Almuthanna Jamal Aljajan

Teori

Oppgave 1 - Ord og begreper

Lag deg en oversikt over hva følgende ord/begreper/teknologier betyr/er:

- **Exception**

  En exception er en hendelse som "forstyrrer" den normale flyten i et programs instruksjoner mens programmet kjører. Det finnes tre typer exceptions:

  Checked Exceptions er forventede feil som kan oppstå. I utgangspunktet feil vi kan eller bør håndtere

  Runtime Exceptions (Unchecked Exceptions) indikerer feil som er internt i applikasjonen, som man vanligvis ikke skal forvente eller håndtere som regel feil i logikken i koden

  Errors (Unchecked Exceptions) er alvorlige feilsituasjoner for eksempel all lagringsplass er brukt opp eller feil med hardware.

- Tråd

  Thread klasse gir konstruktører og metoder for å lage og utføre operasjoner på et thread. Thread klasse arver fra Object-klassen og implementerer Runnable interface. Det er to måter å lage en thread på: ved å arve Thread-klassen eller ved å implementere Runnable interface. Runnable interface har bare en metode som heter run () og den brukes til å utføre handling for en tråd.

- Collections Framework
  - List

    List er et interface av Collection framework. List implementeres av klassene ArrayList, LinkedList, Vector og Stack. LinkedList er en del av collection framework i Java og den Implementerer List<> interfacet. En LinkedList er en liste med elementer slik som en ArrayList og vi i må definere objektstypen denne skal kunne holde på for eksempel LinkedList<Integer> tallLenke = new LinkedList<>(); Siden LinkedList implementerer List interfacet har også tilgang til metoder for å legge til add() og fjerne elementer i listen remove(). LinkedList lagrer ikke elementene i en intern Array. Derimot har alle elementer i en LinkedList en oversikt over hvilke elementer som står før og etter den selv i listen (next og previous). Vi

kan enkelt finne første og siste verdi i listen uten å vite hvor stor listen er ved å bruke getFirst() og getLast()

- HashMap

  En HashMap er en klasse i java som implementerer Map interfacet . En HashMap kan holde på 2 elementer som er koblet sammen (Key-Value pair). Når vi oppretter Map må vi definer objekttypen for begge elementene for eksempel HashMap<Integer, String> tallKart = new HashMap();. I motsetning til lister må en key være unik i settet, men en value kan vi ha flere ganger. Vi henter Values ved å si hvilken Key vi vi hente verdien fra. Vi henter Values ved å si hvilken Key vi vi hente verdien fra.

- Queue

  Queue er et interface og det finnes en rekke ulike queue implementasjoner i java som ArrayBlockingQueue. Queue er en lagringsstruktur som gir oss en mulighet til å kontrollere i hvilken rekkefølge man vil benytte innholdet i køen. En kø er delt opp i tre deler

  Head, dette er første elementet i køen

  Tail, siste elementet i køen

  Body, alle andre elementer i køen

  En kø baserer seg på prinsippet FIFO, First In First Out. Det elementet som først kom inn i køen skal være det første til å gå ut av køen for eksempel hvis vi har køen {1,2,3,4,5}, vi får ikke lov til å hente ut andre verdier enn 1. for å legge elementet i køen, bruker vi Offer(Object o) og bruker Poll() for å hente første elementet i en kø, denne fjerner også elementet i køen når den returneres. Men vi kan også kun se på headen i køen ved å benytte peek()

- Stack

  stack er egen en klasse og det baserer seg på LIFO prinsippet, Last In-First Out. Så en stack er en «bunke» med elementer. Nar vi legger til ett element i stacken kommer den sist(toppen) og når vi henter et element så tar vi alltid det siste(toppen) elementet først. Ved å bruke push(Object o), kan vi legge til elementer og vi henter det øverste elementet ved å bruke pop();. Til å se på det siste(øverste) elementet i en stack, bruker vi peek(); metode.

Oppgave 2 - Kodesammenligning

Oppgave 2.1

Oppgaven spurte om hvilke klasser som er naturlig å definere de som abstract, så i denne oppgaven jeg har definert CeletilaBody som abstract fordi vi skal aldri lage noe konkrete objekter av den.

Oppgave 2.2

I denne oppgaven, har implementerte Comparable interfacet i bygge CeletialBody og PlanetSystem klassene. I PlanetSystem spesifiserte jeg PlanetSystem som typen til objektene som skal sorteres og implementerte metoden compareTo som skal sammenligne med to objekter basert på navn. I CeletialBody spesifiserte jeg CeletialBody som typen til objektene som skal sorteres og implementerte metoden compareTo som skal sammenligne med to objekter basert på navn.

Oppgave 2.3

I denne oppgaven, laget jeg Gradle-project og i build-gradle, settet jeg opp dependencies som står i oppgaven. Deretter laget jeg Application-Klasse isteden av Main-Klasse. Så defilerte jeg et object av Javalin Klasse og starte webserveren ved å kalle Javalin.create().start(). Til slutt brukte jeg get metoden, og der definerte jeg pathen / og definere en ny Handle for å håndtere denne requesten ved å sende teksten «Hello, World!» som svar vha. result() metode.

Oppgave 2.4

I denne oppgaven, kopierte jeg vue-mappen og settet den under resources-mappen og jeg laget en model pakke og flyttet alle klassene hit unntatt Main.java

Oppgave 2.5

jeg laget en repository pakke og inni det laget jeg IUniverseRepository interfacet og definerte metodene getPlanetSystems som returnerer en liste med planetsystemer og metoden getPlanetSystem som returnerer en PlanetSystem basert på navn.

Deretter laget jeg **UniverseRepository**, som implementerer interfacet. Og inni denne klassen, definerte en liste med planetsystemer som database til mitt programmet. I konstruktøren opperettet jeg to planetsystemer og legget det til listen.Til slutt implementerte jeg metodene som definerte i interfacet.

Oppgave 2.6

I Application-klassen

For det Første:

Koplet jeg pathene med vue-filene ved hjelp av get metode, hvor først definert jege pathen og i andre parameter definerte jeg definerte jeg vue-komponent som skal vises når vi får en request på den pathen.

For det andre:

Laget jeg controller pakke og inni det definerte controller klassen. I klassen defierte jeg en referanse variabel til interfacet for å kunne ha tilgang til metodene som er definert der. Og laget en konstuktør som tar denne variabelen som parameter. Deretter laget jeg metodene som tar objekt av Context klassen som parameter. Den enne metoden skal returnere list med planetsystemer i json format. Og den andre skal returnere en PlanetSystem i json format også

Til slutt:

I Application-Klasse definerte jeg api pather vha. get metode. Og jeg settet controlleren for å svare på disse api requestene ved å kalle metodene som kan gir tilbake ønskene datene.

Oppgave 2.7

I denne oppgaven, vi skal gjøre nesten samme vi har gjørt i forrige oppgaver. Her skal vi jobbe med planeter og en planet, men ikke planetsystemer og en planetsystem

Oppgave 2.8

I denne oppgaven, definerte jeg en ny instanse variabel i bygge PlanetSystem og CeletialBodey klassene. Så mætte jeg å fike konstruktørene nedover i nedover i hierarkiet for å kunne sette denne verdien og laget jeg også set- og get- metoder for denne verdien. Tilslutt lagget jeg bilde-referanser til objektene som jeg har opprettet i konstruktøren til UniverseRepository klassen.

Oppgave 2.9

I siste oppgaven, I Controller klassen, I getPlanets metoden, hentet jeg verdien «sort-by» vha. queryParam metoden og laget jeg en liste med planeter. Deretter brukte jeg swich cases for å skjekke:

Hvis den den verdien er navn, så returnerer sortert liste basert på navn

Og hvis den den verdien er mass, så returnerer sortert liste basert på mass

Og hvis den den verdien er radius, så returnerer sortert liste basert på radius

Og hvis den den verdien er null, så returnerer sortert liste uten å bli sortert.

Kodesammenligning

det var faktisk to forskjeller mellom min løsning og løsningsforslag:

for den første, var at jeg har definert bare CeletialBody som abstract klassen, men i løsnninsforslag, var det to klasser som definert som abstract CeletialBody og NaturalSatellite.

For den andre, jeg har laget bare en Controller for begge planetsystemer og planeter , men i løsnninsforslag, var det to cotroller en for planetsystemer og den andre for planeter.