEvent e) {
        clc.createInstanceWithParameters(constructor, params.getText());
                                                                    Słuchacz wybrania konstruktora
class ConsListListener implements ListSelectionListener{
    public void valueChanged(ListSelectionEvent e) {
                                                                    z listy
        if (e.getValueIsAdjusting() == false) {
            if (constructorsList.getSelectedIndex() == -1) {
                invokeButton.setEnabled(false);
                System.out.println("Nie zaznaczono zadnego konstruktora");
                repaint();
            } else {
                System.out.println("Zaznaczono konstuktor " + constructorsList.getSelectedValue().toString());
                constructor = (Constructor) constructorsList.getSelectedValue();
                invokeButton.setEnabled(true);
                it.repaint();
                                               Słuchacz wybrania klasy z listy
                                                                          Zawiera on także klasy
class ClassListListener implements ListSelectionListener{
   public void valueChanged(ListSelectionEvent e) {
                                                                          wewnętrzne implementujące
       if (e.getValueIsAdjusting() == false) {
                                                                          słuchaczy zdarzeń kliknięcia
           if (classesList.getSelectedIndex() == -1) {
                                                                          przycisku, powołania nowej
               System.out.println("Nie zaznaczono zadnej klasy");
                                                                          instancji klasy oraz
           } else {
               Class clazz = (Class)classesList.getSelectedValue();
                                                                          zaznaczenia elementu na
               System.out.println("Zaznaczono klase " + clazz.toString());
               clc.loadConstructors(clazz);
                                                                          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;
                                                       wywołuje w modelu ładowanie klas z podanej
                                                       ścieżki, notyfikuje słuchaczy o załadowaniu klas,
    public void loadClasses(String path) { 
        try {
                                                       lub o błędzie
            fcl.loadClasses(path);
           fireClassEvent(this, ClassEvent.LOAD LIST, fcl.getLoadedClasses());
        } catch (ClassLoaderException e) {
            fireClassEvent(this, ClassEvent.ERROR, e.getErrMessage());
                                                    wywołuje w modelu ładowanie konstruktorów,
                                                    notyfikuje słuchaczy konstruktorów
    public void loadConstructors(Class clazz) { \( \nabla \)
        fireConstructorEvent(this, fcl.getConstructors(clazz));
```

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

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 errMesage) {
        super();
        this.errMessage = errMesage;
    }
    public String getErrMessage() {return errMessage;}
}
```

Kontroler – ClassLoaderController clc cd.

Metody obsługujące słuchaczy

```
public synchronized void addClassListener( ClassListener 1 ) {
    classListeners.add( 1 );
public synchronized void removeClassListener( ClassListener 1 ) {
    classListeners.remove( 1 );
private synchronized void fireClassEvent(Object source, final int opCode, List<Class> classes) {
    ClassEvent ce = new ClassEvent(source,opCode, classes);
    for(ClassListener 1 : classListeners) 1.classesReceived(ce);
private synchronized void fireClassEvent(Object source, final int opCode, String errMessage) {
   ClassEvent ce = new ClassEvent(source,opCode, errMessage);
    for(ClassListener 1 : classListeners) 1.classesReceived(ce);
public synchronized void addConstructorListener( ConstructorListener 1 ) {
    constructorListeners.add( 1 );
public synchronized void removeConstructorListener( ConstructorListener 1 ) {
    constructorListeners.remove( 1 );
private synchronized void fireConstructorEvent(Object source, List<Constructor> constructors) {
    ConstructorEvent ce = new ConstructorEvent(source, constructors);
    for(ConstructorListener 1 : constructorListeners) 1.constructorsReceived(ce);
public synchronized void addObjectListener( ObjectListener 1 ) {
    objectListeners.add( 1 );
public synchronized void removeObjectListener( ObjectListener 1 ) {
    objectListeners.remove( 1 );
private synchronized void fireObjectEvent(Object source, Object instance) {
    ObjectEvent oe = new ObjectEvent(source, instance);
    for(ObjectListener 1 : objectListeners) 1.objectReceived(oe);
private synchronized void fireObjectEvent(Object source, String errorMessage) {
   ObjectEvent oe = new ObjectEvent(source, errorMessage);
    for(ObjectListener 1 : objectListeners) 1.objectReceived(oe);
```

Model - FolderClassLoaderSingleton fcl

```
private void listFilesForFolder(final File folder, String prefix) {
                                                                          Metoda przechodząca
   for (final File fileEntry : folder.listFiles()) {
       if (fileEntry.isDirectory()) {
           listFilesForFolder(fileEntry, prefix + fileEntry.getName() + "."); rekurencyjnie po folderach
           System.out.println("Nazwa folderu " + fileEntry.getName());
                                                                          podanej ścieżki, automatycznie
       } else if(fileEntry.getName().endsWith(".class")) {
                                                                          tworzy przedrostki z nazwami
           String entryName = prefix + fileEntry.getName();
           System.out.println("Odnaleziono " + entryName);
                                                                          pakietów dla znajdowanych klas
           classNames.add(entryName);
public void loadClasses(String path) throws ClassLoaderException {
   classes.clear();
                                                                         Metoda powołująca instancję
   classNames.clear();
                                                                          UMLClassLoader'a, ładuje
   try {
       final File folder = new File(path);
                                                                         znalezione klasy,
       listFilesForFolder(folder, "");
       URL[] urls = {folder.toURI().toURL()};
                                                                         w przypadku niepowodzenia
       URLClassLoader cl = URLClassLoader.newInstance(urls);
       for (String className : classNames) {
                                                                         wyrzuca spersonalizowany
           className = className.substring(0, className.length() - 6);
           Class c = cl.loadClass(className);
                                                                         komunikat
           classes.add(c);
           System.out.println("Załadowano klase " + className);
     catch (MalformedURLException | ClassNotFoundException | NullPointerException e) {
       Logger.getGlobal().severe(e.getMessage());
       throw new ClassLoaderException ("Nie udało się załadować plików z podanej ścieżki!");
```

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 ClassLoaderException{
    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++) {</pre>
        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 ClassLoaderException ("Nie udało się utworzyć obiektu!\nBłędny format parametrów.");
    constructor.setAccessible(true);
    try {
        newObject = constructor.newInstance(objs);
    } catch(Exception el) {
        Logger.getGlobal().severe(el.getMessage());
        throw new ClassLoaderException("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 nasłuchiwania 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 errMessage;
                                                          Metoda toString tej klasy zwraca
   private Object instance;
                                                          sformatowany tekst, który jest gotowy do
   public ObjectEvent(Object source, Object instance) {
                                                          wyświetlenia w polu tekstowym widoku
       super(source);
      this.instance = instance;
                                                          z opisem powołanego obiektu, jeśli wystąpił
   public ObjectEvent(Object source, String errMessage) {
                                                          błąd w wywołaniu obiektu, zwraca ona treść
       super(source);
       this.instance = null;
                                                          komunikatu o błędzie.
       this.errMessage = errMessage;
   public Object getInstance() {return instance;}
   public String toString(){
       if(instance == null) return errMessage;
           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();
```