Faculty of Electronics and Information Technology
WARSAW UNIVERSITY OF TECHNOLOGY

Programowanie Aplikacyjne (PAP)

JPA

Julian Myrcha
Institute of Computer Science
26.02.2025

- pisanie klas odpowiadających danym jest nużące

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola

```
1 public class Person {
2       private String firstName;
3       private String lastName;
4       ...
5 }
```

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)

```java
public class Person {
    private String firstName;
    private String lastName;

    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return "Person [firstName=" + firstName + ", lastName=" + lastName + "]";
    }
```

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)

```
22      @Override
23      public int hashCode() {
24          final int prime = 31;
25          int result = 1;
26          result = prime * result + ((firstName == null) ? 0 : firstName.hashCode());
27          result = prime * result + ((lastName == null) ? 0 : lastName.hashCode());
28          return result;
29      }
```

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)

```
30      @Override
31      public boolean equals(Object obj) {
32          if (this == obj)
33              return true;
34          if (obj == null)
35              return false;
36          if (getClass() != obj.getClass())
37              return false;
38          Person other = (Person) obj;
39          if (firstName == null) {
40              if (other.firstName != null)
41                  return false;
42          } else if (!firstName.equals(other.firstName))
43              return false;
44          if (lastName == null) {
45              if (other.lastName != null)
46                  return false;
47          } else if (!lastName.equals(other.lastName))
48              return false;
49          return true;
50      }
51  }
```

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)
- za pomocą adnotacji i biblioteki `lombok` możemy wymusić generację

```
1 import lombok.Data;
2
3 @Data
4 public class Person {
5    private String firstName;
6    private String lastName;
7 }
```

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)
- za pomocą adnotacji i biblioteki `lombok` możemy wymusić generację

  @AllArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)
- za pomocą adnotacji i biblioteki `lombok` możemy wymusić generację

  @AllArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola

  @RequiredArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola zaznaczone jako `@NotNull` oraz final

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)
- za pomocą adnotacji i biblioteki `lombok` możemy wymusić generację

  @AllArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola

  @RequiredArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola zaznaczone jako `@NotNull` oraz final

  @ToString - wygeneruje metodę `toString`

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)
- za pomocą adnotacji i biblioteki `lombok` możemy wymusić generację

  @AllArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola

  @RequiredArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola zaznaczone jako `@NotNull` oraz final

  @ToString - wygeneruje metodę `toString`

  @EqualsAndHashCode - wygeneruje metody `equals` oraz `hashCode`

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)
- za pomocą adnotacji i biblioteki `lombok` możemy wymusić generację

  @AllArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola

  @RequiredArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola zaznaczone jako `@NotNull` oraz final

  @ToString - wygeneruje metodę `toString`

  @EqualsAndHashCode - wygeneruje metody `equals` oraz `hashCode`

  @Getter @Setter - generuje gettery i settery na tak oznaczonych polach

- pisanie klas odpowiadających danym jest nużące
  - mamy klasę przechowującą 2 pola
  - musimy napisać 60 linii kodu (gettery/settery, toString, hashCode, equals)
- za pomocą adnotacji i biblioteki `lombok` możemy wymusić generację

  @AllArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola

  @RequiredArgsConstructor - wygeneruje nam konstruktor ustawiający wszystkie pola zaznaczone jako `@NotNull` oraz final

  @ToString - wygeneruje metodę `toString`

  @EqualsAndHashCode - wygeneruje metody `equals` oraz `hashCode`

  @Getter @Setter - generuje gettery i settery na tak oznaczonych polach

  @Data - `@ToString`+`@EqualsAndHashCode`+ `@Getter` (na wszystkich polach)+ `@Setter` (na wszystkich polach) + `@RequiredArgsConstructor`

- Ochronę zasobów możemy zrealizować wykorzystując konstrukcje Javy:

```
1  package pap;
2
3  class Resource implements AutoCloseable {
4    public void close() {
5      System.out.println("closed: " + this);
6    }
7  }
8  public class BookTest {
9    public static void main(String[] args) {
10     try (var r2 = new Resource();
11          var r1 = new Resource()) {
12       System.out.println("computed: " + r1);
13       System.out.println("computed: " + r2);
14     }
15   }
16 }
```

- Ochronę zasobów możemy zrealizować wykorzystując konstrukcje Javy:
- Lub adnotację `@Cleanup` z biblioteki lombook

```
1  package pap;
2  import lombok.Cleanup;
3  class Resource implements AutoCloseable {
4    public void close() {
5      System.out.println("closed: " + this);
6    }
7  }
8  public class Test {
9    public static void main(String[] args) {
10     @Cleanup var r2 = new Resource();
11     @Cleanup var r1 = new Resource();
12     System.out.println("computed: " + r1);
13     System.out.println("computed: " + r2);
14   }
15 }
16
```

- Ochronę zasobów możemy zrealizować wykorzystując konstrukcje Javy:
- Lub adnotację `@Cleanup` z biblioteki lombook
- wynik działania programu nie ulega zmianie:

```
computed: pap.Resource@6ff3c5b5
computed: pap.Resource@3764951d
closed: pap.Resource@6ff3c5b5
closed: pap.Resource@3764951d
```

- dodanie bibliotek za pomocą mavena

```xml
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.16.20</version>
</dependency>
```

- dodanie bibliotek za pomocą mavena

```xml
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.16.20</version>
</dependency>
```

- dodanie pluginu obsługującego kompilację do VS Code

```
https://marketplace.visualstudio.com/items?itemName=GabrielBB.vscode-lombok
Gabriel Basilio Brito
A lightweight extension to support Lombok annotations processing in Visual Studio Code
```

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.
  - EclipseLink (implementacja referencyjna)

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.
  - EclipseLink (implementacja referencyjna)
  - Hibernate (najbardziej popularna)

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.
  - EclipseLink (implementacja referencyjna)
  - Hibernate (najbardziej popularna)
  - Apache OpenJPA

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.
  - EclipseLink (implementacja referencyjna)
  - Hibernate (najbardziej popularna)
  - Apache OpenJPA
- JPA pozwala programiście pracować bezpośrednio z obiektami, a nie z instrukcjami SQL

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.
  - EclipseLink (implementacja referencyjna)
  - Hibernate (najbardziej popularna)
  - Apache OpenJPA
- JPA pozwala programiście pracować bezpośrednio z obiektami, a nie z instrukcjami SQL
  - Implementacja JPA jest zwykle nazywana persistence provider

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.
  - EclipseLink (implementacja referencyjna)
  - Hibernate (najbardziej popularna)
  - Apache OpenJPA
- JPA pozwala programiście pracować bezpośrednio z obiektami, a nie z instrukcjami SQL
  - Implementacja JPA jest zwykle nazywana persistence provider
  - Odwzorowanie między obiektami Java a tabelami bazy danych jest definiowane za pomocą metadanych

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.
  - EclipseLink (implementacja referencyjna)
  - Hibernate (najbardziej popularna)
  - Apache OpenJPA
- JPA pozwala programiście pracować bezpośrednio z obiektami, a nie z instrukcjami SQL
  - Implementacja JPA jest zwykle nazywana persistence provider
  - Odwzorowanie między obiektami Java a tabelami bazy danych jest definiowane za pomocą metadanych
  - Metadane JPA są zwykle definiowane za pomocą adnotacji w klasie Java

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.
  - EclipseLink (implementacja referencyjna)
  - Hibernate (najbardziej popularna)
  - Apache OpenJPA
- JPA pozwala programiście pracować bezpośrednio z obiektami, a nie z instrukcjami SQL
  - Implementacja JPA jest zwykle nazywana persistence provider
  - Odwzorowanie między obiektami Java a tabelami bazy danych jest definiowane za pomocą metadanych
  - Metadane JPA są zwykle definiowane za pomocą adnotacji w klasie Java
- JPA definiuje język zapytań podobny do SQL dla zapytań statycznych i dynamicznych.

- Mapowanie obiektów Java do tabel bazy danych i odwrotnie nazywa się mapowaniem obiektowo-relacyjnym (ORM)
- JPA to specyfikacja i dostępnych jest kilka implementacji.
  - EclipseLink (implementacja referencyjna)
  - Hibernate (najbardziej popularna)
  - Apache OpenJPA
- JPA pozwala programiście pracować bezpośrednio z obiektami, a nie z instrukcjami SQL
  - Implementacja JPA jest zwykle nazywana persistence provider
  - Odwzorowanie między obiektami Java a tabelami bazy danych jest definiowane za pomocą metadanych
  - Metadane JPA są zwykle definiowane za pomocą adnotacji w klasie Java
- JPA definiuje język zapytań podobny do SQL dla zapytań statycznych i dynamicznych.
- Większość implementacji JPA oferuje opcję automatycznego tworzenia schematu bazy danych na podstawie metadanych.

EntityManagerFactory - klasa fabryczna EntityManager. Tworzy i zarządza wieloma instancjami EntityManager

EntityManagerFactory - klasa fabryczna EntityManager. Tworzy i zarządza wieloma instancjami EntityManager

EntityManager - interfejs, zarządza operacjami trwałości na obiektach. Działa jak fabryka dla instancji Query

EntityManagerFactory - klasa fabryczna EntityManager. Tworzy i zarządza wieloma instancjami EntityManager

EntityManager - interfejs, zarządza operacjami trwałości na obiektach. Działa jak fabryka dla instancji Query

Entity - obiektami trwałości przechowywanymi jako rekordy w bazie danych

EntityManagerFactory - klasa fabryczna EntityManager. Tworzy i zarządza wieloma instancjami EntityManager

EntityManager - interfejs, zarządza operacjami trwałości na obiektach. Działa jak fabryka dla instancji Query

Entity - obiektami trwałości przechowywanymi jako rekordy w bazie danych

EntityTransaction - jest w relacji jeden do jednego z EntityManager. Dla każdego EntityManager operacje są obsługiwane przez klasę EntityTransaction

EntityManagerFactory - klasa fabryczna EntityManager. Tworzy i zarządza wieloma instancjami EntityManager

EntityManager - interfejs, zarządza operacjami trwałości na obiektach. Działa jak fabryka dla instancji Query

Entity - obiektami trwałości przechowywanymi jako rekordy w bazie danych

EntityTransaction - jest w relacji jeden do jednego z EntityManager. Dla każdego EntityManager operacje są obsługiwane przez klasę EntityTransaction

Persistence - zawiera metody statyczne do uzyskiwania wystąpienia EntityManagerFactory

EntityManagerFactory - klasa fabryczna EntityManager. Tworzy i zarządza wieloma instancjami EntityManager

EntityManager - interfejs, zarządza operacjami trwałości na obiektach. Działa jak fabryka dla instancji Query

Entity - obiektami trwałości przechowywanymi jako rekordy w bazie danych

EntityTransaction - jest w relacji jeden do jednego z EntityManager. Dla każdego EntityManager operacje są obsługiwane przez klasę EntityTransaction

Persistence - zawiera metody statyczne do uzyskiwania wystąpienia EntityManagerFactory

Query - interfejs jest implementowany przez każdego dostawcę JPA w celu uzyskania obiektów relacyjnych, które spełniają kryteria

@ManyToOne - wiele do jednego

```java
@Data       // to z lomboka
@Entity
public class Department {
   @Id @GeneratedValue( strategy=GenerationType.AUTO )
   private int id;
   private String name;
}

@Data
@Entity
public class Employee{
   @Id
   @GeneratedValue( strategy= GenerationType.AUTO )
   private int employeeID;
   private String employeeName;
   private double salary;

   @ManyToOne
   private Department department;
}
```

@ManyToOne - wiele do jednego

@OneToMany - jeden do wielu

```
@Data
@Entity
public class Department {
    @Id @GeneratedValue( strategy=GenerationType.AUTO )
    private int id;
    private String name;

    @OneToMany( targetEntity=Employee.class )
    private List employeelist;
}

@Data
@Entity
public class Employee {
   @Id  @GeneratedValue( strategy= GenerationType.AUTO )

   private int employeeID;
   private String employeeName;
   private double salary;
}
```

@ManyToOne - wiele do jednego

@OneToMany - jeden do wielu

@OneToOne - jeden do jednego

```java
@Data @Entity
public class Department {
   @Id @GeneratedValue( strategy=GenerationType.AUTO )
   private int id;
   private String name;
}

@Data
@Entity
public class Employee {
   @Id  @GeneratedValue( strategy= GenerationType.AUTO )
   private int eid;
   private String employeeName;
   private double salary;

   @OneToOne
   private Department department;
}
```

@ManyToOne - wiele do jednego

@OneToMany - jeden do wielu

@OneToOne - jeden do jednego

@ManyToMany - wiele do wielu (tworzy tablicę pośrednią)

```
@Data @Entity
public class Department {
   @Id @GeneratedValue( strategy=GenerationType.AUTO )
   private int id;
   private String name;

   @ManyToMany(targetEntity=Employee.class)
   private Set employeeSet;

}

@Data @Entity
public class Employee {
   @Id  @GeneratedValue( strategy= GenerationType.AUTO )
   private int eid;
   private String employeeName;
   private double salary;
   @ManyToMany(targetEntity=Department.class)
   private Set departmentSet;
}
Set<Employee> employees = new HashSet();
employees.add(e1);
```

- Baza MySQL

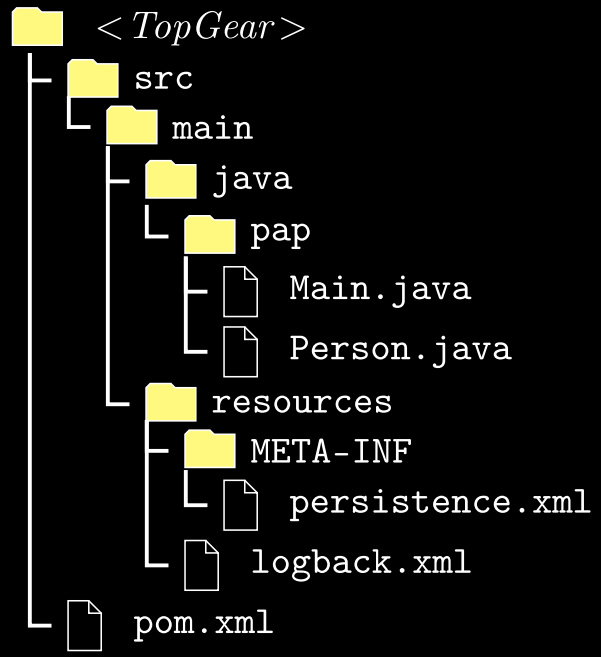
```
CREATE DATABASE IF NOT EXISTS pap;
use pap;
CREATE USER 'pap'@'localhost' IDENTIFIED BY 'pap.2020.PAP';
GRANT ALL ON pap.* TO 'pap'@'localhost';
CREATE TABLE Persons (
    PersonID INT NOT NULL,
    FirstName VARCHAR(30) NOT NULL,
    LastName VARCHAR(30) NOT NULL,
    PRIMARY KEY (PersonID)
);
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderValue float NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
                              REFERENCES Persons(PersonID)
);
```

- Baza MySQL

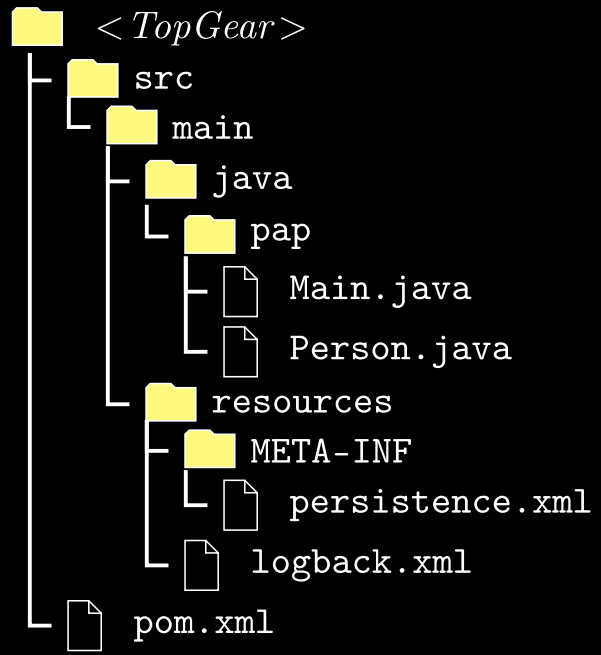
```
mysql> desc Persons;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| PersonID  | int         | NO   | PRI | NULL    |       |
| FirstName | varchar(30) | NO   |     | NULL    |       |
| LastName  | varchar(30) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> desc Orders;
+-------------+-------+------+-----+---------+-------+
| Field       | Type  | Null | Key | Default | Extra |
+-------------+-------+------+-----+---------+-------+
| OrderID     | int   | NO   | PRI | NULL    |       |
| OrderNumber | int   | NO   |     | NULL    |       |
| OrderValue  | float | NO   |     | NULL    |       |
| PersonID    | int   | YES  | MUL | NULL    |       |
+-------------+-------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

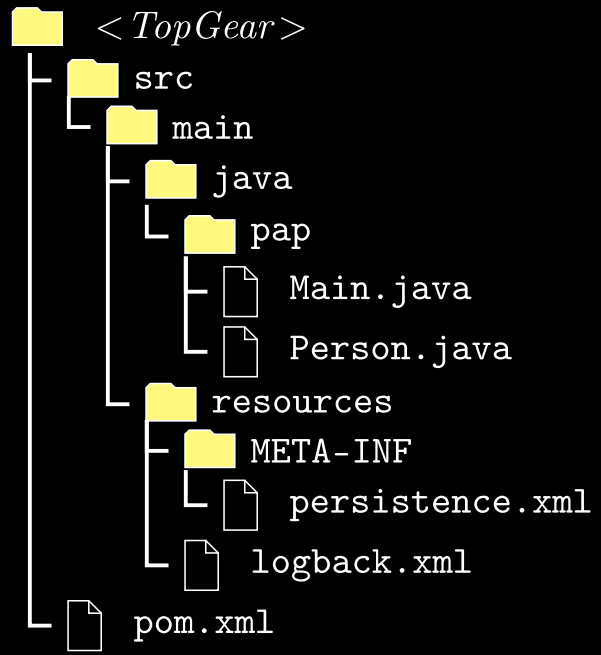
- projekt maven



- projekt maven
- zależności sciągane przez shade



- projekt maven
- zależności sciągane przez shade
- implementacja JPA za pomocą Hibernate 5



```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>edu.pw.ii</groupId>
  <artifactId>TopGear-Hibernated</artifactId>
  <version>1.0.0</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>15</maven.compiler.source>
    <maven.compiler.target>15</maven.compiler.target>
    <exec.mainClass>pap.Main</exec.mainClass>
    <cleanupDaemonThreads>false</cleanupDaemonThreads>
  </properties>

  <dependencies>
    <!-- Logback replaces Log2J to print the logs generated by Hibernate -->
    <dependency>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
      <version>1.1.7</version>
    </dependency>

    <!-- Core of Hibernate -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.2.3.Final</version>
    </dependency>

    <!-- Entity manager which is required for JPA -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-entitymanager</artifactId>
      <version>5.2.3.Final</version>
    </dependency>

    <!-- JDBC Connector for MySQL -->
```

```
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.16</version>
    </dependency>

    <dependency>
      <groupId>javax.xml.bind</groupId>
      <artifactId>jaxb-api</artifactId>
      <version>2.3.0</version>
    </dependency>

  </dependencies>

  <build>
    <plugins>
    <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.2.1</version>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
                        <goal>shade</goal>
                    </goals>
                    <configuration>
                        <transformers>
                            <transformer
    implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                            </transformer>
                        </transformers>
                    </configuration>
                </execution>
            </executions>
        </plugin>
      <!-- This plugin is used to set the Java version 15 -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <source>15</source>
          <target>15</target>
        </configuration>
```

```
        </plugin>
      </plugins>
    </build>
</project>
```

```xml
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
  "http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="PAP" transaction-type="RESOURCE_LOCAL">
    <!-- Persistence provider -->
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <!-- Entity classes -->
    <class>pap.Person</class>

    <properties>
      <!-- The JDBC URL to the database instance -->
      <property name="javax.persistence.jdbc.url"
                    value="jdbc:mysql://localhost:3306/pap" />
      <!-- The database username -->
      <property name="javax.persistence.jdbc.user" value="pap" />
      <!-- The database password -->
    <property name="javax.persistence.jdbc.password" value="pap.2020.PAP"/>

      <property name="hibernate.hbm2ddl.auto" value="update" />
    </properties>
  </persistence-unit>
</persistence>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="false">
  <contextListener class="ch.qos.logback.classic.jul.LevelChangePropagator"/>
  <!-- Format the log output -->
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n
      </pattern>
    </encoder>
  </appender>
  <!-- Set the application log level to INFO -->
  <root level="INFO">
    <appender-ref ref="STDOUT" />
  </root>
  <!-- Set log level of Hibernate to ERROR level -->
  <logger name="org.hibernate">
    <level value="ERROR" />
  </logger>
</configuration>
```

```java
package pap;
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "Persons")
public class Person implements Serializable {
  private static final long serialVersionUID = -3741264748388489528L;
  @Id
  @Column(name = "PersonID", unique = true)
  private int id;

  @Column(name = "FirstName", nullable = false)
  private String firstName;

  @Column(name = "LastName", nullable = false)
  private String lastName;

  public int getId() { return id;  }
  public void setId(int id) {
    this.id = id;
  }

  public String getFirstName() {  return firstName;}
  public void setFirstName(String name) {
    this.firstName = name;
  }

  public String getLastName() { return lastName;  }
  public void setLastName(String name) {
    this.lastName = name;
  }

  @Override
  public String toString() {
    return id + "\t" + firstName + "\t" + lastName;
  }
```

```
41  }
```

```java
package pap;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class Main {
  // Create an EntityManagerFactory when you start the application.
  private static final EntityManagerFactory
     ENTITY_MANAGER_FACTORY = Persistence.createEntityManagerFactory("PAP");
  public static void main(String[] args) {
    deletePersons();
    createPersons();
    updatePerson();
    deletePerson();

    List<Person> persons = readAll();
    if (persons != null) {
      for (Person p : persons)
        System.out.println(p);
    }

    // must be closed!
    ENTITY_MANAGER_FACTORY.close();
  }
  ...
}
```

```java
public static void createPersons() {
  EntityManager manager = ENTITY_MANAGER_FACTORY.createEntityManager();
  EntityTransaction transaction = null;
  try {
    transaction = manager.getTransaction();
    transaction.begin();
    Person jeremy = new Person();
    jeremy.setId(1);
    jeremy.setFirstName("Jeremy");
    jeremy.setLastName("Clarkson");
    manager.persist(jeremy);
    Person james = new Person();
    james.setId(2);
    james.setFirstName("James");
    james.setLastName("May");
    manager.persist(james);
    Person richard = new Person();
    richard.setId(3);
    richard.setFirstName("Richard");
    richard.setLastName("Hammond");
    manager.persist(richard);
    transaction.commit();
  } catch (Exception ex) {
    if (transaction != null)   transaction.rollback();
    ex.printStackTrace();
  } finally {
    manager.close();
  }
}
```

```java
public static List<Person> readAll() {
  List<Person> result = null;
  EntityManager manager = ENTITY_MANAGER_FACTORY.createEntityManager();
  EntityTransaction transaction = null;
  try {
    transaction = manager.getTransaction();
    transaction.begin();
    result = manager.createQuery("SELECT p FROM Person p", Person.class).getResultList();
    transaction.commit();
  } catch (Exception ex) {
    if (transaction != null)
      transaction.rollback();
    ex.printStackTrace();
  } finally {
    manager.close();
  }
  return result;
}
```

```java
public static void deletePerson() {
  EntityManager manager = ENTITY_MANAGER_FACTORY.createEntityManager();
  EntityTransaction transaction = null;
  try {
    transaction = manager.getTransaction();
    transaction.begin();
    Person p = manager.find(Person.class, 3);
    manager.remove(p);
    transaction.commit();
  } catch (Exception ex) {
    if (transaction != null)
      transaction.rollback();
    ex.printStackTrace();
  } finally {
    manager.close();
  }
}
```

```java
public static void deletePersons() {
  EntityManager manager = ENTITY_MANAGER_FACTORY.createEntityManager();
  EntityTransaction transaction = null;
  try {
    transaction = manager.getTransaction();
    transaction.begin();
    Query query = manager.createQuery("DELETE FROM Person p");
    int deleteRecords = query.executeUpdate();
    transaction.commit();
    System.out.println("All records were deleted.");
  } catch (Exception ex) {
    System.out.println(ex.getMessage());
  } finally {
    manager.close();
  }
}
```

```java
public static void updatePerson() {
  EntityManager manager = ENTITY_MANAGER_FACTORY.createEntityManager();
  EntityTransaction transaction = null;
  try {
    transaction = manager.getTransaction();
    transaction.begin();
    Person p = manager.find(Person.class, 2);
    p.setFirstName("JAMES");
    manager.persist(p);
    transaction.commit();
  } catch (Exception ex) {
    if (transaction != null)
      transaction.rollback();
    ex.printStackTrace();
  } finally {
    manager.close();
  }
}
```

Wynik działania programu

```
>mvn clean
>mvn compile
>mvn package
>java -cp target/TopGear-Hibernated-1.0.0.jar pap.Main
All records were deleted.
1       Jeremy  Clarkson
2       JAMES   May
>
```

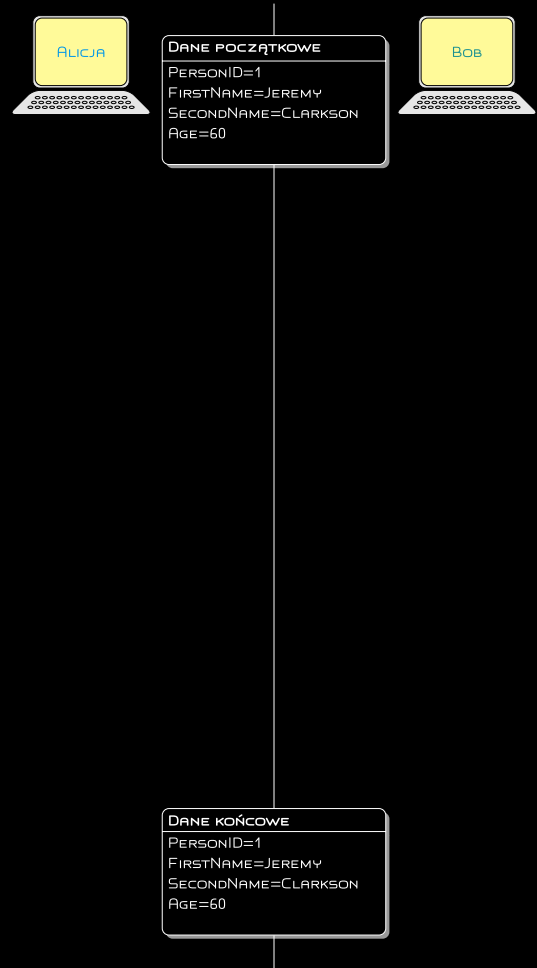
- import bibliotek maven

```xml
<dependency>
  <groupId>org.apache.derby</groupId>
  <artifactId>derby</artifactId>
  <version>10.15.2.0</version>
</dependency>
```

- import bibliotek maven
- plik persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
            http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="PAP" transaction-type="RESOURCE_LOCAL">
    <!-- Persistence provider -->
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <!-- Entity classes -->
    <class>pap.Person</class>
    <properties>
      <!-- The JDBC URL to the database instance -->
      <property name="javax.persistence.jdbc.url"
                      value="jdbc:derby:./data;create=true" />
      <!-- The database username -->
      <property name="javax.persistence.jdbc.user" value="sa" />
      <!-- The database password -->
      <property name="javax.persistence.jdbc.password" value="" />
      <property name="hibernate.hbm2ddl.auto" value="update" />
    </properties>
  </persistence-unit>
</persistence>
```

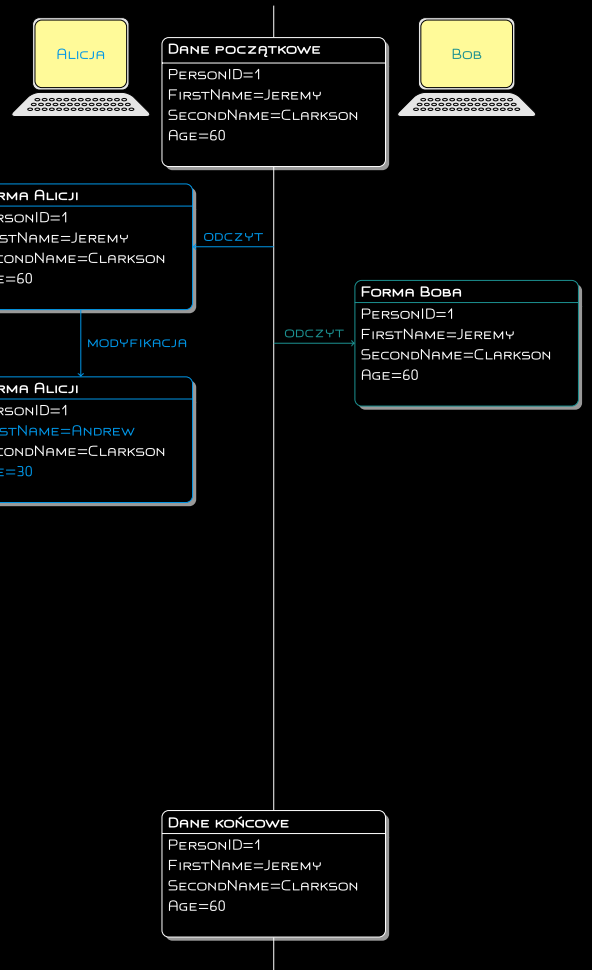
- Alicja i Bob edytują



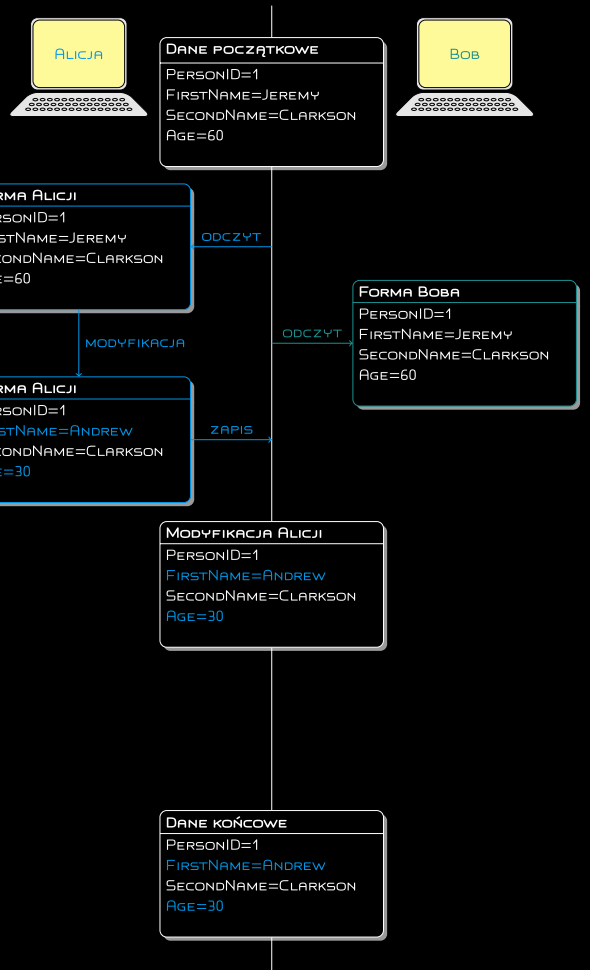
- Alicja i Bob edytują
- Alicja czyta

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta

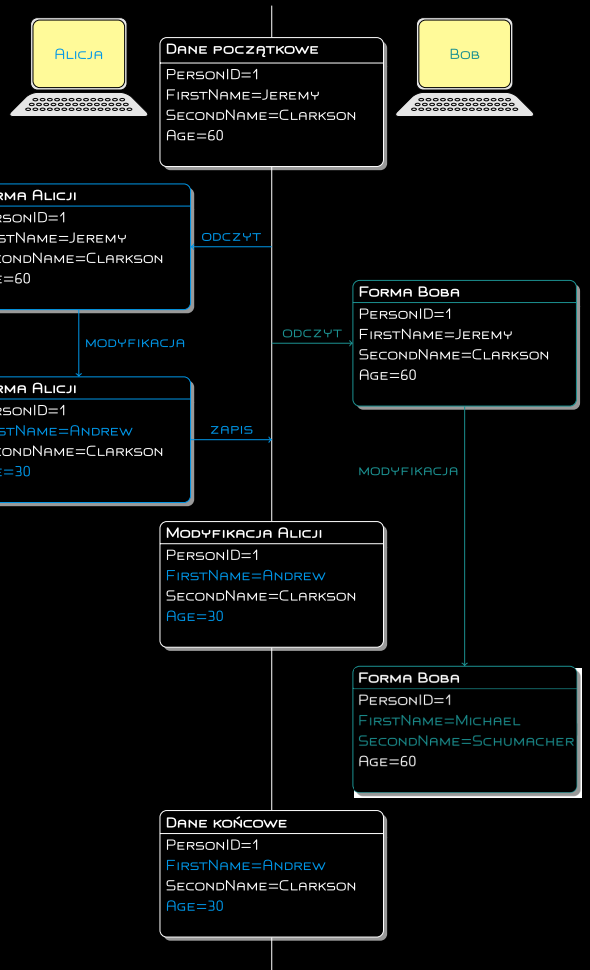
- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje



- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje

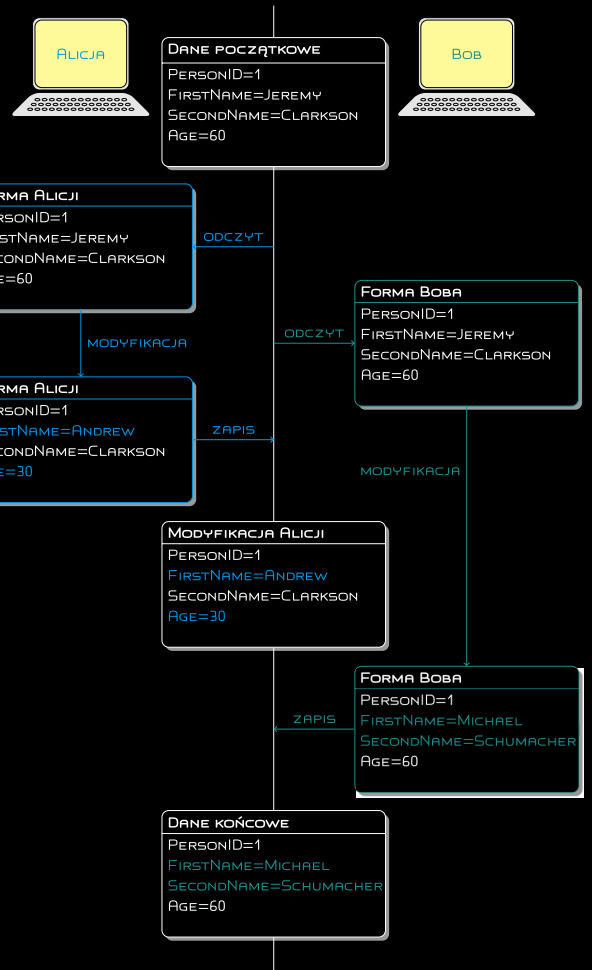


- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje

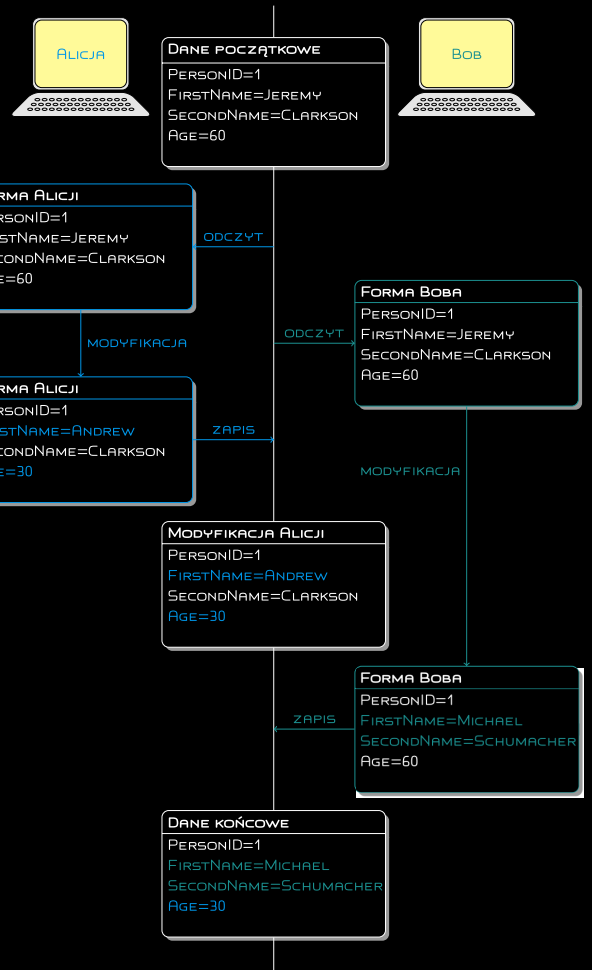


- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
  - cały rekord

```
1 UPDATE Persons SET
2   FirstName=?, SecondName=?,
3   Age = ?
4 WHERE PersonID = ?
```



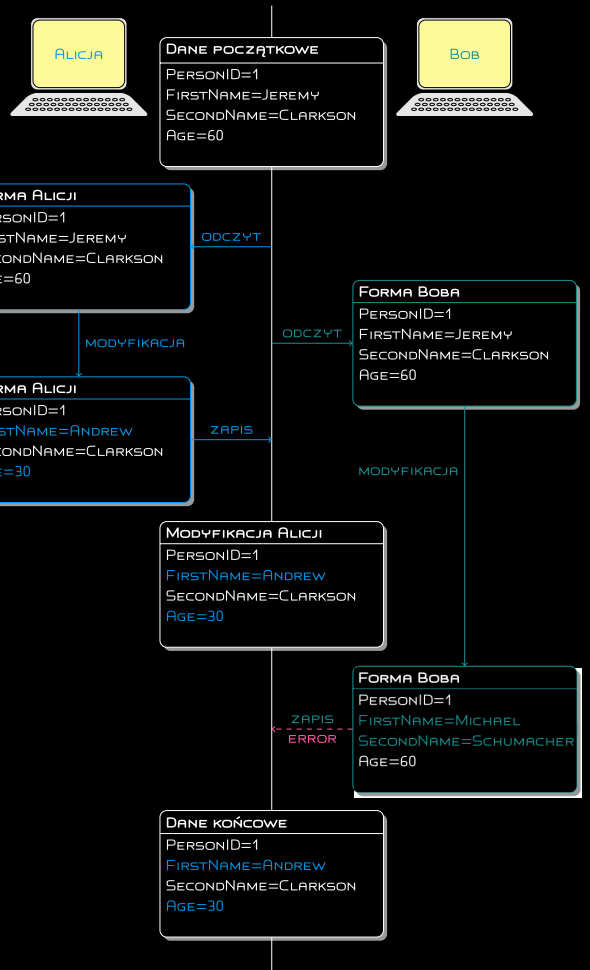
- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
  - cały rekord
  - tylko zmienione pola




```
1 UPDATE Persons SET
2   FirstName=?,
3   SecondName=?
4 WHERE PersonID = ?
```

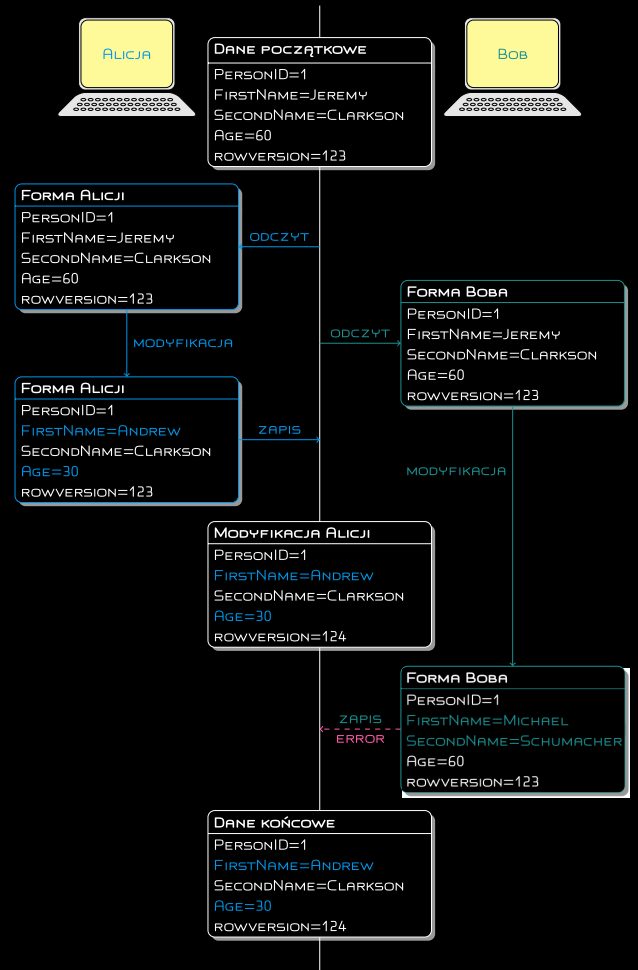
- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
  - cały rekord
  - tylko zmienione pola
  - sprawdzając wszystkie pola

```
1 UPDATE Persons SET
2   FirstName=?, SecondName=?
3 WHERE PersonID = ? AND FirstName = ?
4   AND SecondName = ? AND Age = ?
```

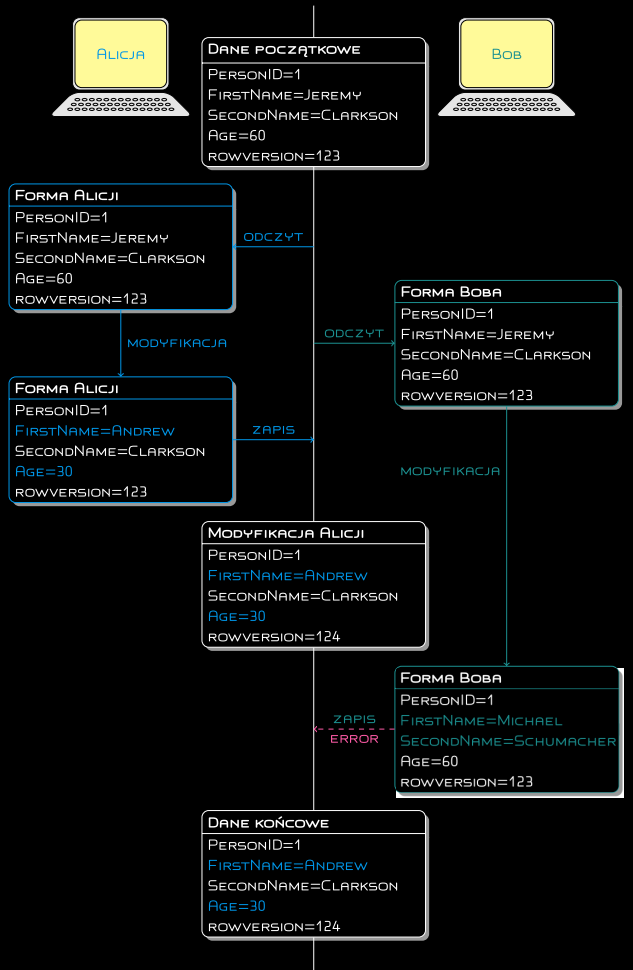


- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
  - cały rekord
  - tylko zmienione pola
  - sprawdzając wszystkie pola
  - tylko pole RowVersion

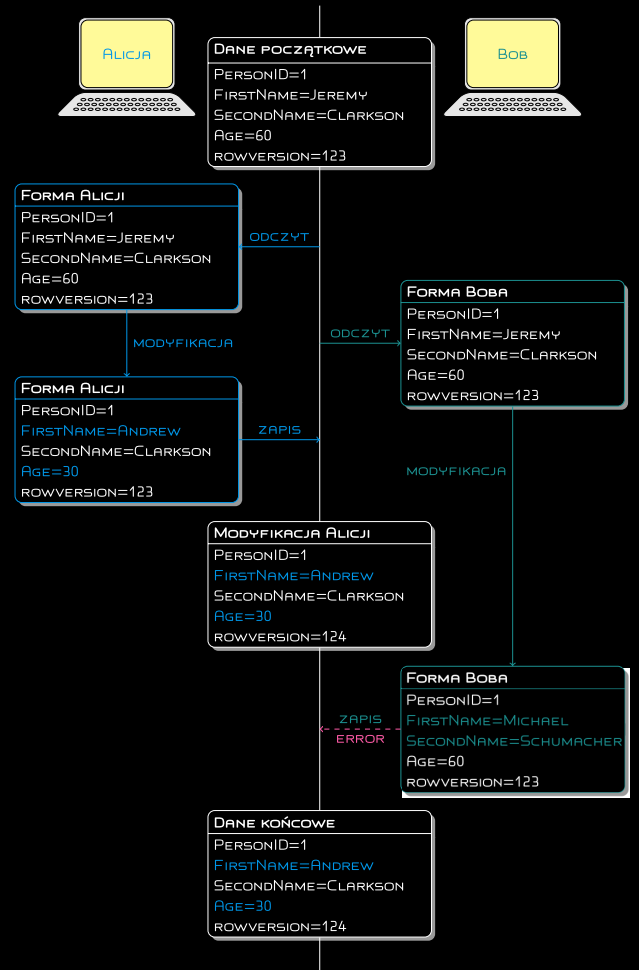
```
1 UPDATE Persons SET
2   FirstName=?, SecondName=?
3 WHERE PersonID = ? AND RowVersion = ?
```



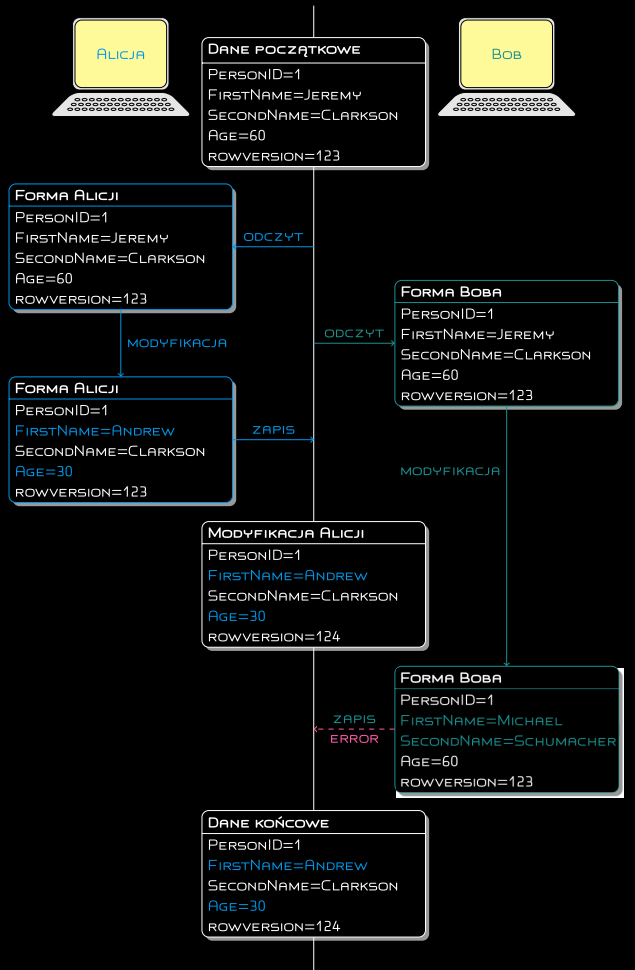
- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
- musimy przechować stare wartości pól do porównania



- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
- musimy przechować stare wartości pól do porównania
  - przy modyfikacji pól zmienionych



- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
- musimy przechować stare wartości pól do porównania
  - przy modyfikacji pól zmienionych
  - przy sprawdzeniu równoległości



- Rejestrowanie (logowanie) zdarzeń ułatwia kontrolę wykonania programu.

- Rejestrowanie (logowanie) zdarzeń ułatwia kontrolę wykonania programu.
- Umożliwia analizę sytuacji w przypadku awarii systemu

- Rejestrowanie (logowanie) zdarzeń ułatwia kontrolę wykonania programu.
- Umożliwia analizę sytuacji w przypadku awarii systemu
- Zamiast wydruków na konsolę (`System.out.println()`), których nie można konfigurować wykorzystujemy biblioteki

- Rejestrowanie (logowanie) zdarzeń ułatwia kontrolę wykonania programu.
- Umożliwia analizę sytuacji w przypadku awarii systemu
- Zamiast wydruków na konsolę (`System.out.println()`), których nie można konfigurować wykorzystujemy biblioteki

  log4j -log for Java - już nie wspierana

- Rejestrowanie (logowanie) zdarzeń ułatwia kontrolę wykonania programu.
- Umożliwia analizę sytuacji w przypadku awarii systemu
- Zamiast wydruków na konsolę (`System.out.println()`), których nie można konfigurować wykorzystujemy biblioteki

  log4j -log for Java - już nie wspierana

  log4j2 -log for Java 2

- Rejestrowanie (logowanie) zdarzeń ułatwia kontrolę wykonania programu.
- Umożliwia analizę sytuacji w przypadku awarii systemu
- Zamiast wydruków na konsolę (`System.out.println()`), których nie można konfigurować wykorzystujemy biblioteki

  log4j -log for Java - już nie wspierana

  log4j2 -log for Java 2

  JCL - (Jakarta Commons Logging)

- Rejestrowanie (logowanie) zdarzeń ułatwia kontrolę wykonania programu.
- Umożliwia analizę sytuacji w przypadku awarii systemu
- Zamiast wydruków na konsolę (`System.out.println()`), których nie można konfigurować wykorzystujemy biblioteki

  log4j -log for Java - już nie wspierana

  log4j2 -log for Java 2

  JCL - (Jakarta Commons Logging)

  Logback -

- Rejestrowanie (logowanie) zdarzeń ułatwia kontrolę wykonania programu.
- Umożliwia analizę sytuacji w przypadku awarii systemu
- Zamiast wydruków na konsolę (`System.out.println()`), których nie można konfigurować wykorzystujemy biblioteki

  log4j -log for Java - już nie wspierana

  log4j2 -log for Java 2

  JCL - (Jakarta Commons Logging)

  Logback -
- SLF4J - Simple Logging Facade for Java (biblioteka przykrywająca jedną z powyższych implementacji)

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu
  - możemy konfigurację dla konkretnej klasy w pakiecie

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu
  - możemy konfigurację dla konkretnej klasy w pakiecie
- możliwość definiowania appenderów, określających miejsce gdzie trafiają komunikaty

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu
  - możemy konfigurację dla konkretnej klasy w pakiecie
- możliwość definiowania appenderów, określających miejsce gdzie trafiają komunikaty
  - konsola

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu
  - możemy konfigurację dla konkretnej klasy w pakiecie
- możliwość definiowania appenderów, określających miejsce gdzie trafiają komunikaty
  - konsola
  - plik

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu
  - możemy konfigurację dla konkretnej klasy w pakiecie
- możliwość definiowania appenderów, określających miejsce gdzie trafiają komunikaty
  - konsola
  - plik
  - pliki z nadpisywaniem najstarszego

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu
  - możemy konfigurację dla konkretnej klasy w pakiecie
- możliwość definiowania appenderów, określających miejsce gdzie trafiają komunikaty
  - konsola
  - plik
  - pliki z nadpisywaniem najstarszego
  - baza danych

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu
  - możemy konfigurację dla konkretnej klasy w pakiecie
- możliwość definiowania appenderów, określających miejsce gdzie trafiają komunikaty
  - konsola
  - plik
  - pliki z nadpisywaniem najstarszego
  - baza danych
- ten sam komunikat może być użyty przez wiele logerów

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu
  - możemy konfigurację dla konkretnej klasy w pakiecie
- możliwość definiowania appenderów, określających miejsce gdzie trafiają komunikaty
  - konsola
  - plik
  - pliki z nadpisywaniem najstarszego
  - baza danych
- ten sam komunikat może być użyty przez wiele logerów
- każdy loger może mieć podpięte wiele appenderów

- możliwość definiowania nazwanych konfiguracji logerów, dla których ustawiamy parametry
  - nazwy mogą być ręczne
  - możemy mieć wspólną konfigurację dla pakietu
  - możemy konfigurację dla konkretnej klasy w pakiecie
- możliwość definiowania appenderów, określających miejsce gdzie trafiają komunikaty
  - konsola
  - plik
  - pliki z nadpisywaniem najstarszego
  - baza danych
- ten sam komunikat może być użyty przez wiele logerów
- każdy loger może mieć podpięte wiele appenderów
- konfiguracja plikiem na zewnątrz

pom.xml

```
<properties>
  <org.slf4j.version>1.7.30</org.slf4j.version>
</properties>
<dependencies>
  <!-- Logging -->
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>\${org.slf4j.version}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>\${org.slf4j.version}</version>
    <scope>runtime</scope>
  </dependency>
  ...
</dependencies>
```

log2j.properties

```
log4j.rootLogger=WARN, stdout
log4j.logger.pap.DBContext=DEBUG

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
```

- domyślnie wszystkie klasy mają poziom WARN i używają appendera o nazwie stdout

log2j.properties

```
log4j.rootLogger=WARN, stdout
log4j.logger.pap.DBContext=DEBUG

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
```

- domyślnie wszystkie klasy mają poziom WARN i używają appendera o nazwie stdout
- klasa pap.DBContext ma ustawienia domyślne, ale poziom DEBUG

log2j.properties

```
log4j.rootLogger=WARN, stdout
log4j.logger.pap.DBContext=DEBUG

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
```

- domyślnie wszystkie klasy mają poziom WARN i używają appendera o nazwie stdout
- klasa pap.DBContext ma ustawienia domyślne, ale poziom DEBUG
- appender stdout pisze na konsolę

DBContext.java

```
1  package pap;
2
3  import org.slf4j.Logger;
4  import org.slf4j.LoggerFactory;
5
6  public class DBContext {
7    private static final Logger log = LoggerFactory.getLogger(DBContext.class);
8    private Connection conn = null;
9
10   public void close() {
11     if (conn != null) {
12       try {
13         log.info("Closing database connection to bookStoreDB");
14         ...
15       } catch( ...
16       ...
17   }
18   ...
19 }
```

- info - czyli przy takich ustawieniach będzie nie milczał i wyświetli

pom.xml

```xml
<dependencies>
  ...
  <!-- Log4j 2 -->
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.6.1</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.6.1</version>
  </dependency>
  ...
</dependencies>
```

- Poziomy służą do określania wagi zdarzenia

```
private static Logger logger = LogManager.getLogger(App.class);
...
logger.fatal("Fatal Message!");
```

- Poziomy służą do określania wagi zdarzenia
- Poziomy są zorganizowane od największego do najmniej szczegółowego:

```
1 private static Logger logger = LogManager.getLogger(App.class);
2 ...
3 logger.fatal("Fatal Message!");
```

- Poziomy służą do określania wagi zdarzenia
- Poziomy są zorganizowane od największego do najmniej szczegółowego:

  OFF - bez logowania

```
private static Logger logger = LogManager.getLogger(App.class);
...
logger.fatal("Fatal Message!");
```

- Poziomy służą do określania wagi zdarzenia
- Poziomy są zorganizowane od największego do najmniej szczegółowego:

  OFF - bez logowania

  FATAL - poważne błędy, które uniemożliwiają dalsze działanie aplikacji

```
1 private static Logger logger = LogManager.getLogger(App.class);
2 ...
3 logger.fatal("Fatal Message!");
```

- Poziomy służą do określania wagi zdarzenia
- Poziomy są zorganizowane od największego do najmniej szczegółowego:

  OFF - bez logowania

  FATAL - poważne błędy, które uniemożliwiają dalsze działanie aplikacji

  ERROR - poważne błędy, możliwe do naprawienia

```
1 private static Logger logger = LogManager.getLogger(App.class);
2 ...
3 logger.error("Error Message!");
```

- Poziomy służą do określania wagi zdarzenia
- Poziomy są zorganizowane od największego do najmniej szczegółowego:

  OFF - bez logowania

  FATAL - poważne błędy, które uniemożliwiają dalsze działanie aplikacji

  ERROR - poważne błędy, możliwe do naprawienia

  WARN - problemy potencjalnie istotne

```
1 private static Logger logger = LogManager.getLogger(App.class);
2 ...
3 logger.warn("Warn Message!");
```

- Poziomy służą do określania wagi zdarzenia
- Poziomy są zorganizowane od największego do najmniej szczegółowego:

  OFF - bez logowania

  FATAL - poważne błędy, które uniemożliwiają dalsze działanie aplikacji

  ERROR - poważne błędy, możliwe do naprawienia

  WARN - problemy potencjalnie istotne

  INFO - potwierdzenia

```
1 private static Logger logger = LogManager.getLogger(App.class);
2 ...
3 logger.info("Info Message!");
```

- Poziomy służą do określania wagi zdarzenia
- Poziomy są zorganizowane od największego do najmniej szczegółowego:

  OFF - bez logowania

  FATAL - poważne błędy, które uniemożliwiają dalsze działanie aplikacji

  ERROR - poważne błędy, możliwe do naprawienia

  WARN - problemy potencjalnie istotne

  INFO - potwierdzenia

  DEBUG - informacja pomocna w uruchamianiu

```
1 private static Logger logger = LogManager.getLogger(App.class);
2 ...
3 logger.debug("Debug Message!");
```

- Poziomy służą do określania wagi zdarzenia
- Poziomy są zorganizowane od największego do najmniej szczegółowego:

  OFF - bez logowania

  FATAL - poważne błędy, które uniemożliwiają dalsze działanie aplikacji

  ERROR - poważne błędy, możliwe do naprawienia

  WARN - problemy potencjalnie istotne

  INFO - potwierdzenia

  DEBUG - informacja pomocna w uruchamianiu

  TRACE - szczegółowa informacja o uruchamianiu

```
1 private static Logger logger = LogManager.getLogger(App.class);
2 ...
3 logger.trace("Trace Message!");
```

- Poziomy służą do określania wagi zdarzenia
- Poziomy są zorganizowane od największego do najmniej szczegółowego:

  OFF - bez logowania

  FATAL - poważne błędy, które uniemożliwiają dalsze działanie aplikacji

  ERROR - poważne błędy, możliwe do naprawienia

  WARN - problemy potencjalnie istotne

  INFO - potwierdzenia

  DEBUG - informacja pomocna w uruchamianiu

  TRACE - szczegółowa informacja o uruchamianiu

  ALL - wszystko

składniki:

appenders -gdzie i jaki format

```
status = no
name = PropertiesConfig
filters = threshold
filter.threshold.type = ThresholdFilter
filter.threshold.level = trace

appenders = console,console2,console3
appender.console.type = Console
appender.console.name = STDOUT
appender.console.layout.type = PatternLayout
appender.console.layout.pattern  = STDOUT  %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
appender.console2.type = Console
appender.console2.name = STDOUT2
appender.console2.layout.type = PatternLayout
appender.console2.layout.pattern = STDOUT2 %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
appender.console3.type = Console
appender.console3.name = STDOUT3
appender.console3.layout.type = PatternLayout
appender.console3.layout.pattern = STDOUT3 %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
```

składniki:

appenders -gdzie i jaki format

```
20 loggers=app
21 logger.app.name = pap.App
22 logger.app.level = fatal
23 #logger.app.additivity = false
24 logger.app.appenderRefs = stdout
25 logger.app.appenderRef.stdout.ref = STDOUT
26
27 rootLogger.level = error
28 rootLogger.appenderRefs = stdout
29 rootLogger.appenderRef.stdout.ref = STDOUT2
30 rootLogger.appenderRef.stdout1.ref = STDOUT3
```

składniki:

appenders -gdzie i jaki format
loggers -skąd i jaki poziom

```
status = no
name = PropertiesConfig
filters = threshold
filter.threshold.type = ThresholdFilter
filter.threshold.level = trace

appenders = console,console2,console3
appender.console.type = Console
appender.console.name = STDOUT
appender.console.layout.type = PatternLayout
appender.console.layout.pattern  = STDOUT  %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
appender.console2.type = Console
appender.console2.name = STDOUT2
appender.console2.layout.type = PatternLayout
appender.console2.layout.pattern = STDOUT2 %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
appender.console3.type = Console
appender.console3.name = STDOUT3
appender.console3.layout.type = PatternLayout
appender.console3.layout.pattern = STDOUT3 %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
```

składniki:

appenders -gdzie i jaki format
loggers -skąd i jaki poziom

```
20 loggers=app
21 logger.app.name = pap.App
22 logger.app.level = fatal
23 #logger.app.additivity = false
24 logger.app.appenderRefs = stdout
25 logger.app.appenderRef.stdout.ref = STDOUT
26
27 rootLogger.level = error
28 rootLogger.appenderRefs = stdout
29 rootLogger.appenderRef.stdout.ref = STDOUT2
30 rootLogger.appenderRef.stdout1.ref = STDOUT3
```

składniki:

appenders -gdzie i jaki format
loggers -skąd i jaki poziom
additivity oznacza że nie kończymy dopasowania nazwy

```
status = no
name = PropertiesConfig
filters = threshold
filter.threshold.type = ThresholdFilter
filter.threshold.level = trace

appenders = console,console2,console3
appender.console.type = Console
appender.console.name = STDOUT
appender.console.layout.type = PatternLayout
appender.console.layout.pattern  = STDOUT  %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
appender.console2.type = Console
appender.console2.name = STDOUT2
appender.console2.layout.type = PatternLayout
appender.console2.layout.pattern = STDOUT2 %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
appender.console3.type = Console
appender.console3.name = STDOUT3
appender.console3.layout.type = PatternLayout
appender.console3.layout.pattern = STDOUT3 %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
```

składniki:

appenders -gdzie i jaki format
loggers -skąd i jaki poziom
additivity oznacza że nie kończymy dopasowania nazwy

```
20 loggers=app
21 logger.app.name = pap.App
22 logger.app.level = fatal
23 #logger.app.additivity = false
24 logger.app.appenderRefs = stdout
25 logger.app.appenderRef.stdout.ref = STDOUT
26
27 rootLogger.level = error
28 rootLogger.appenderRefs = stdout
29 rootLogger.appenderRef.stdout.ref = STDOUT2
30 rootLogger.appenderRef.stdout1.ref = STDOUT3
```

formaty plików:

tekstowy -log4j2.properties

```
status = no
name = PropertiesConfig
filters = threshold
filter.threshold.type = ThresholdFilter
filter.threshold.level = trace

appenders = console,console2,console3
appender.console.type = Console
appender.console.name = STDOUT
appender.console.layout.type = PatternLayout
appender.console.layout.pattern  = STDOUT  %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
appender.console2.type = Console
appender.console2.name = STDOUT2
appender.console2.layout.type = PatternLayout
appender.console2.layout.pattern = STDOUT2 %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
appender.console3.type = Console
appender.console3.name = STDOUT3
appender.console3.layout.type = PatternLayout
appender.console3.layout.pattern = STDOUT3 %d{yyyy-MM-dd HH:mm:ss}
 ↪  %-5p %c{1}:%L - %m%n
```

formaty plików:

tekstowy -log4j2.properties

```
20 loggers=app
21 logger.app.name = pap.App
22 logger.app.level = fatal
23 #logger.app.additivity = false
24 logger.app.appenderRefs = stdout
25 logger.app.appenderRef.stdout.ref = STDOUT
26
27 rootLogger.level = error
28 rootLogger.appenderRefs = stdout
29 rootLogger.appenderRef.stdout.ref = STDOUT2
30 rootLogger.appenderRef.stdout1.ref = STDOUT3
```

formaty plików:

tekstowy -log4j2.properties
xml -log4j2.xml

formaty plików:

tekstowy -log4j2.properties
xml -log4j2.xml
json -log4j2.json

App.java:

```java
package pap;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class App {
  private static Logger logger = LogManager.getLogger(App.class);
  public static void main(String[] args) {
    System.out.println("++++++++++++Hello World! ++++++++++");
    logger.trace("Trace Message!");
    logger.debug("Debug Message!");
    logger.info("Info Message!");
    logger.warn("Warn Message!");
    logger.error("Error Message!");
    logger.fatal("Fatal Message!");
    System.out.println("-------------Hello World!-----------");
    new Compute().compute();
  }
}
```

Compute.java:

```
1  package pap;
2  import org.apache.logging.log4j.LogManager;
3  import org.apache.logging.log4j.Logger;
4
5  public class Compute {
6    private static Logger logger = LogManager.getLogger(Compute.class);
7    public void compute() {
8      System.out.println("compute: ++++++++++++Compute World! ++++++++++");
9      logger.trace("compute: Trace Message!");
10     logger.debug("compute: Debug Message!");
11     logger.info("compute: Info Message!");
12     logger.warn("compute: Warn Message!");
13     logger.error("compute: Error Message!");
14     logger.fatal("compute: Fatal Message!");
15     System.out.println("compute: ------------Compute World!----------");
16   }
17 }
18
```

- w klasie `App` działają wszystkie loggery - mamy komunikaty od ich appenderów

wynik:

```
++++++++++++Hello World! ++++++++++
STDOUT  2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT3 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
------------Hello World!-----------
compute: ++++++++++++Compute World! ++++++++++
STDOUT3 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT2 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT3 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
compute: ------------Compute World!-----------
```

- w klasie `App` działają wszystkie loggery - mamy komunikaty od ich appenderów

  app - ma podpięty appender STDOUT

wynik:

```
++++++++++++Hello World! ++++++++++
STDOUT  2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT3 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
------------Hello World!-----------
compute: ++++++++++++Compute World! ++++++++++
STDOUT3 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT2 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT3 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
compute: ------------Compute World!-----------
```

- w klasie `App` działają wszystkie loggery - mamy komunikaty od ich appenderów

  app - ma podpięty appender STDOUT

  rootlogger - loger domyślny ma podpięte appendery STDOUT2 i STDOUT3

wynik:

```
++++++++++++Hello World! ++++++++++
STDOUT  2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT3 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
------------Hello World!-----------
compute: ++++++++++++Compute World! ++++++++++
STDOUT3 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT2 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT3 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
compute: ------------Compute World!-----------
```

- w klasie `App` działają wszystkie loggery - mamy komunikaty od ich appenderów

  app - ma podpięty appender STDOUT

  rootlogger - loger domyślny ma podpięte appendery STDOUT2 i STDOUT3

- poziom logowania ustalony na fatal - tylko komunikaty fatal są wyświetlane

wynik:

```
++++++++++++Hello World! ++++++++++
STDOUT  2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT3 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
-------------Hello World!-----------
compute: ++++++++++++Compute World! ++++++++++
STDOUT3 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT2 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT3 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
compute: -------------Compute World!-----------
```

- w klasie `Compute` działa appender rootloger -

wynik:

```
++++++++++++Hello World! ++++++++++
STDOUT  2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT3 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
-------------Hello World!-----------
compute: ++++++++++++Compute World! ++++++++++
STDOUT3 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT2 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT3 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
compute: -------------Compute World!-----------
```

- w klasie `Compute` działa appender rootloger -

  rootlogger - loger domyślny ma podpięte appendery STDOUT2 i STDOUT3

wynik:

```
++++++++++++Hello World! ++++++++++
STDOUT  2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT3 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
-------------Hello World!-----------
compute: ++++++++++++Compute World! ++++++++++
STDOUT3 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT2 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT3 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
compute: -------------Compute World!-----------
```

- w klasie `Compute` działa appender rootloger -

  rootlogger - loger domyślny ma podpięte appendery STDOUT2 i STDOUT3

- poziom logowania ustalony na error - tylko komunikaty error i fatal są wyświetlane

wynik:

```
++++++++++++Hello World! ++++++++++
STDOUT  2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT3 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL App:15 - Fatal Message!
-------------Hello World!-----------
compute: ++++++++++++Compute World! ++++++++++
STDOUT3 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT2 2021-01-10 22:35:09 ERROR Compute:14 - compute: Error Message!
STDOUT3 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
STDOUT2 2021-01-10 22:35:09 FATAL Compute:15 - compute: Fatal Message!
compute: -------------Compute World!-----------
```