Eksamen PGR112 – Objektorientert programmering V2024

Rapport om håndtering av unntak, innkapsling og arv

- **Arv**: Ved å benytte arv forenkler jeg kodegjenbruken og spesialiseringen i underklasser uten å utsette felt for urelaterte klasser.

  I klassen «HistoricalObject» brukte jeg instansvaribelen ***protected*** for verdiene tilhørende «Id», «funnsted», «finnerId, «funntidspunkt», «antattårstall» og «museumId». Dette gjorde jeg fordi at disse verdiene skulle være tilgjengelige for subklassene mine «Mynt2», «Vaapen2» og «Smykke2». Ved å benytte denne instansvariabelen, kunne jeg gå inn i subklassene og få tilgang til, og endre på, disse verdiene.
  Disse verdiene er heller ikke tilgjengelig utenfor selve filpakken, og dette opprettholder et nivå av innkapsling samtidig som det tillater subklassespesialisering.
  Klassene «Myn2t», «Vaapen2» og «Smykke2» utvider selve «HistoricalObject» klassen, og arver de beskyttede verdiene direkte, eller gjennom *gettere* og *settere.*

- **Innkapsling**: Ved å benytte meg av instansvariabelen private, kan jeg sørge for at de ikke er tilgjengelige utenfor den aktuelle klassen, og sikrer på denne måten den interne tilstanden til objektet.
  I klassene «Person» og «Funn» brukte jeg private instansvariabler, for å sikre at verdiene som blir lagret her brukes kun i disse klassene. Ved å videre bruke **public** *settere* og *gettere* tillater jeg kontrollert tilgang til disse variablene, slik at eksterne klasser kan hente ut verdiene, uten å kunne manipulere de. Det kommer godt med når f.eks verdiene «Tlf» og «Navn» skal hentes ut fra klassen «Person», for å linkes videre til klassen «Funn», hvor en instansvariabel heter det samme: «Navn». Dette sikrer at verdiene som blir laget i respektive klasser, forblir der.

- **Unntak**: Ved å benytte **exceptions** kan jeg håndtere uventede forhold og feil.
  I «DatabaseManager» klassen min brukte jeg for eksempel exceptions for å håndtere en eventuell ved tilkoblingen til MySQL databasen. I dette eksempelet valgte jeg å bruke SQLException. Det gjorde jeg fordi jeg ville fange opp eventuelle feil som kan oppstå ved tilkobling til databasen, slik at jeg får informasjon om hva som eventuelt har gått feil. Dette kan for eksempel være feil database URL, eller noe så enkelt som feil brukernavn.
  I klassen min «Main» har jeg for eksempel brukt en FileNotFoundException, for å håndtere feil hvis programmet ikke kan finne den aktuelle filen.
  Jeg brukte også NumberFormatException for å håndtere feil ved parsing av Strings til tall, for eksempel i instansvariabelen «Funntidspunkt».

Forutsetninger jeg har lagt til grunn i løsningen min, og refleksjoner:

Jeg synes det var litt vanskelig å tolke hvorvidt man skulle legge filen «Funn.txt» på skrivebordet, og lese inn innholdet derifra og rett inn i MySQL via en scanner metode i IntelliJ, eller om man skulle kopiere innholdet i filen og lage en egen tekstfil i IntelliJ, for så å lage en scanner metode som leser innholdet videre til MySQL databasen.

Jeg tolket det som at man ikke skulle kopiere innholdet i filen «Funn.txt» til en egen tekstfil inne i IntelliJ, og jeg valgte derfor å lage en scannermetode som leste innholdet direkte fra filen på skrivebordet. Dette var derimot krevende, og jeg har strevet ekstremt lenge for å klare å sikre et skille mellom all informasjonen, slik som *Person* informasjon, *Funn* informasjon etc.

I ettertid ser jeg at det kanskje hadde vært lettere å lage egen tekstfil inne i IntelliJ og lese informasjonen fra der med scannermetode.

Jeg tror også at jeg burde ha lagt innloggingsdetaljene til MySQL i en egen *properties* fil, og lastet de videre inn i klassen «DatabaseManager», og ikke skrevet all den informasjonen som instansvariabler rett i klassen. Dette hadde nok vært tryggere og sikret at ikke alle kan lese mine påloggingsdetaljer.

Jeg ser nå i etterkant at når jeg kjører Main metoden, får jeg opp en feilmelding lydende: «Duplicate entry '1' for key 'museum.PRIMARY'».

Tiden er knapp, og med en 2-åring i hus og en bortreist kone, strekker tiden dessverre ikke til for å rette opp i disse forholdene. Jeg håper uansett at jeg har fått vist god kode og forståelse med det jeg har laget.