Zajęcia powtórkowe

Adam Łagoda

Agenda

1. Współbieżność (Concurrency) w Javie
2. Budowanie projektu
3. Operacje I/O
4. Aplikacja bazodanowa

Przydatne informacje

1. Repozytorium jest dostępne pod https://github.com/adamlagoda/zajecia_powtorkowe.git
2. Rozwiązania zadań znajdują się na branchach, które będą wrzucane do repozytorium zdalnego pod koniec czasu na wykonanie zadania
3. Prezentacja jest dostępna w formie PDF-a w repozytorium


Concurrency

Concurrency - synchronizacja

- Klasa `Thread` reprezentuje wątek w Javie
- Wątek tworzymy poprzez:
  - *extends Thread*
  - *implements Runnable*
- Wątek ma własny stos pamięci
- Metody:
  - sleep() – zawiesza wykonywanie wątku na określony w milisekundach czas
  - join() – powoduje, że wywołujący wątek czeka na zakończenie wywoływanego
  - interrupt() – wysłanie sygnału do wątku, aby przerwał swoją pracę
  - isAlive() – sprawdzenie, czy wątek zakończył pracę
- *Race condition* – gdy > 1 wątek próbuje uzyskać dostęp do współdzielonego zasoby


Demo

Concurrency - synchronizacja

- *Interleaving*
- *Memory consistency*
- `synchronized` – słówko kluczowe umieszczane na poziomie:
  - bloku kodu
  - Metody
- *Reentrant lock*
- `volatile` – dostęp do zmiennej atomow (wymusza zapis bezpośrednio do RAM-u)

Concurrency - synchronizacja

- *Deadlock* – gdy komunikujące się ze sobą wątki czekają na dostęp do swoich monitorów
- *Starvation* - gdy jeden z wątków nie może kontynuować pracy, bo dostęp do współdzielonego zasoby jest blokowany przez inny przez bardzo długi czas
- *Livelock* – gdy metoda wykonywana przez jeden z wątków jest reakcją na metodę drugiego. Może się wtedy okazać, że oba wątki będą w nieskończoność wykonywać swoje metody – aplikacja się „zapętli”

https://docs.oracle.com/javase/tutorial/essential/concurrency/

Concurrency - synchronizacja

- *Guarded lock* – metoda synchronizacji wątków. Aby dostać się do sekcji krytycznej aplikacji, należy spełnić warunek dostępu oraz następnie pobrać lock
- W Javie realizowany z wykorzystaniem metod:
  - `wait()` – zawieszenie wątku do momentu otrzymania notyfikacji z innych wątków
  - `notifyAll()` – wysłanie do pozostałych wątków notyfikacji, że jakieś szczególne zdarzenie miało miejsce (np. zmiana wartości jakiejś flagi `boolean`)

https://docs.oracle.com/javase/tutorial/essential/concurrency/

Zadanie

1. Zaimplementuj rozwiązanie problemu producent-konsument z wykorzystaniem strategii *guarded lock*
2. Wewnątrz pakietu `concurrency.guardedlock` znajdują się klasy:
   1. `Producer` – wysyła wiadomość opakowaną w obiekt `Message`
   2. `Consumer` – pobiera wiadomość ze współdzielonego obiektu `Message`

https://docs.oracle.com/javase/tutorial/essential/concurrency/

Budowanie projektu

Zadanie

1. Stwórz w notatniku plik `HelloWorld.java` i zapisz do katalogu `/org/example/`
2. Skompiluj plik: `javac -d target org/example/HelloWorld.java`
3. Uruchom aplikację: `java -cp <path_to_target> org.example.HelloWorld`
4. Dodaj Logger:
   1. Pobierz paczkę Log4j: https://repo1.maven.org/maven2/log4j/log4j/1.2.17/ do katalogu `lib`
   2. W pliku `HelloWorld.java` dodaj linijki:

```
import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;
public class HelloWorld {
     private static final Logger logger = Logger.getLogger(HelloWorld.class);
     public static void main(String[] args) {
               BasicConfigurator.configure();
               logger.info("Hello world!");
     }
}
```

5. Powtórz kroki 1-2
6. Dodaj `log4j-1.2.17.jar` do classpath

Maven

1. Zbuduj aplikację z poprzedniego zadania z wykorzystaniem Mavena
2. Dodaj plugin `exec-maven-plugin`

```
<plugin>
   <groupId>org.codehaus.mojo</groupId>
   <artifactId>exec-maven-plugin</artifactId>
   <version>1.5.0</version>
   <configuration>
      <mainClass>org.baeldung.java.App</mainClass>
   </configuration>
</plugin>
```

3. Wykonaj polecenie `mvn exec:java`


Java I/O

ByteStream

- `FileinputStream/FileOutputStream` – operują na pojedynczych bajtach. Wartości int o zakresie od 0 do 255. W większości przypadków, mało wydajne. Można poprawić wydajność, korzystając z buforowania
- `BufferedInputStream/BufferedOutputStream` – bufor jest zarządzany prze Javę.

Zadanie

1. Uzupełnij metodę `copySingleBytes()` z klasy `BytesStream`
2. W metodzie `copyUsingBuffer()` do tej samej operacji użyj 4kB bufora
3. Przepisz zawartość plików `input.txt` do `output.txt`, wykorzystując `BufferedInputStream` i `BufferedOutputStream` w metodzie `copyUsingBufferedStreams()`

CharacterStream

- Java przechowuje znaki korzystając z kodowania USC-2 16-bitowego
- Aby poprawnie wczytywać i zapisywać znaki, niezależnie od kodowania zdefiniowanego w pliku, reader/writer musi wykonać konwersję (dla dewelopara jest transparentna)
- `FileWriter/FileReader` - implementacje klas `Writer/Reader`. Działają analogicznie do `FileInputStream/FileOutputStream`, ale operują na 16-bitowych `int`-ach
- `BufferedReader/BufferedWriter` – udostępniają metody `readLine()` i `println()`
- `InputStreamReader/OutputStreamWriter` – umożliwiają wybór sposobu kodowania znaków.

Zadanie

1. Analogicznie do metody `copySingleBytes()` w klasie `ByteStream`, uzupełnij metodę `copySingleCharacters()` w klasie `CharacterStreams`
2. Uzupełnij kod metody `copyUsingBuffer()` w klasie `CharacterStreams`, wykorzystując bufor 4kB
3. Wykonaj tą samą operację, kopiowania zawartości pliku `input.txt`, korzystając z `BufferedReader/BufferedWriter` w metodzie `copyUsingBufferedStreams()`
4. Przepisz zawartość pliku `input.txt` do `output.txt` zmieniając kodowanie na `UTF-8` i klasy `InputStreamReader` oraz `OutputStreamWriter`

Formatting

- `Scanner` – na podstawie delimitera, dzieli źródło na tokeny, po czym dokonuje konwersji typów
- `Formatter` – interpreter, za pomocą którego można zapisać sformatowany tekst
- `PrintWriter` – stumień wyjściowy. Najczęściej spotykany w konstrukcji `System.out`

Zadanie

1. Wczytaj zawartość pliku `formatted.txt`, wykorzystując `Scanner` w klasie `Formatting`
2. Pobraną zawartość pliku zapisz do `formatted-output.txt`, korzystając z
   1. Klasy `Formatter`
   2. Klasy `PrintWritter`

Java.util.nio

- Małe pliki:
  - `Files.readAllBytes()/Files.write()`- bajty
  - `Files.readAllLines()/Files.writeString()` – pliki tekstowe
- Ogólnie:
  - `Files.newBufferedReader()/Files.newBufferedWriter()` - bajty
  - `Files.newOutputStream()/Files.newInputStream()` – pliki tekstowe

Java.nio.file.Path

- Klasa, która reprezentuje ścieżkę do pliku w systemie (istniejącą lub nieistniejącą)
- Od wersji Javy 7 zastępuje i wzbogaca o nową funkcjonalność klasę `java.io.File`
- Klasy pomocnicze, które zawierają metody statyczne:
  - `Files`
  - `Paths`

Zadanie

1. Napisz aplikację, która skopiuje plik, który znajduje się pod ścieżką podaną, jako pierwszy argument, do lokalizacji, podanej jako drugi argument. Proces kopiowania musi spełniać określone warunki:
   1. Jeżeli plik jest tekstowy, to wykorzystaj jedną z metod operującą na znakach,
   2. Jeżeli jest binarny (np. zdjęcia), to skorzystaj z metod zapisu/odczytu bajtów,
   3. Jeżeli jego wielkość przekracza 4kB, to skorzystaj z buforowania
   4. Jeżeli jest mniejszy, niż 4kB, to całość pobierz do pamięci przed zapisem.
2. Pamiętaj o obsłużeniu błędów:
   - Nieprawidłowa ścieżka
   - Pusty plik


JDBC/ORM

JDBC

- API do zarządzania bazą danych z poziomu aplikacji Javowej
- Implementacja dostarczana przez sterownik
- Konfiguracja przez obiekt klasy `DataSource`

Zadanie

1. Otwórz moduł `database-access`
2. W oparciu o klasy w paczce `jdbc.model` stwórz (z wykorzystaniem Workbench-a) model bazy danych, który odzwierciedla strukturę klas
3. Uzupełnij w pliku `database.properties` właściwości do połączenia z utworzoną bazą danych
4. Uzupełnij implementacje `StudentDao` i `CourseDao`

Hibernate

- Implementacja ORM i JPA
- Konfiguracja przez xml lub adnotacje
- `hibernate.cfg.xml` – gdy korzystamy z czystego Hibernate'a
- `pesistence.xml` – gdy korzystamy z implementacji JPA

Zadanie

1. Otwórz moduł `database-access`
2. Dodaj odpowiednie adnotacje w klasach pakietu `hibernate.entity` tak, aby korespondowały ze strukturą bazy z poprzedniego zadania
3. Uzupełnij plik `persistence.xml`, aby połączyć się z bazą
4. Uzupełnij implementacje `StudentDao` i `CourseDao`