# Programowanie II

Dariusz Zbyrad

# Plan bloku

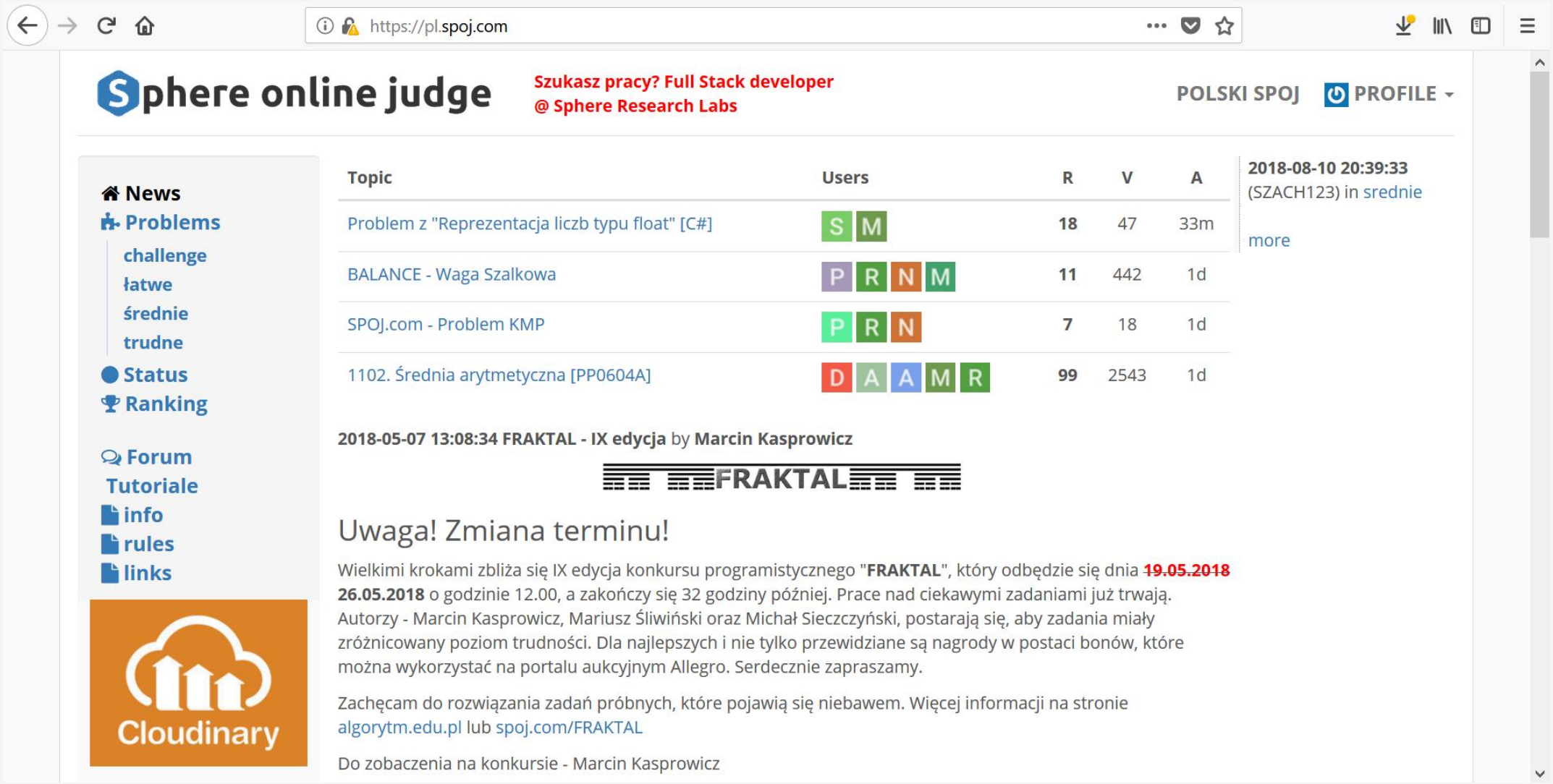
1. Powtórka z bloku „Programowanie I”
2. UML
3. Działanie na plikach
4. Obsługa czasu
5. Logowanie informacji o pracy programu
6. Formaty plików do przechowywania danych (XML, JSON)
7. GUI (JavaFX)
8. Wielowątkowość
9. Dobre praktyki
10. Przykłady bibliotek „na czasie”
11. Maven
12. Wspólnie rozwijany projekt

Autor: Dariusz Zbyrad

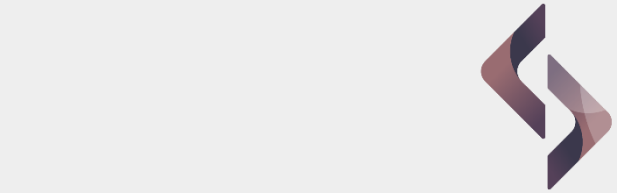
# Czas na powtórkę

Autor: Dariusz Zbyrad

# SPOJ – ćwicz myślenie algorytmiczne



# Przykładowy szablon rozwiązania



```java
public class Main {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        List<String> inputLines = new ArrayList();

        while (scanner.hasNextLine()) {
            inputLines.add(scanner.nextLine());
        }

        String result = executeTask(inputLines);

        System.out.println(result);

    }

    private static String executeTask(List<String> inputLines) {
        return "";
    }

}
```

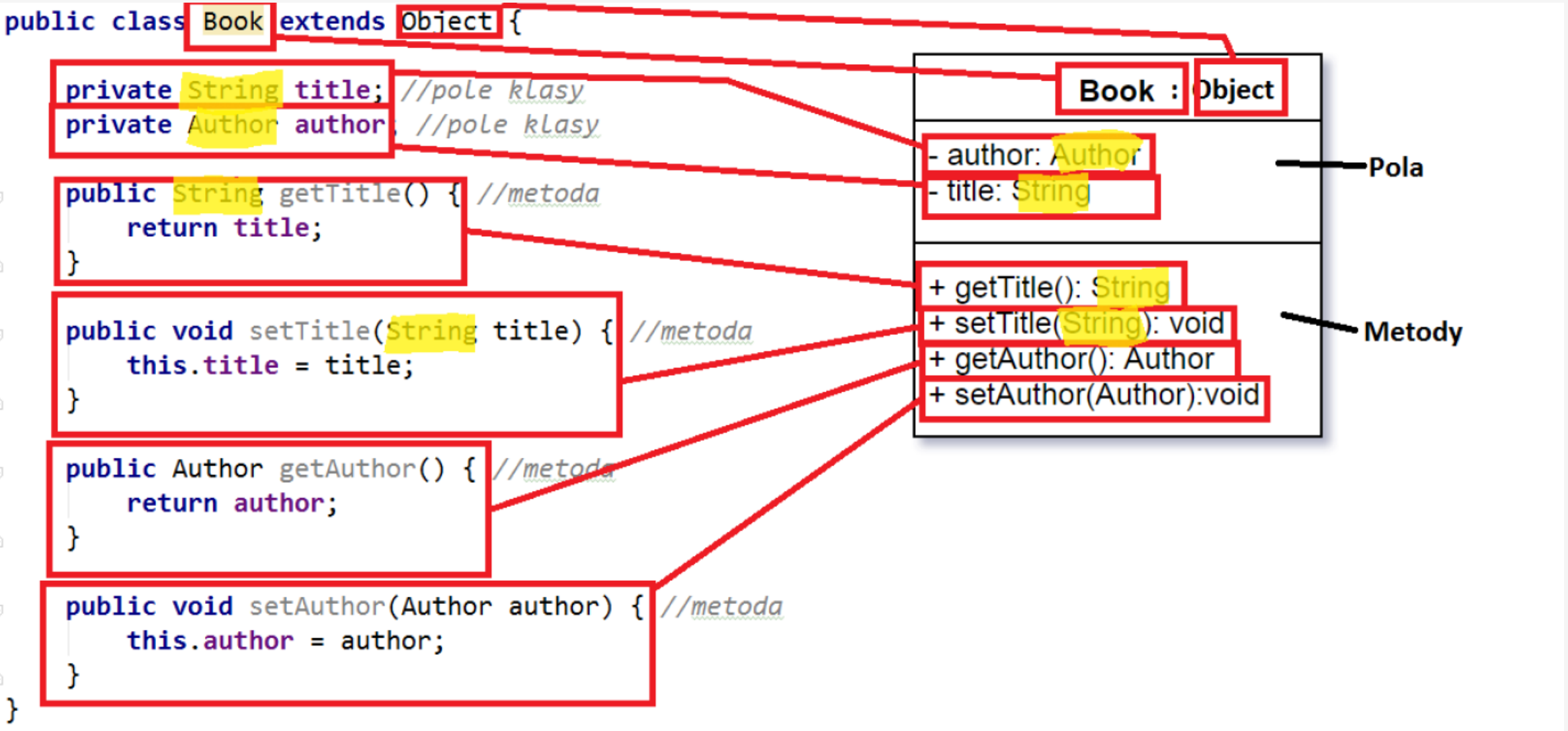
Autor: Dariusz Zbyrad

# Pierwsze zadanie

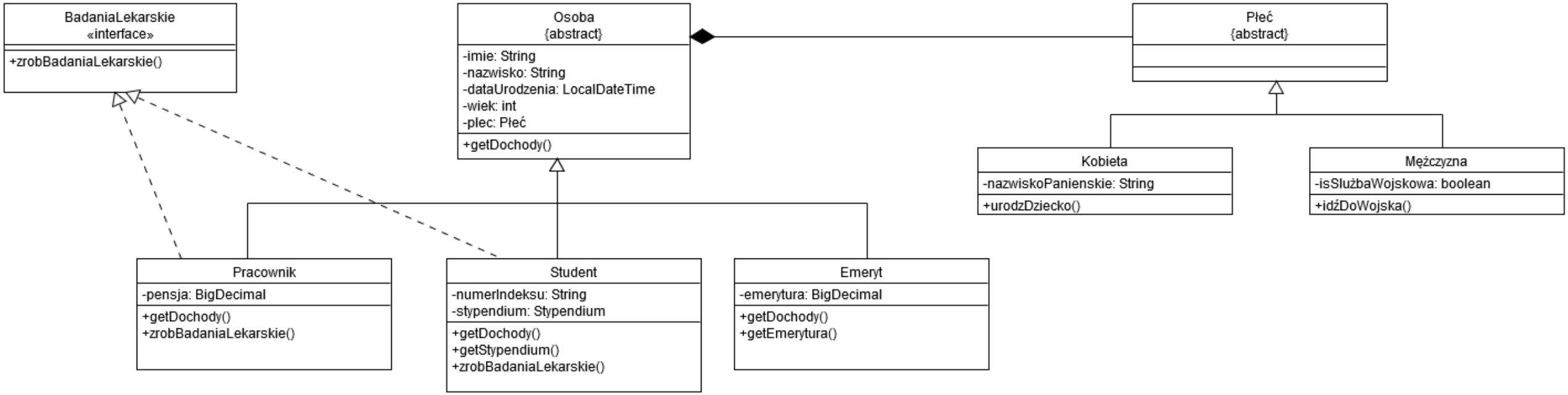
1. Załóż konto na SPOJ
2. Otwórz zadanie „**PTEST - Zadanie próbne**”
3. Stwórz rozwiązanie w oparciu o TDD
4. Wyślij rozwiązanie do SPOJ

Autor: Dariusz Zbyrad

# UML - cd



# UML – diagram klas (dziedziczenie, kompozycja, implementacja)




Autor: Dariusz Zbyrad

# Działanie na plikach

## Lista plików

```
Path path = Paths.get( first: "...");
Files.list(path);
```

## Informacje o pliku

```
Path path = Paths.get( first: "...");
BasicFileAttributes attr = Files.readAttributes(path, BasicFileAttributes.class);

System.out.println("creationTime: " + attr.creationTime());
System.out.println("lastAccessTime: " + attr.lastAccessTime());
System.out.println("lastModifiedTime: " + attr.lastModifiedTime());

System.out.println("isDirectory: " + attr.isDirectory());
System.out.println("isOther: " + attr.isOther());
System.out.println("isRegularFile: " + attr.isRegularFile());
System.out.println("isSymbolicLink: " + attr.isSymbolicLink());
System.out.println("size: " + attr.size());
```

Autor: Dariusz Zbyrad

# Działanie na plikach cd.

## Odczyt pliku tekstowego

```
Path path = Paths.get( first: "...");
Files.lines(path);
```

## Zapis do pliku

```
Arrays.asList("Line 1", "Line2");
Path path = Paths.get( first: "...");
Files.write(path, content);
```

## Szukanie plików rekurencyjnie

```
Path path = Paths.get( first: "...");
int maxDepth = 2;
Files.walk(path, maxDepth);
```

# Zadanie

Znajdź wszystkie pliki w dowolnym katalogu na dysku (oraz jego podkatalogach) z rozszerzeniem txt, następnie odczytaj pierwszą linię każdego z nich i zapisz do nowego pliku txt w formacie:

NAZWA_PLIKU – PIERWSZA_LINIA_Z_PLIKU

Autor: Dariusz Zbyrad

# Data/Czas

```
//Old way
Calendar cal = Calendar.getInstance();
cal.set(Calendar.HOUR, cal.get(Calendar.HOUR) + 2);

//New way
LocalTime now = LocalTime.now();
LocalTime later = now.plus( amountToAdd: 2, ChronoUnit.HOURS);
```

# Zadania

1. Napisz metodę, która dla podanej daty sprawdzi czy jest to piątek trzynastego.
2. Napisz metodę, która dla podanego roku wyświetli listę miesięcy wraz z ich długością (ilością dni).
3. Napisz metodę, która dla zadanego miesiąca z bieżącego roku wyświetli wszystkie poniedziałki.

# Log4j – logowanie działania aplikacji

**Apache log4j** – biblioteka języka programowania Java służąca do tworzenia logów podczas działania aplikacji.

źródło: wikipedia

Podstawowe poziomy logowania informacji:

- DEBUG – informacje pomocne przy debugowaniu aplikacji
- INFO – informacje o działaniu aplikacji (biznesowe)
- WARN – informacje o potencjalnych błędach
- ERROR - informacje o poważnych błędach

# Log4j – przykład użycia

```java
final static Logger logger = Logger.getLogger(GenericDAO.class);

public void save(T input) {
    logger.info("Attempt to save data to: " + path);

    try {
        objectMapper.writeValue(new File(path), input);
    } catch (IOException e) {
        logger.error( message: "Failed to save data to: "+ path, e);
    }
}
```

Autor: Dariusz Zbyrad

# Log4j – przykładowa konfiguracja (log4j.properties)

```
# Root logger option
log4j.rootLogger=DEBUG, stdout, file

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

# Redirect log messages to a log file, support file rolling.
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=log4j-application.log
log4j.appender.file.MaxFileSize=5MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

Autor: Dariusz Zbyrad

# Formaty plików wymiany danych

## XML

```xml
<people>
    <person>
        <name>Janek</name>
        <age>20</age>
    </person>
    <person>
        <name>Agata</name>
        <age>23</age>
    </person>
</people>
```

## JSON

```json
{
  "people": {
    "person": [
      {
        "name": "Janek",
        "age": "20"
      },
      {
        "name": "Agata",
        "age": "23"
      }
    ]
  }
}
```

# Przykład odwzorowania pliku JSON na obiekty

```
public class ReadWriteJackson {
  public static void main(String[] args) throws IOException {
    ObjectMapper mapper = new ObjectMapper();

    String jsonInput = "{\"id\":0,\"firstName\":\"Robin\",\"lastName\":\"Wilson\"}";
    Person q = mapper.readValue(jsonInput, Person.class);
    System.out.println("Read and parsed Person from JSON: " + q);

    Person p = new Person("Roger", "Rabbit");
    System.out.print("Person object " + p + " as JSON = ");
    mapper.writeValue(System.out, p);
  }
}
```

Źródło: https://en.wikipedia.org/wiki/Jackson_(API)

# Zadanie

**Baza danych osób**

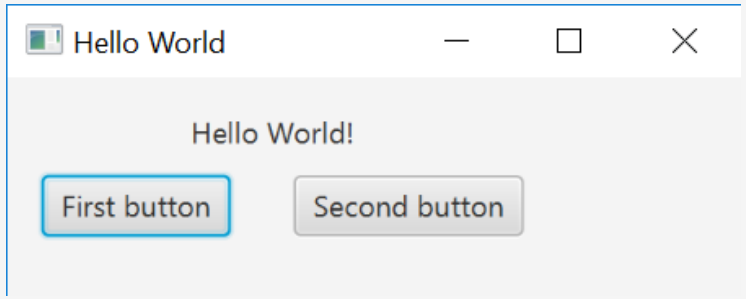
1. Pobierz od użytkownika imię, nazwisko oraz pesel
2. Sprawdź czy pesel jest poprawny, jeżeli nie, wyświetl komunikat i poproś o wpisanie ponownie
3. Zapisz osobę do bazy w postaci pliku XML
4. Zapytaj użytkownika czy chce wprowadzić kolejną osobę, jeżeli tak, idź do punktu 1., jeżeli nie to zakończ

# GUI - JavaFX

```java
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = FXMLLoader.load(getClass().getResource( name: "sample.fxml"));
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(new Scene(root,  width: 300,  height: 275));
        primaryStage.show();
    }


    public static void main(String[] args) { launch(args); }
}
```



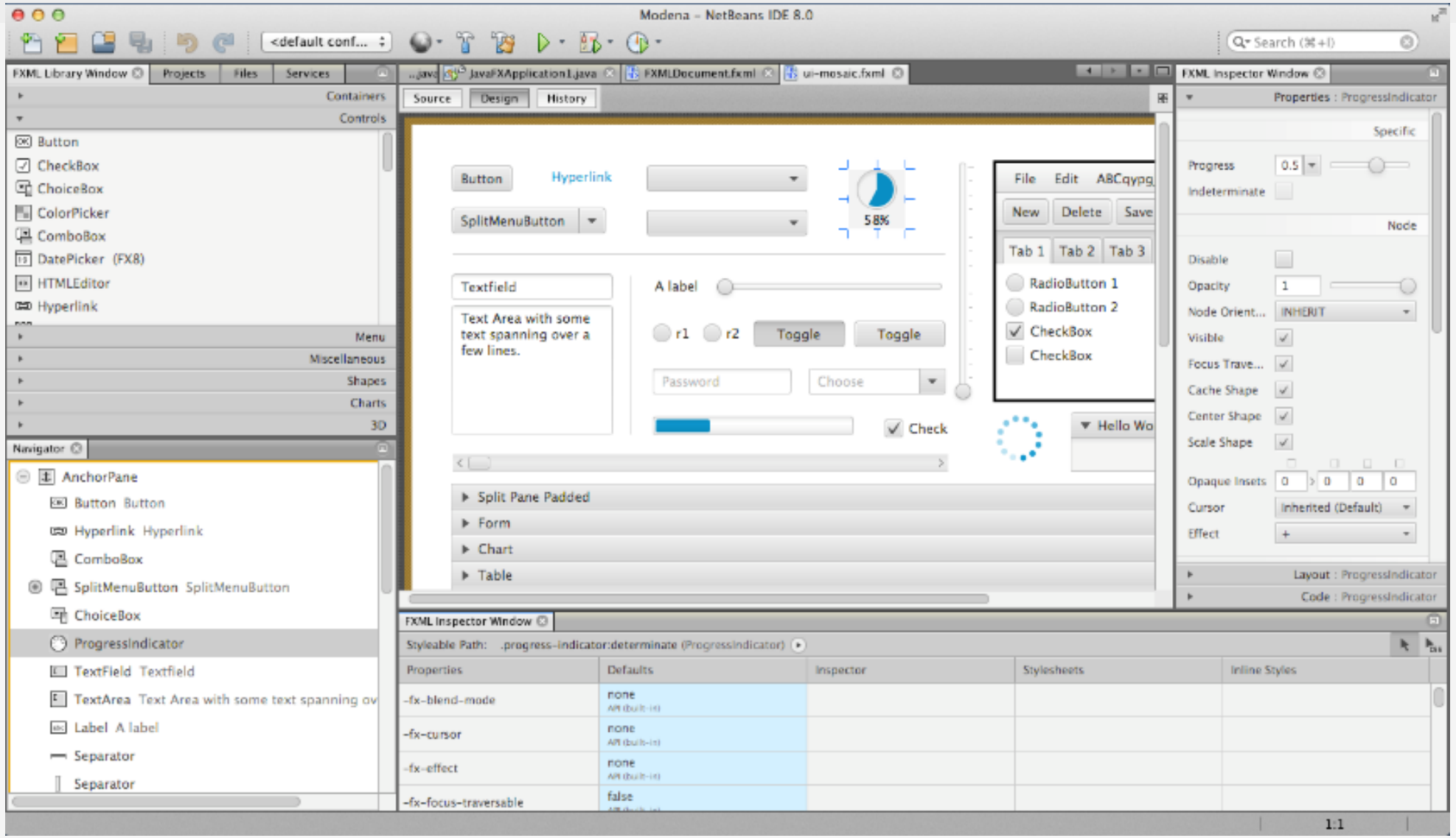

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.Pane?>


<Pane fx:controller="sample.Controller" maxHeight="-Infinity" maxWidth="-Infinity"
      minHeight="-Infinity" minWidth="-Infinity" prefHeight="89.0" prefWidth="216.0"
      xmlns="http://javafx.com/javafx/10.0.1" xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Button fx:id="firstButton" layoutX="14.0" layoutY="40.0" mnemonicParsing="false"
              onMouseClicked="#onClickFirstButton" text="First button" />
      <Button fx:id="secondButton" layoutX="117.0" layoutY="40.0" mnemonicParsing="false"
              onMouseClicked="#onClickSecondButton" text="Second button" />
      <Label layoutX="75.0" layoutY="14.0" text="Hello World!" />
   </children>
</Pane>
```

Autor: Dariusz Zbyrad

# JavaFX - SceneBuilder



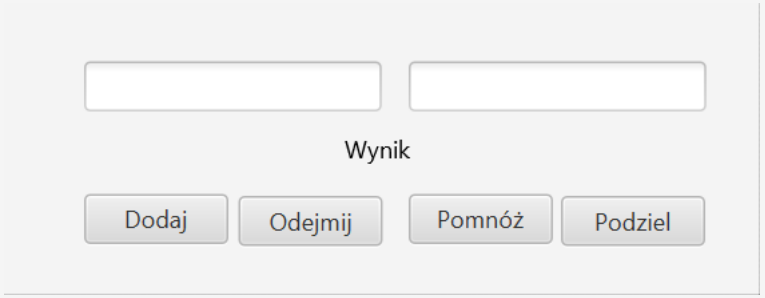

Źródło: http://plugins.netbeans.org/plugin/55434/monet-the-javafx-scene-builder-integration

# Przykładowa aplikacja

Stwórz prostą aplikację graficzną – kalkulator, który będzie umożliwiał dodawanie, odejmowanie, dzielenie i mnożenie dwóch liczb podanych przez użytkownika.

Autor: Dariusz Zbyrad

# Przykładowy design




Autor: Dariusz Zbyrad

# Wielowątkowość – tworzenie nowego wątku

```java
public class MyThreads {

    public static void main(String[] args) {
        Thread thread = new Thread(new Async1());
        thread.start();
    }

    private static class Async1 implements Runnable {

        @Override
        public void run() {
            for (int i = 0; i<10; i++) {
                System.out.println("Hello World from new Thread!");
            }
        }
    }
}
```

```java
public class MyThreads {

    public static void main(String[] args) {
        Thread thread = new Async1();
        thread.start();
    }

    private static class Async1 extends Thread {

        public void run() {
            for (int i = 0; i<10; i++) {
                System.out.println("Hello World from new Thread!");
            }
        }
    }
}
```

Autor: Dariusz Zbyrad

# Wielowątkowość – tworzenie nowego wątku, cd

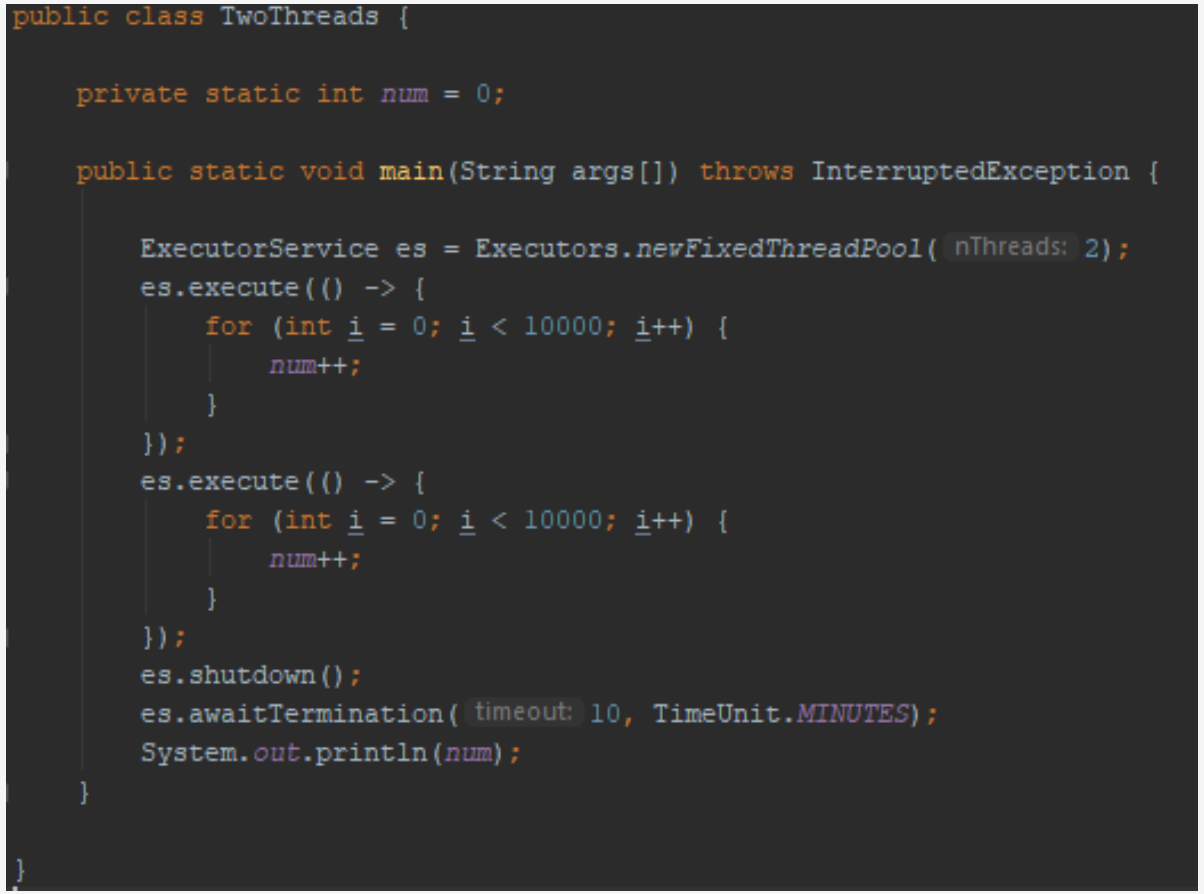
```
public class MyThreads {

    public static void main(String[] args) {
        ExecutorService es = Executors.newFixedThreadPool( nThreads: 1);
        es.execute(() -> {
            for (int i = 0; i<10; i++) {
                System.out.println("Hello World from new Thread!");
            }
        });
    }

}
```

Autor: Dariusz Zbyrad

# Wielowątkowość – problem współdzielenia zasobów



```
public class TwoThreads {

    private static int num = 0;

    public static void main(String args[]) throws InterruptedException {

        ExecutorService es = Executors.newFixedThreadPool( nThreads: 2);
        es.execute(() -> {
            for (int i = 0; i < 10000; i++) {
                num++;
            }
        });
        es.execute(() -> {
            for (int i = 0; i < 10000; i++) {
                num++;
            }
        });
        es.shutdown();
        es.awaitTermination( timeout: 10, TimeUnit.MINUTES);
        System.out.println(num);
    }

}
```

- Jaki będzie wynik działania programu?
- Dlaczego jest on niedeterministyczny?
- Jak to naprawić?

Autor: Dariusz Zbyrad

# Dobre praktyki

**SOLID** to akronim od:
**S** – Single-responsiblity principle
**O** – Open-closed principle
**L** – Liskov substitution principle
**I** – Interface segregation principle
**D** – Dependency Inversion Principle

**DRY** – *Don't Repeat Yourself*

**KISS** - *Keep It Simple, Stupid*

**YAGNI** - You Aren’t Gonna Need It.

# Dobre praktyki - cd

1. Przede wszystkim spójność
2. Trzymaj się ustalonej architektury
3. Dostosuj się do reguł w zespole
4. Korzystaj z JavaDocs
5. Nie odkrywaj na nowo koła
6. Nie twórz długo technicznego
7. Korzystaj ze statycznej analizy kodu
8. Sprawdzaj pokrycie kodu testami

# Biblioteki warte poznania - lombok

Project Lombok jest biblioteką Java, która automatycznie podłącza się do edytora i buduje narzędzia, poprawiając jakość Java.
Biblioteka umożliwia dynamicznie tworzenie takich metod jak:

- setter
- getter
- equals
- hashCode
- toString

I wiele, wiele innych rzecz…

https://projectlombok.org

# Biblioteki warte poznania - StringUtils

**StringUtils** to biblioteka zawierająca ponad 50 metod ułatwiających pracę ze stringami, np.:

- sprawdzanie czy string jest pusty
- czyszczenie
- porównywanie
- Przeszukiwanie

W skrócie pozwala zastąpić złożenie wielu metod z biblioteki standardowej pojedynczą metodą.

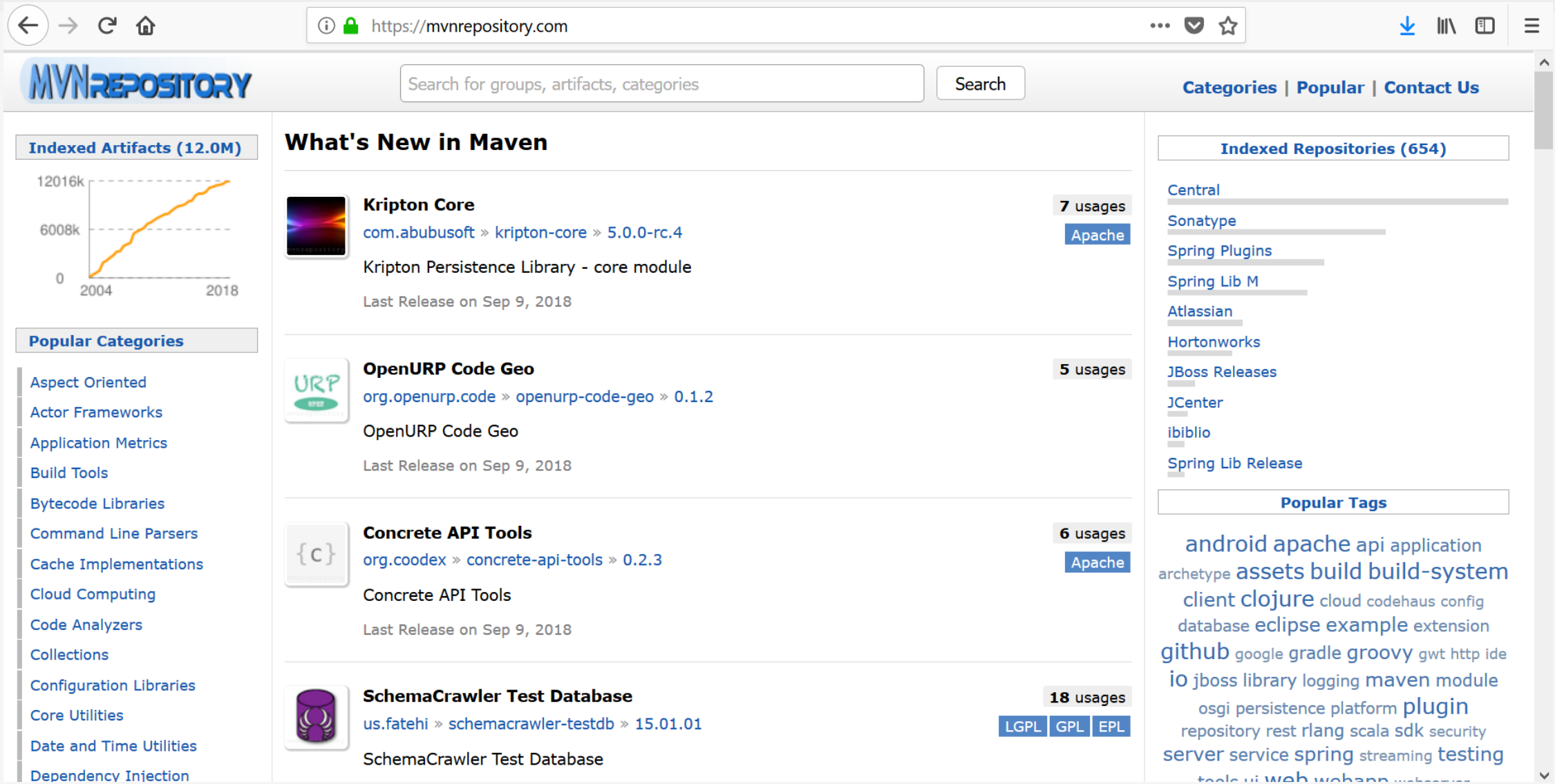
# Maven

**Apache Maven** - narzędzie automatyzujące budowę oprogramowania na platformę Java. Poszczególne funkcje Mavena realizowane są poprzez wtyczki, które są automatycznie pobierane przy ich pierwszym wykorzystaniu. Plik określający sposób budowy aplikacji nosi nazwę POM-u (ang. *Project Object Model*)

Źródło: Wikipedia

Autor: Dariusz Zbyrad

# Repozytorium Maven



# Maven – cykl życia projektu

**validate** - sprawdzenie, czy projekt jest poprawny i czy wszystkie niezbędne informacje zostały określone

**compile** - kod źródłowy jest kompilowany

**test** - przeprowadzane są testy jednostkowe

**package** - budowana jest paczka dystrybucyjna

**integration-test** - zbudowany projekt umieszczany jest w środowisku testowym, gdzie przeprowadzane są testy integracyjne

**verify** - sprawdzenie, czy paczka jest poprawna

**install** - paczka umieszczana jest w repozytorium lokalnym - może być używana przez inne projekty jako zależność

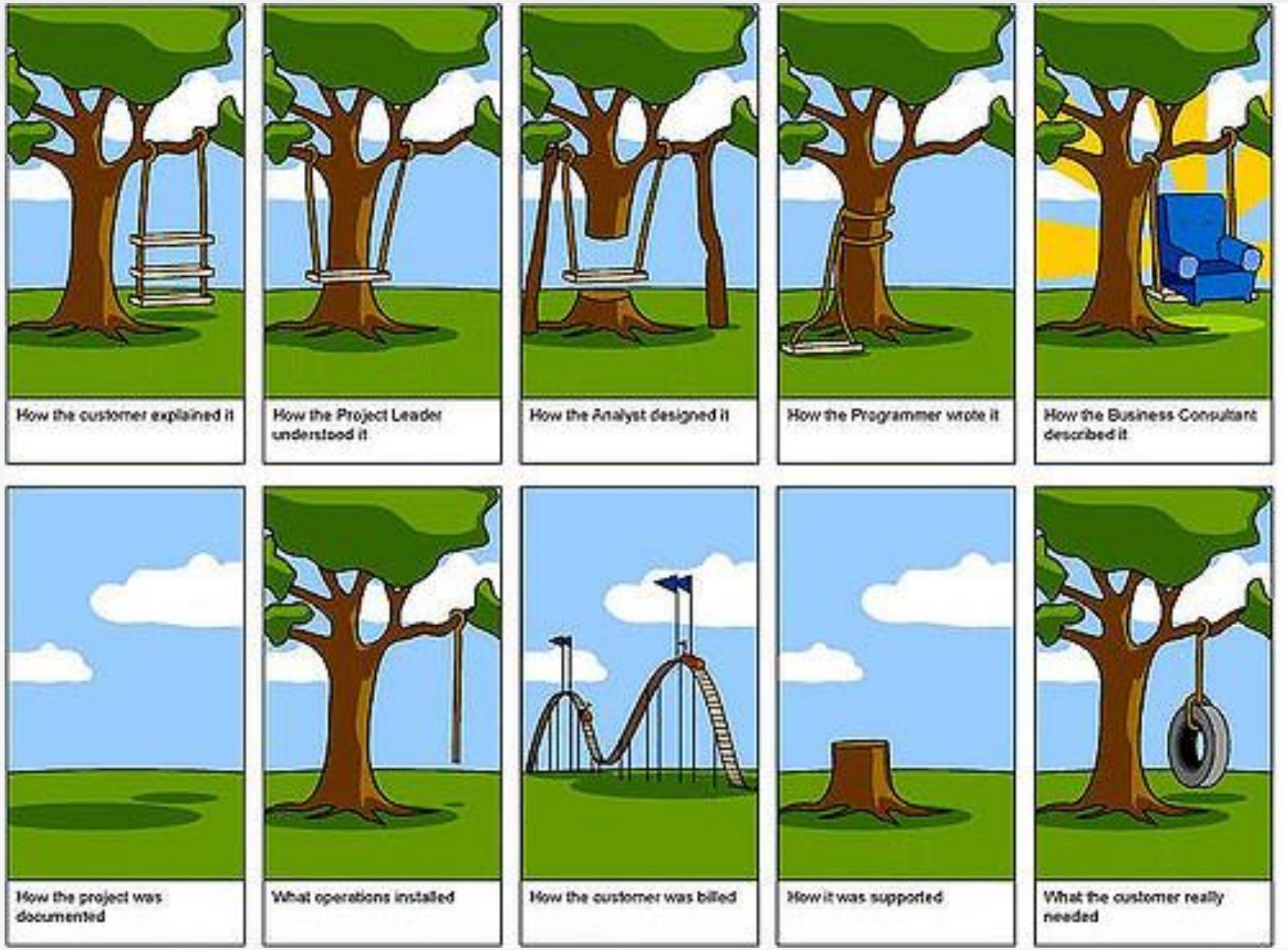
**deploy** - paczka umieszczana jest w repozytorium zdalnym (opublikowana)

# Czas na własny projekt

## Cykl życia oprogramowania:

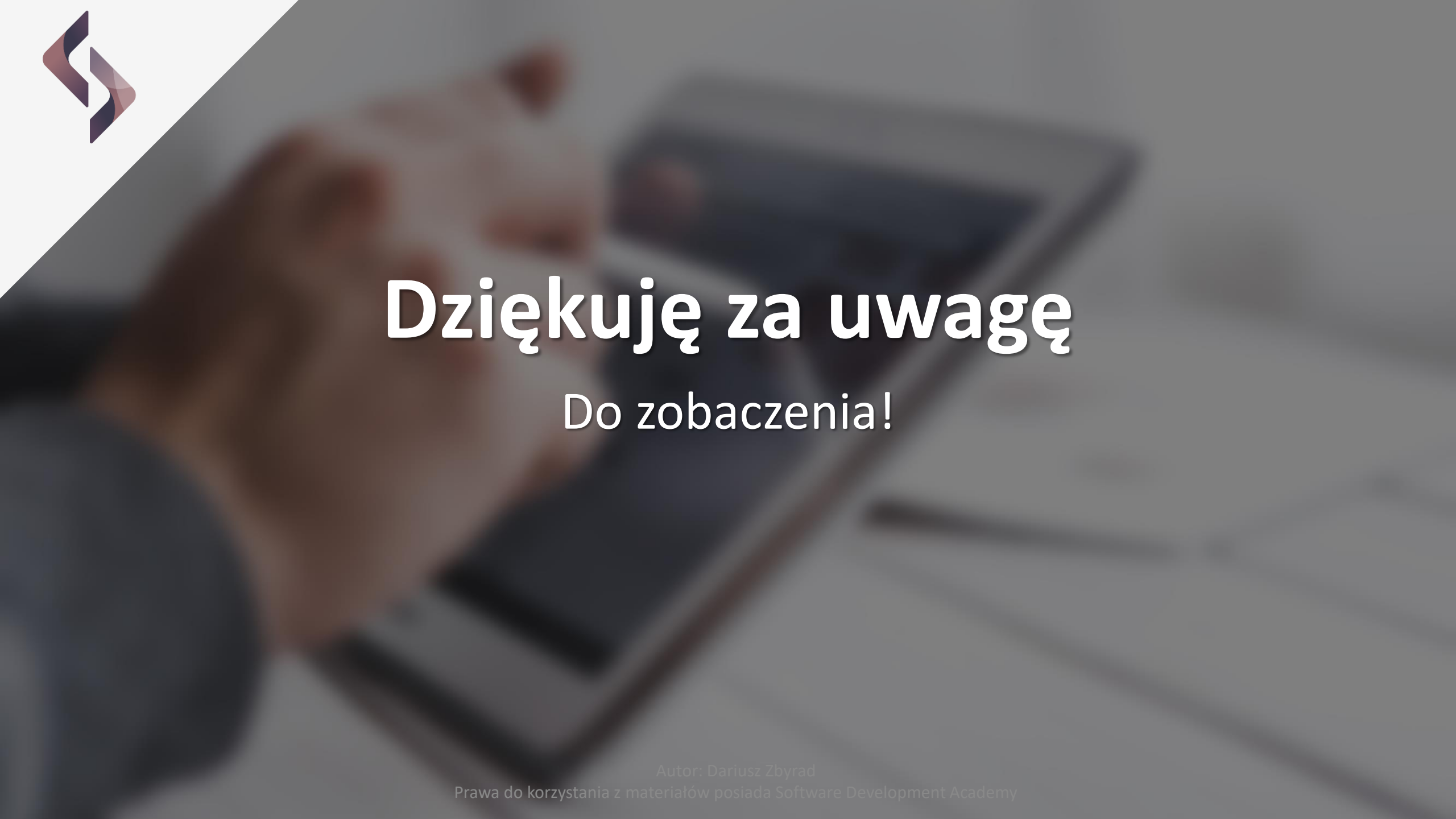
1. Określanie wymagań i specyfikacji
2. Projektowanie
3. Implementacja
4. Testowanie – walidacja (atestowanie) i weryfikacja
5. Konserwacja (pielęgnacja)

Autor: Dariusz Zbyrad






Źródło: https://www.flickr.com/photos/hoalit/355398491/



# Dziękuję za uwagę

Do zobaczenia!