FOI praksa radni listić

TIS 2024.

Sadržaj

FOI praksa radni listić 1

Uvod 1

1. Model 2
2. Repository sučelje/interface i implementacija 2
3. Pisanje datoteka na datotečni sustav 4
4. Čitanje datoteke s datotečnog sustava 5
5. Implementacija ProductRepository koristeći datotečni sustav 6
6. BONUS - Implementacija ProductRepository koristeći bazu podataka 7

Web Scraping 9

7. Dohvaćanje podataka sa stranice 9
8. ProductService 10
9. Konzola aplikacije 10
10. BONUS - frontend 11

Uvod

Radni listić sadrži upute za implementaciju aplikacije za dohvat i spremanje proizvoda. Aplikacija se zove *product-manager* i za implementaciju ćemo koristiti Javu 21, Maven, biblioteke org.jsoup i com.h2database.

Podijelit ćemo se u dva tima.

Početna aplikacija se nalazi na linku https://github.com/ivanjukic237/foi-praksa. Potrebno je napraviti git *checkout* repozitorija i svaki tim treba napraviti zaseban *branch* na kojem će pushati svoj kod. Predlažem da svaka osoba u tima ima i zasebni branch, ali nije nužno. Predlažem još da se prije pushanja ili mergeanja u branch odradi *code review* koda koji se pusha.

Upute su dosta detaljne, ali u svakom trenutku možete tražiti pomoć ako zapnete. Koristite Google i dokumentaciju prilikom implementacije. Zabranjeno je korištenje LLM-ova (ChatGPT i slično).

Bitno

Pazite na nazive klasa, metoda i paketa. Paketi se nazivaju malim slovima, za metode, polja i klase koristimo *camelCase*, a za nazive klasa *CamelCase*.

Svi datumi u aplikaciji će nam se prikazivati i spremati u formatu yyyy-MM-dd.

Koristite ctrl + alt + L za formatiranje koda dok kodirate.

1. Model

a) Napravite klase hr.tis.praksa.model.ProductsMetadata i hr.tis.praksa.model.Product. Klasa Product će sadržavati naziv proizvoda, cijenu proizvoda (nemojte koristiti double, pronađite klasu u Javi koja se koristi za računanje s iznosima), mjernu jedinicu (npr. EUR/kom.) i ocjenu koju proizvod ima (broj zvjezdica). Ako proizvod nitko još nije ocijenio, onda stavljamo da je ocjena = 0.

ProductsMetadatas klasa će sadržavati id (Long vrijednost), datum kreiranja (LocalDate), naslov i listu proizvoda.

Sva polja neka budu privatna, koristite gettere i settere (Intellij ih može sam generirati). Napravite defaultni konstruktor i konstruktor sa svim poljima klase (Intellij ih isto može sam generirati).

b) Overrideajte metodu *toString* klase ProductsMetadata (za to isto možete koristiti Intellij). Koristite anotaciju *@Override* iznad metode.

c) U klasi ProductsMetadata napravite *main* metodu i u njoj **inicijalizirajte** jedan objekt ProductsMetadatas (ProductsMetadata productMetadata = new ProductsMetadatas();) koji će sadržavati dva objekta Product. Isprintajte objekt na konzolu koristeći System.out.println(). Trebate li eksplicitno pozivati metodu toString()? Kako se printa lista proizvoda? Overrideajte i toString() metodu Products klase pa ponovno printajte.

PITANJA:

1. Kako zovemo relaciju ProductsMetadata – Products ako te dvije klase gledamo kao na tablice u bazi podataka?
2. Zašto ne koristimo float tipove podataka za novčane iznose?

2. Repository sučelje/interface i implementacija

a) Napravite sučelje hr.tis.praksa.repository.ProductRepository.

```
public interface ProductRepository {

    Long insertProducts(ProductsMetadata ProductsMetadata);

    BigDecimal fetchSumOfPrices(LocalDate createdDate);

    BigDecimal fetchSumOfPrices(Long id);

    ProductsMetadata fetchProductsMetadata(LocalDate createdDate);

    ProductsMetadata fetchProductsMetadata(Long id);

    Integer fetchProductsMetadataCount();

    // dodati i implementirati defaultnu metodu
    // BigDecimal calculateSumOfPrices(List<Product> products)
```

```
}
```

b) Implementirajte sučelje ProductRepository tako da u istom paketu napravite klasu ProductRepositoryInMemory. Koristite ključnu riječ *implements*. Kompajler će nam javiti da moramo implementirati metode iz sučelja. Sučelje nam koristi kao popis/ugovor kojim garantiramo da će svaka klasa koja ga implemetirati sadržavati iste metode.

   Iskoristite Intellij da vam kreira metode klase.

   Dodajte u klasu polje private **static final** List<ProductsMetadata> productsMetadataList = new ArrayList<>();. U tu listu ćemo spremati proizvode, dohvaćati i pomoću nje računati sumu iznosa svih proizvoda. Proučite čemu služe ključne riječi *static* i *final*.

c) Definicija metoda koje treba implementirati. Potencijalni duplicirani kod u metodama probajte izdvojiti u private metode.

   public Long insertProducts(ProductsMetadata ProductsMetadata);
   Metoda dodaje objekt ProductsMetadata u listu, postavlja ID objekta kao redni broj u listi (počinjemo od 1) i vraća ID ProductsMetadata objekta.

   public BigDecimal fetchSumOfPrices(LocalDate createdDate)
   Metoda dohvaća jedan ProductsMetadata iz liste za zadani datum kreiranja tako da vraća najnovije dodani objekt. Metoda zatim zbraja sve vrijednosti iz liste proizvoda koje ProductsMetadata ima i vraća taj iznos.

   public BigDecimal fetchSumOfPrices(Long id)
   Metoda dohvaća jedan ProductsMetadata iz liste za zadani ID. Metoda zatim zbraja sve vrijednosti iz liste proizvoda koje ProductsMetadata ima i vraća taj iznos.

   public ProductsMetadata fetchProductsMetadata(LocalDate createdDate)
   Metoda dohvaća jedan ProductsMetadata iz liste za zadani datum kreiranja tako da vraća najnovije dodani objekt.

   public ProductsMetadata fetchProductsMetadata(Long id)

   Metoda dohvaća jedan ProductsMetadata iz liste za zadani ID.

   public Integer fetchProductsMetadataCount()

   Metoda vraća broj koliko ProductsMetadata imamo.

d) U slučaju da ne možemo naći ProductsMetadata za ID ili createdDate, bacamo RuntimeException s porukom “Record doesn’t exist.”. Koristite ključnu riječ *throws*.

   Za testiranje možete u toj klasi napraviti main metodu i probati implementacije.

   PITANJA:
   1) Koji su problem ovakve implementacije “baze”? Što će se dogoditi ako nam program bude radio jako dugo?

3. Pisanje datoteka na datotečni sustav

a) Napravite klasu hr.tis.praksa.file.FileSystemConfiguration

```
package hr.tis.praksa.file;

import java.nio.file.Path;

public class FileSystemConfiguration {

    public static final Path PRODUCTS_FILES_FOLDER_PATH =
Path.of("products");

}
```

Primijetite kako nazivamo konstante u Javi – CAMEL_CASE.

b) Napravite klasu hr.tis.praksa.file.ProductWriter koja će nam koristiti za pisanje proizvoda na datotečni sustav. Klasa će imati samo jednu metodu *public static void writeProducts(ProductsMetadata ProductsMetadata)*.

c) Naziv datoteke će nam biti informacije iz ProductsMetadata objekta: createdTime_id_title.txt (npr. 2024-06-01_2_naslov.txt). Pogledajte String.format() metodu.

d) Kod pisanja datoteke koristite:

```
try (BufferedWriter writer = Files.newBufferedWriter(

FileSystemConfiguration.PRODUCTS_FILES_FOLDER_PATH.resolve(fileName))
) {
    // TODO pisanje proizvoda u datoteku
} catch (IOException e) {
    throw new RuntimeException(e);
}
```

Proučite ovu sintaksu, nemamo klasični try – catch blok s kojim ste možda više upoznati.

Unutar try – catch bloka zapišite sve proizvode (listu proizvoda od ProductsMetadata) u datoteku tako da je formatirate na sljedeći način:

Slog	Duljina
Naziv proizvoda (lijevo poravnato)	100
Cijena proizvoda (desno poravnato)	10
Mjerna jedinica (lijevo poravnato)	10
Ocjena	1

Primjer jednog retka (nisu realne duljine slogova):

Čokolino 1kg 7.19EUR/kg 5

Od Č do 7 imamo 100 znakova. Za padding koristimo razmake.

Koristite String.format() metodu! Proučite kako možete to napraviti. Guglajte String.format padding.

PITANJA:

1) Koji je benefit ovakvog try – catch bloka?
2) Gdje nam se sprema datoteka s ovom implementacijom?
3) Na koja dva načina možemo pozvati statičnu metodu neke klase?

4. Čitanje datoteke s datotečnog sustava

a) Napravite klasu hr.tis.praksa.file.ProductReader koja će nam koristiti za čitanje proizvoda s datotečnog sustava. Klasa će imati samo jednu metodu *public static ProductsMetadata read(String fileName)*.

Kod čitanja datoteke koristite ovu sintaksu:

```
try (BufferedReader reader =
Files.newBufferedReader(FileSystemConfiguration.PRODUCTS_FILES_FOLD
ER_PATH.resolve(fileName))) {
    String line;
    while ((line = reader.readLine()) != null) {
        // TODO mapiranje proizvoda
    }
} catch (IOException e) {
    throw new RuntimeException(e);
}
```

Proučite ovu sintaksu, nemamo klasični try – catch blok s kojim ste možda više upoznati.

Unutar try – catch bloka zapišite sve proizvode (listu proizvoda od ProductsMetadata) u datoteku tako da je formatirate na sljedeći način:

Slog	Duljina
Naziv proizvoda (lijevo poravnato)	100
Cijena proizvoda (desno poravnato)	10
Mjerna jedinica (lijevo poravnato)	10
Ocjena	1

Primjer jednog retka (nisu realne duljine slogova):

Čokolino 1kg 7.19EUR/kg 5

Od Č do 7 imamo 100 znakova. Za padding koristimo razmake.

b) Iz naziva datoteke izvadite Id, naziv i datum kreiranja. Primjer formata naziva datoteke je DATUM_ID_NASLOV.txt datoteke, npr. 2024-06-01_2_naslov.txt.

PITANJA:

1) Koji je benefit ovakvog try – catch bloka?
2) Gdje nam se sprema datoteka s ovom implementacijom?
3) Na koja dva načina možemo pozvati statičnu metodu neke klase?

5. Implementacija ProductRepository koristeći datotečni sustav

a) Implementirajte Sučelje ProductRepository tako da napravite klasu hr.tis.praksa.repository.ProductRepositoryFile po istoj specifikaciji iz zadatka 3., samo u ovom slučaju ćete za pohranu ProductsMetadata objekta koristiti klasu ProductWriter, a za dohvat ProductsMetadata objekta klasu ProductReader.

Svaka nova datoteka koja se dodaje mora imati sekvencijalni id. Ako imamo već spremljene datoteke 2024-06-04_1_naziv.txt, 2024-06-04_2_naziv.txt i 2024-06-04_3_naziv.txt, sljedeća datoteka mora imati ID = 4. Za dohvat broja datoteka u direktoriju koristite:

```
File directory =
FileSystemConfiguration.PRODUCTS_FILES_FOLDER_PATH.toFile();
```

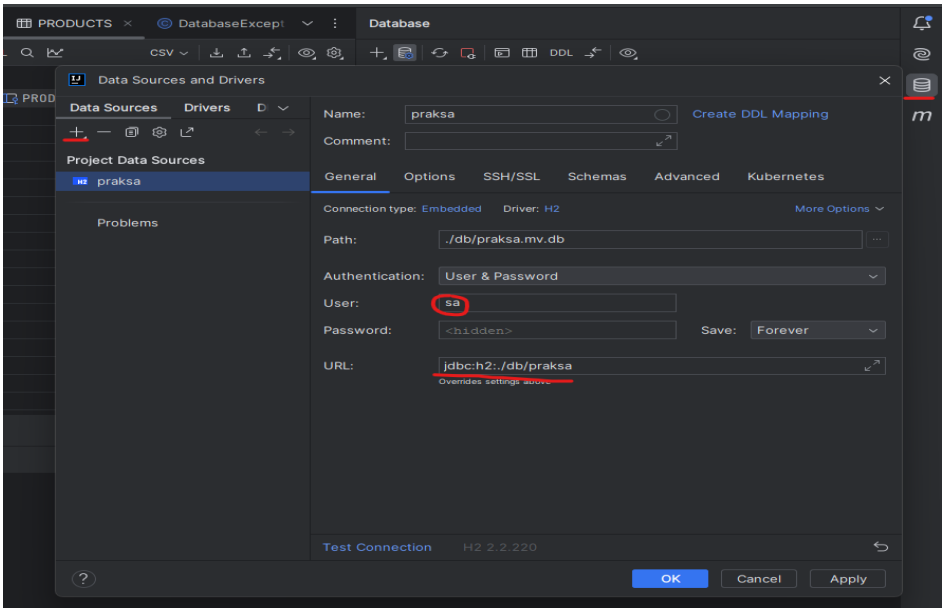
Proučite metode koje biblioteka java.io.File ima da dobijete broj datoteka u direktoriju. Pretpostavite da nećemo nikada brisati datoteke u direktoriju.

PITANJA:

1) Koji su problemi ovakve implementacije “baze”?

6. BONUS - Implementacija ProductRepository koristeći bazu podataka

a) Proučite klasu hr.tis.praksa.db.Database, specifično metodu init().
Napravite main klasu unutar Database klase. Pokrenite init() metodu. Aplikacija će kreirati lokalnu bazu koju ćete moći pristupiti preko konzole unutar Intellij-a. Nakon toga namjestite u Intellij-u konekciju na bazu. S desne strane odaberite Database gumb. Dodajte H2 bazu sa sljedećim postavkama.



b) Pronađite datoteku init.sql u resources direktoriju. Pogledajte kako je napravljena tablica KORISNICI. Na isti način napravite SQL skripte za kreiranje tablica PRODUCTS_METADATA i PRODUCTS. Razmislite koja je relacija između te dvije tablice i stavite FOREIGN KEY na ispravno mjesto.

c) Kreirajte klasu hr.tis.praksa.repository.ProductRepositoryDB koja implementira ProductRepository.

d) Primjer poziva baze, proučite dohvat s parametrima:

```
public String fetchKorisnik(Long id) {
    String querySQL = "SELECT * FROM KORISNICI WHERE ID = ?";

    try (Connection connection = Database.getInstance().getConnection();
         PreparedStatement preparedStatement =
connection.prepareStatement(querySQL)) {

        preparedStatement.setLong(1, id);

        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next())
            return String.format("%s %s", resultSet.getLong("ID"),
resultSet.getString("NAME"));
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
```

```
    return null;
}
```

dohvat bez parametara:

```
public List<String> fetchAll() {
    String querySQL = "SELECT * FROM KORISNICI";

    List<String> users = new ArrayList<>();

    try (Connection connection = Database.getInstance().getConnection();
         Statement statement = connection.createStatement()) {

        ResultSet resultSet = statement.executeQuery(querySQL);

        while (resultSet.next()) {
            Long id = resultSet.getLong("ID");
            String name = resultSet.getString("NAME");
            users.add(String.format("%s %s", id, name));
        }
    } catch (Exception e) {
        throw new DatabaseException(e);
    }
    return users;
```

e) Dodajte implementaciju za spremanje ProductsMetadata s listom Product objekata. Dodajte mapiranja za ocjenu i mjernu jedinicu:

```
public Long insertProducts(ProductsMetadata ProductsMetadata) {
    String recordSQL = "INSERT INTO PRODUCTS_METADATA (CREATED_TIME, TITLE) VALUES (?,
?)";
    String productSQL = "INSERT INTO PRODUCTS (NAME, PRICE, PRODUCTS_METADATA_ID)
VALUES (?, ?, ?)";

    long recordId;

    try (Connection connection = Database.getInstance().getConnection()) {
        connection.setAutoCommit(false);

        try (PreparedStatement recordStmt = connection.prepareStatement(recordSQL,
Statement.RETURN_GENERATED_KEYS)) {
            recordStmt.setDate(1, Date.valueOf(ProductsMetadata.getCreatedTime()));
            recordStmt.setString(2, ProductsMetadata.getTitle());
            recordStmt.executeUpdate();

            try (ResultSet generatedKeys = recordStmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    recordId = generatedKeys.getLong(1);

                    try (PreparedStatement productStmt =
connection.prepareStatement(productSQL)) {
                        for (Product product : ProductsMetadata.getProducts()) {
                            productStmt.setString(1, product.getName());
                            productStmt.setBigDecimal(2, product.getPrice());
                            productStmt.setLong(3, recordId);
                            productStmt.addBatch();
                        }
                        productStmt.executeBatch();
                    }
                } else {
                    throw new RuntimeException("Creating PRODUCTS_METADATA failed, no
ID obtained.");
                }
            }
        }
        connection.commit();
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
```

```
    return recordId;
}
```

Zbog jednostavnosti, za dohvat ProductsMetadata retka s listom Product redaka, dohvatite svaki pojedinačno. Nemojte raditi JOIN.
Metodu BigDecimal fetchSumOfPrices() riješite koristeći jedan upit, nemojte zbrajati u Javi.

PITANJA:

1) Objasnite što je to transakcija i gdje možemo vidjeti transakciju u primjeru metode insertProducts(ProductsMetadata ProductsMetadata).
2) Napišite u SQL-u jedan JOIN PRODUCTS tablice na PRODUCTS_METADATA koristeći Intellij.
3) Koji se oblikovni obrazac (eng. Design pattern) koristi u klasi Database? Čemu služi privatni konstruktor?

Web Scraping

Aplikacija će koristiti web scraping za dohvat podataka o proizvodima koje ćemo spremati i prikazivati. Više na linku https://en.wikipedia.org/wiki/Web_scraping.

Stranica koju ćemo koristiti je https://www.konzum.hr/web/posebne-ponude. Proučite stranicu i kako **paginacija** funkcionira, odnosno kako se URL-ovi stranica mijenjaju tako da na dnu stranice mijenjate stranice na kojima se nalaze proizvodi.

Predlažem da koristite Chrome i opciju Inspect element (desni click mišem na proizvod – inspect element) da pronađete pravila kako se HTML elementi organizirani.

Za dohvat podataka ćemo koristiti org.jsoup biblioteku (link na dokumentaciju: https://jsoup.org/). Pogledajte datoteku pom.xml u repozitoriju da vidite kako se bibiloteka dohvaća.

7. Dohvaćanje podataka sa stranice

a) napravite klasu hr.tis.praksa.scraper.WebScraper

klasa će imati jednu metodu public ProductsMetadata fetchProducts(). Koristite jsoup za skidanje podataka sa stranice. Napravite implementaciju koja će proći kroz sve stranice linka https://www.konzum.hr/web/posebne-ponude, ali napravite da maksimalno povuče podatke s 3 stranice.

Za ProductsMetadata mapiranje: za createdDate stavite trenutni datum, a naziv stavite vrijednost iz </title> elementa stranice.

8. ProductService

a) Napravite sučelje hr.tis.praksa.service.ProductService i klasu hr.tis.praksa.service.impl.ProductServiceImpl.

b) Napravite metode za dohvat ProductsMetadata klase direktno s web-a bez spremanja

c) metodu za spremanje proizvoda s web-a

d) metodu za dohvaćanje proizvoda za zadani datum.

Neka klasa ima sljedeća polja i konstruktor.

```
private final ProductRepository productRepository;
private final WebScraper webScraper = new WebScraper ();

public ProductServiceImpl(ProductRepository productRepository) {
    this.productRepository = productRepository;
}
```

PITANJA:

1) Koji nam je benefit toga što u klasi ProductServiceImpl imamo ProductRepository interface umjesto npr. ProductServiceInMemory?

9. Konzola aplikacije

Napravite klasu hr.tis.praksa.console.ProductsConsole koja će imati main metodu i primat će input s konzole.

Kod pokretanja aplikacije nam se prikazuje:

```
System.out.println("Welcome to FOI product-manager app!\nStart with 'help'
command to see the list of commands.");
```

Ako se upiše help u konzolu prikazujemo:

```
System.out.println("Choose how to save the products from commands: (file,
in-memory, db)\n" +
        "Fetch products from web: fetch-products\n" +
        "Fetch products for date in format yyyy-MM-dd: fetch-products
{date}\n" +
        "Fetch and save products: save-products\n" +
        "Exit console: exit");
```

U slučaju da nijednom nije odabrana jedna od naredbi file, in-memory ili db, printamo sljedeće:

```
System.out.println("Choose how to save the products from commands: (file,
in-memory, db)");
```

Odabirom naredbe file koristimo implementaciju ProductRepositoryFile, odabirom naredbe in-memory koristimo implementaciju ProductRepositoryInMemory, a ukoliko ste riješili zadatak s bazom, za db koristimo ProductRepositoryDB, inače napišite da opcija nije moguća.

Razmislite kako ćete mijenjati implementaciju repozitorija koristeći konstruktor klase ProductServiceImpl.

Uredite toString metode od Product i ProductsMetadata da nam se ispisuje lijepo na konzolu.

Za popis naredbi možete, ali ne morate koristiti *enum*.

Konzola se ne smije zaustaviti ako se baci exception.

Ispišite poruku ako se unese nepostojeća naredba.

Ispišite poruku ako unese datum u krivom format.

10. BONUS - frontend

Java u sebi ima integriran HttpServer koji možemo koristiti za razgovor s Internetskim preglednikom. U response sljedeće klase možemo poslati ili tekst ili html i naš preglednik će nam to prikazati.

```
public class ProductsController {

    private final ProductRepositoryDB productRepositoryDB = new
ProductRepositoryDB();

    public void createContext(HttpServer server) {
        server.createContext("/hello", new HelloHandler());
        // TODO ostali handleri
    }

    private class HelloHandler implements HttpHandler {
    @Override
    public void handle(HttpExchange exchange) throws IOException {
        String response = """
        <!DOCTYPE html>
        <html>
        <head>
            <meta charset="UTF-8">
            <title>FOI praksa</title>
            <style>
                body {
                    font-family: Arial, sans-serif;
                    background-color: #f0f0f0;
                    text-align: center;
                    padding-top: 50px;
                }
                h1 {
                    color: #333;
                }
            </style>
        </head>
        <body>
            <h1>Bok!</h1>
            <p>Šaljemo nekakav HTML koji nam se rendera na stranici :)</p>
        </body>
        </html>
        """;

        exchange.getResponseHeaders().set("Content-Type", "text/html;
charset=utf-8");

        try {
            byte[] bytesResponse =
response.getBytes(StandardCharsets.UTF_8);
            exchange.sendResponseHeaders(200, bytesResponse.length);
```

```
            OutputStream os = exchange.getResponseBody();
            os.write(bytesResponse);
            os.close();
        } catch (Exception e) {
            e.printStackTrace();
            byte[] errorResponse = "Internal Server Error
:(".getBytes(StandardCharsets.UTF_8);
            exchange.sendResponseHeaders(500, errorResponse.length);
            OutputStream os = exchange.getResponseBody();
            os.write(errorResponse);
            os.close();
        }
    }
}
```

```
            } catch (Exception e) {
                e.printStackTrace();
                response = "Interna pogreška
:(".getBytes(StandardCharsets.UTF_8);
                exchange.sendResponseHeaders(500, response.length);
            }

            OutputStream os = exchange.getResponseBody();
            os.write(response);
            os.close();
        }
    }

}
```

Pokretanje servera:

```
public class App {


    public static void main(String[] args) throws IOException {
        HttpServer server = HttpServer.create(new InetSocketAddress(8080),
0);


        ProductsController productsController = new ProductsController();
        productsController.createContext(server);

        server.start();
        System.out.println("Server started on port 8080.");

    }

}
```