Politechnika Wrocławska

# Platformy Programistyczne .NET i Java

## Zaawansowane programowanie w języku JAVA

Dr inż. Radosław Idzikowski

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

# Agenda

1. Wątki
2. Bazy danych
3. Powtórzenie materiału

# Wątki – wprowadzenie

## Wątki (threads)

w języku Java są podstawową jednostką wykonania programu i umożliwiają wykonywanie zadań współbieżnie. Dzięki wątkom możemy uruchamiać równocześnie wiele operacji, co jest szczególnie przydatne w aplikacjach wymagających wysokiej wydajności, takich jak serwery webowe, aplikacje obliczeniowe czy aplikacje reagujące na zdarzenia.

# Tworzenie wątków

## Rozszerzenie klasy Thread

```
class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("Thread: " + Thread.currentThread().getName());
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        myThread.start();
    }
}
```

# Tworzenie wątków

## Implementacja interfejsu Runnable

```
class MyRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("Thread: " + Thread.currentThread().getName());
    }
}

public class Main {
    public static void main(String[] args) {
        MyRunnable myRunnable = new MyRunnable();
        Thread thread = new Thread(myRunnable);
        thread.start();
    }
}
```

# Synchronizacja wątków

## Synchronizacja wątków

w Javie jest kluczowym aspektem programowania współbieżnego, pozwalającym na bezpieczne współdzielenie zasobów między wieloma wątkami. Synchronizacja zapobiega problemom takim jak wyścigi danych, deadlocki czy niespójność danych. Oto szczegółowy opis mechanizmów synchronizacji dostępnych w Javie:

# Synchronizacja wątków

## Synchronizacja metody

Synchronizowanie całej metody za pomocą słowa kluczowego synchronized zapewnia, że tylko jeden wątek może uzyskać dostęp do tej metody na raz.

```
class Counter {
    private int count = 0;

    public synchronized void increment() {
        count++;
    }

    public synchronized int getCount() {
        return count;
    }
}
```

# Synchronizacja wątków

## Synchronizacja bloku kodu

Synchronizowanie bloku kodu jest bardziej elastyczne, ponieważ pozwala na synchronizację tylko części metody.

```
class Counter {
    private int count = 0;

    public void increment() {
        synchronized (this) {
            count++;
        }
    }

    public int getCount() {
        synchronized (this) {
            return count;
        }
    }
}
```

# Synchronizacja wątków

## Synchronizacja z użyciem obiektu

Bloki kodu mogą być synchronizowane na dowolnym obiekcie, co pozwala na bardziej precyzyjną kontrolę nad dostępem do różnych zasobów.

```
class Counter {
    private int count = 0;
    private final Object lock = new Object();

    public void increment() {
        synchronized (lock) {
            count++;
        }
    }

    public int getCount() {
        synchronized (lock) {
            return count;
        }
    }
}
```

# Synchronizacja wątków

## Monitorowanie Obiektów

### Monitorowanie Obiektów

Każdy obiekt w Javie może być używany jako monitor (lock), co pozwala na synchronizację wątków. Synchronizacja bloku kodowego na danym obiekcie sprawia, że wątki muszą uzyskać dostęp do monitora tego obiektu, zanim będą mogły wykonać kod wewnątrz bloku.

# Synchronizacja wątków

## Monitorowanie Obiektów

Poniższe `metody` służą do zarządzania komunikacją między wątkami, które są zsynchronizowane na tym samym obiekcie. Są one definiowane w klasie Object.

- `wait()`: Powoduje, że bieżący wątek czeka, dopóki inny wątek nie wywoła metody `notify()` lub `notifyAll()` na tym samym obiekcie,
- `notify()`: Budzi jeden z wątków, który czeka na monitorze tego obiektu.,
- `notifyAll()`: Budzi wszystkie wątki, które czekają na monitorze tego obiektu.

# Synchronizacja wątków

## Monitorowanie Obiektów

```
class SharedResource {
private int value = 0;
private boolean available = false;

public synchronized void put(int value) {
    while (available) {
        try {
            wait();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
    this.value = value;
    available = true;
    notifyAll();
}
public synchronized int get() {
    ...
}
}
```

# Synchronizacja wątków

## Monitorowanie Obiektów

```
class SharedResource {
private int value = 0;
private boolean available = false;

public synchronized void put(int value) {
    ...
}
public synchronized int get() {
    while (!available) {
        try {
            wait();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
    available = false;
    notifyAll();
    return value;
}
}
```

# Zaawansowane techniki zarządzania wątkami

Java oferuje również bardziej zaawansowane techniki, takie jak:

- `Executor Framework`: zbiór interfejsów i klas do zarządzania pulami wątków,
- `Fork/Join Framework`: wspiera wykonywanie zadań, które można podzielić na mniejsze części.
- `Classes from java.util.concurrent package`: Takie jak `CountDownLatch`, `CyclicBarrier`, `Semaphore`, itp.

# Zaawansowane techniki zarządzania wątkami

## Przykład użycia ExecutorService

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Main {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(2);

        executor.submit(() -> {
            System.out.println("Thread 1 is runing: " + Thread.currentThread().
        });

        executor.submit(() -> {
            System.out.println("Thread 2 is runing: " + Thread.currentThread().
        });

        executor.shutdown();
    }
}
```

**Hibernate**

Jest popularnym narzędziem ORM (Object-Relational Mapping) w języku Java, które umożliwia programistom mapowanie obiektów Java na relacyjne bazy danych. Pozwala to na efektywne zarządzanie danymi i operacjami na bazie danych, nie wymagając bezpośredniego używania języka SQL.

# Bazy danych

## Przykład użycia ExecutorService

```
@Entity
@Table(name = "uzytkownicy")
public class Uzytkownik {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "imie")
    private String imie;

    @Column(name = "nazwisko")
    private String nazwisko;
}
```

Politechnika Wrocławska

Powtórzenie materiału.