2024

Rapport - Objektorientert programmering

INTRODUKSJON

Dette prosjektet innebærer utvikling av en programvareløsning for å administrere data om historiske gjenstander funnet av en gruppe hobbyarkeologer. Programvaren importerer data fra en tekstfil til en database og gir funksjonalitet for å søke og administrere disse dataene gjennom et brukergrensesnitt. Prosjektet demonstrerer viktige objektorienterte programmeringskonsepter, inkludert unntakshåndtering, innkapsling, arv og polymorfisme.

PROSJEKTSTRUKTUR

Prosjektet består av følgende komponenter:

1. DatabaseManager.java: Håndterer databaseforbindelser og operasjoner.
2. Funngjenstand.java: Abstrakt basisklasse for gjenstander.
3. Mynt.java, Våpen.java, Smykke.java: Konkrete klasser for forskjellige typer gjenstander.
4. Person.java: Representerer en person tilknyttet en gjenstand.
5. Museum.java: Representerer et museum hvor gjenstander kan lagres.
6. Main.java: Inneholder hovedprogramlogikken, inkludert brukerinteraksjon og dataimport.
7. funn.sql: SQL-skript for å opprette databaseskjemaet.
8. funn.txt: Eksempelfil for import av gjenstander.

Unntakshåndtering

Unntakshåndtering er brukt for å sikre at programmet er sterkt og pålitelig, spesielt når det leser filer, gjør databaseoperasjoner og behandler brukerinput.

Databaseoperasjoner

Unntakshåndtering under databaseoperasjoner sikrer at applikasjonen kan håndtere SQL-feil på en smidig måte. Klassen DatabaseManager viser dette med try-catch-blokker rundt SQL-operasjoner:

```
java

public DatabaseManager() {
    try {
        connection = DriverManager.getConnection(URL, USER, PASSWORD);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}


public void insertPerson(Person person) {
    String sql = "INSERT INTO personer (id, navn, telefonnummer, epost)
VALUES (?, ?, ?, ?)";
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.setInt(1, person.getId());
        statement.setString(2, person.getNavn());
        statement.setString(3, person.getTelefonnummer());
        statement.setString(4, person.getEpost());
        statement.executeUpdate();
```

```
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

FILOPERASJONER

Unntakshåndtering under filoperasjoner sikrer at applikasjonen kan håndtere IO-feil på en smidig måte. For eksempel, metoden importData i klassen Main

```
private static void importData(String filename, DatabaseManager dbManager)
{
    try (BufferedReader br = new BufferedReader(new FileReader(filename)))
{
        String line;
        // Reading and processing the file content
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

INKAPSLING

Innkapsling brukes for å beskytte objektenes interne tilstand og for å gi et kontrollert grensesnitt for samhandling med disse objektene.

EKSMEPEL: FUNNGJENSTAND-KLASSE

Klassen Funngjenstand kapsler inn egenskapene til en gjenstand ved å bruke private felt og offentlige getter-metoder:

```
java
public abstract class Funngjenstand {
    private int id;
    private String koordinater;
    private int personId;
    private String dato;
    private int arstall;
    private Integer museumId;

    // Constructors, getters, and abstract methods
}
```

ARV OG POLYMORFISME

Arv og polymorfisme brukes for å modellere forskjellige typer gjenstander. En basisklasse Funngjenstand er opprettet, og spesifikke gjenstandstyper (f.eks. Mynt, Våpen, Smykke) utvider denne basisklassen.

BASIS-KLASSE: FUNNGJENSTAND

Klassen Funngjenstand inneholder vanlige egenskaper og metoder for alle gjenstander.

```
java
public abstract class Funngjenstand {
    private int id;
    private String koordinater;
    private int personId;
    private String dato;
    private int arstall;
    private Integer museumId;

    // Constructors, getters, and abstract methods
}
```

UNDERKLASSE: MYNT, VÅPEN, SMYKKE

Underklasser utvider Funngjenstand og legger til spesifikke egenskaper.

```
java
public class Mynt extends Funngjenstand {
    private int diameter;
    private String metall;

    public Mynt(int id, String koordinater, int personId, String dato, int
arstall, Integer museumId, int diameter, String metall) {
        super(id, koordinater, personId, dato, arstall, museumId);
        this.diameter = diameter;
        this.metall = metall;
    }

    @Override
    public String getType() {
        return "Mynt";
    }

    @Override
    public String getEgenskaper() {
        return "Diameter: " + diameter + ", Metall: " + metall;
    }
}
```

POLYMORFISME

Polymorfisme gjør det mulig å behandle objekter av forskjellige underklasser som objekter av basisklassen Funngjenstand. Dette er nyttig når man behandler samlinger av gjenstander.

```
java
List<Funngjenstand> artifacts = new ArrayList<>();
artifacts.add(new Mynt(...));
artifacts.add(new Våpen(...));
artifacts.add(new Smykke(...));

for (Funngjenstand artifact : artifacts) {
    System.out.println(artifact.getEgenskaper());
}
```

DATA IMPORT

Klassen Main inneholder metoden importData som leser data fra filen funn.txt og setter dem inn i databasen ved hjelp av klassen DatabaseManager. Fillesingsprosessen bruker unntakshåndtering for å sikre at eventuelle IO-feil blir fanget opp og håndtert på riktig måte.

EKSEMPEL AV DATA IMPORT

```
java
private static void importData(String filename, DatabaseManager dbManager)
{
    try (BufferedReader br = new BufferedReader(new FileReader(filename)))
{
        String line;
        // Reading and processing the file content
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

BRUKERGRENSESNITT

Klassen Main gir et enkelt tekstbasert brukergrensesnitt for interaksjon med databasen. Brukere kan vise alle gjenstander, vise gjenstander eldre enn et spesifisert år, eller få antall registrerte gjenstander. Grensesnittet bruker en Scanner for brukerinput og tilbyr en løkke med en switch-case-setning for menyvalg.

EKSEMPEL AV BRUKERGRENESNITT

```
java
public static void main(String[] args) {
    DatabaseManager dbManager = new DatabaseManager();
    importData("funn.txt", dbManager);

    Scanner scanner = new Scanner(System.in);
    boolean running = true;
```

```
    while (running) {
        System.out.println("Velkommen til historiske gjenstander
systemet!");
        System.out.println("Vennligst velg et alternativ:");
        System.out.println("1. Se informasjon om alle funngjenstander");
        System.out.println("2. Se informasjon om alle funngjenstander eldre
enn et gitt årstall");
        System.out.println("3. Få informasjon om antall funngjenstander
registrert");
        System.out.println("4. Avslutte programmet");

        int valg = scanner.nextInt();
        scanner.nextLine(); // Konsumerer den resterende nye linjen

        switch (valg) {
            case 1:
                visAlleFunngjenstander(dbManager);
                break;
            case 2:
                System.out.println("Vennligst skriv inn årstall:");
                int arstall = scanner.nextInt();
                visFunngjenstanderEldreEnn(dbManager, arstall);
                break;
            case 3:
                visAntallFunngjenstander(dbManager);
                break;
            case 4:
                running = false;
                System.out.println("Programmet avsluttes. Ha en fin dag!");
                break;
            default:
                System.out.println("Ugyldig valg, vennligst prøv igjen.");
                break;
        }
    }

    scanner.close();
}
```

FORUTSETNINGER OG KLARIFISERINGER

Forutsetninger

- Dataformat: Formatet på funn.txt følger strukturen som er gitt i oppgaven. Eventuelle avvik eller feil i filformatet vil resultere i unntak.
- Databaseskjema: Databaseskjemaet som er gitt i funn.sql brukes uten modifikasjoner. Hvis justeringer er nødvendige, er de dokumentert i kodekommentarene.
- Brukerinput: Det antas at brukeren vil gi gyldige input der det blir bedt om det. Inputvalidering og feilhåndtering er implementert for å håndtere vanlige tilfeller av ugyldig input.

Klarifiseringer

- Filhåndtering: Det antas at filen funn.txt er plassert i samme katalog som det kjørbare programmet.

- Databaseforbindelse: Parametere for databaseforbindelse (URL, bruker, passord) er hardkodet for enkelhets skyld. I en reell situasjon bør disse overføres til en konfigurasjonsfil eller miljøvariabler.

KONKLUSJON

Dette prosjektet demonstrerer bruken av viktige objektorienterte programmeringskonsepter, inkludert unntakshåndtering, innkapsling, arv og polymorfisme. Løsningen gir en kraftig og vedlikehold bar kodebase for å administrere data om historiske gjenstander, og sikrer påliteligheten. Bruken av et tekstbasert brukergrensesnitt gjør det enkelt å samhandle med databasen, noe som gjør systemet tilgjengelig og brukervennlig.