Rapport til eksamen i PGR112 våren 2024 • Kandidatnr. 41

Innkapsling

I min kode har jeg brukt innkapsling gjennom å sikre at alle felt og alle metoder som ikke brukes direkte av andre klasser er satt til private. Det er også satt opp en egen klasse som fungerer som et program som kjøres fra Main-klassen, `RelicsProgram`, istedenfor å instansiere flere klasser i Main.

Siden kun `startProgram()` fra `RelicsProgram` er tilgjenglig fra Main er også all annen data inni programmet innkapslet og utilgjengelig for brukeren av programmet utenom direkte gjennom menyene.

```
11      //# Fields
12      private final DataHandler data;  14 usages
13      private final static Scanner input = new Scanner(System.in);  1 usage
14
15      //# Constructor
16  >   public RelicsProgram (DataHandler data) { this.data = data; }
19
20      //# Methods
21  >   public void startProgram() {...}
37
38  >   private void showMenu() {...}
70
71  >   private void menu1_printInfoAboutAllItems() {...}
82  >   private void menu2_printInfoBasedOnAge() {...}
96  >   private void menu3_printInfoBasedOnNumbers() {...}
```

Arv

Jeg har brukt arv gjennom en abstrakt klasse i min besvarelse som heter `FoundItem`. Denne inneholder feltene for data som er lik for alle de tre typene med gjenstander som blir lagret gjennom systemet (våpen, mynter og smykker), samt gettere for å hente ut data fra feltene.

```
public abstract class FoundItem  20 usages  3 inheritors
{
    //# Fields
    protected int id;  24 usages
    protected String placeDiscovered;  9 usages
    protected int finder_id;  15 usages
    protected String dateFound;  15 usages
    protected int expectedYearOfCreation;  15 usages
    protected int museum_id;  24 usages
    protected String type;  7 usages
```

De tre klassene som brukes for å lagre gjenstandene (`ItemCoin`, `ItemWeapon` og `ItemJewelry`) arver disse og legger også til egne felter og tilsvarende gettere som er unike for hver type gjenstand.

```
public class ItemCoin extends FoundItem  16 usages
{
    //# Fields
    private int diameter;  4 usages
    private String metal;  4 usages

    //# Constructor
    public ItemCoin(int id, String placeDiscovered, int finder_id, String dateFound,  1 usage
                    int expectedYearOfCreation, String type, int diameter, String metal) {
        super(id, placeDiscovered, finder_id, dateFound, expectedYearOfCreation, type);
        this.diameter = diameter;
        this.metal = metal;
    }
    public ItemCoin(int id, String placeDiscovered, int finder_id, String dateFound,  3 usages
                    int expectedYearOfCreation, int museum_id, String type, int diameter, String metal) {
        super(id, placeDiscovered, finder_id, dateFound, expectedYearOfCreation, museum_id, type);
        this.diameter = diameter;
        this.metal = metal;
    }
```

I noen metoder i `DataHandler`-klassen som håndterer data input og output til databasen og fra filscanneren brukes også arv ved at jeg har metoder som tar imot `FoundItem` som datatype. Så bruker jeg switch'er og for-løkker til å gi gjenstandene videre til metoder som typekonverterer (downcaster) til subklasser så jeg har tilgang på alle datafeltene.

Her kunne jeg tenkt meg og oppdatert mye av koden min da jeg ikke tenkte så mye på typekonvertering i begynnelsen og mange av mine første metoder ble veldig grove. Hadde jeg hatt mer tid til å finpusse på koden ville jeg optimalisert flere metoder til å typekonvertere mer.

```
11  public class DataHandler {  3 usages
390 @   private void printItemSwitch(FoundItem item) {  1 usage
391         switch (item.type) {
392             case "Mynt" -> printItemIfCoin(item);
393             case "Smykke" -> printItemIfJewelry(item);
394             case "Våpen" -> printItemIfWeapon(item);
395         }
396     }
397     private void printItemIfCoin(FoundItem item) {  1 usage
398         ItemCoin coin = (ItemCoin) item;
```

Unntak

Prosjektet mitt har hatt veldig enkel bruk av unntak (exceptions) i min kode. Jeg håndterer unntak direkte i metoden som oppretter forbindelsen med databasen istedenfor å håndtere de senere i koden.

Jeg holder meg også til å catche SQLExceptions eller kaste RuntimeExceptions når jeg kjører kode som programmet ikke vil fungere uten (som å laste inn data fra databasen ved oppstart).

```
public class DataHandler {  3 usages                                                    ⚠ 59
    private void loadPeopleFromDatabase() {  1 usage
        try (Connection connection = database.getConnection()) {
            peopleInDatabase.clear();

            Statement statement = connection.createStatement();

            ResultSet resultSet = statement.executeQuery( sql: "SELECT * FROM person");

            while (resultSet.next()) {
                var person = new Person(
                        resultSet.getInt( columnLabel: "id"),
                        resultSet.getString( columnLabel: "navn"),
                        resultSet.getInt( columnLabel: "tlf"),
                        resultSet.getString( columnLabel: "e_post")
                );
                peopleInDatabase.add(person);
            }
        } catch (SQLException e) {
            System.out.println("Could not load people from database");
            throw new RuntimeException(e);
        }
```

Forutsetninger for besvarelsen

Tekstfilen vil beholde strukturen i fremtidlige versjoner. For min besvarelse har jeg arbeidet ut ifra *funn.txt* filen er en autogenerert tekstfil fra et system som gruppen som casen omhandler har eksportert. Jeg finner det lite sannsynlig at en person manuelt har skrevet inn så mye tekst så strukturert.

Derfor er programmet jeg har kodet lagd med tanken at når gruppen i fremtiden finner nye gjenstander, så vil det eksporteres en ny *funn.txt* fil som deretter blir byttet ut med den gamle/utdaterte filen i resource-mappen. Den nye filen vil derfor ha den samme strukturen som filen jeg fikk utlevert: først alle personer, deretter alle museer og til slutt alle gjenstander.

Sensor vil kjøre sql-filene jeg har lagd i SQL-mappen før hen kjører programmet. Jeg har endret litt på koden fra *funn.sql* vi fikk utlevert og bl.a. skrevet alle kolonnenavn med små bokstaver (istedenfor stor), endret å i kolonnenavn til aa og endret datatypen på personers telefonnummer fra VARCHAR til INT da jeg ikke fant noen grunn til å lagre den som en String (det var også kun norske nummer i tekstfilen). Så om sensor kjører SQL-koden vi fikk utlevert eller bruker koden fra en besvarelse vil ikke koden min fungere.

Jeg vurderte også å endre DATETIME som brukes på funntidspunkt til DATE siden det ikke er noe klokkeslett lagret i funn.txt, men jeg hadde alt skrevet kode som fjernet klokkeslettet når jeg kom på dette og ville ikke endre på kode som fungerer når jeg har begrenset med tid.

Menyvalget for å se alle funngjenstander presiserer ikke hvilken data som skal vises. Derfor valgte jeg når det printes ut data om gjenstandene å ikke inkludere koordinater for lengde og breddegrader rundt hvor en gjenstand var funnet. I steden for har jeg valgt å printe ut informasjonen i en litt fortellende stil som passer å leses på en terminalskjerm.

Siden dette programmet kjører i en terminal så får man ikke gjort noe med det, så jeg tenker at det heller er data som kan skaffes på andre vis av en bruker som lurer på det.

```
Mynt #125/125 fra rundt år 1534 (ID: 198). Funnet av Maria Nygård i 2024.
- 36 mm i diameter og lagd av bronse.
- For øyeblikket utstilt på Finnetunet.

*** SMYKKER FUNNET ***
Smykke #1/36: Brosje fra rundt år 1567 (ID: 12). Funnet av Astrid Martinsen i 2024.
- Verdi estimert til 3234 kroner (se bilde: 119960192460.jpg).
- Ikke utstilt på museum for øyeblikket. Så ligger i safen på klubbhuset.
```

Sensor husker at dette er en 24 timers eksamen. Tenker det er greit å nevne at jeg synes det er en forutsetning at sensor husker at dette er en eksamen som skrives på kort tid, og selv om det er flere deler av eksamen som kunne vært optimalisert med bedre bruk av typekonvertering, up-/downcasting, arv og polymorfi, så er det jo ikke beregnet at vi skal sitte i 24 timer for å bli ferdig (selv om jeg tror mange ender opp med 15+ timer for å levere). Hadde jeg hatt et døgn til er det mange deler av koden min jeg ville gått over og endret når jeg ser på den nå som jeg er ferdig, men da blir jeg å måtte døgne og jeg tror ikke koden min blir noe bedre av det. Heller det motsatte, jeg tok meg selv i å lage tre metoder for en time siden jeg hadde lagd 6 timere tidligere og gitt andre navn, men det gikk meg helt hus forbi.