**Oblig 5**

**Oppgave 1**

Exception: En exception er en hendelse som forstyrrer den naturlige flyten i et programmert program. Det er tre typer exceptions, Checked exception, runtime exception og errors.

Checked exceptions er forventede feil som vi bør eller kan håndtere.

Runtime exceptions indikerer interne feil i programmet som somregel stammer av feil i logikken i kode.

Errors er alvorlige feil som ofte gjeler lagringsplass eller hardware feil.

Tråd: Tråder er at du kjører kode isolert parallelt med annen kode. Det er nyttig når du skal gjøre ting samtidig. Fordelen med tråder er at det krever mindre av datamaskinen enn å starte en helt ny prosess. Når vi skal bruke det i Java, så bruker vi extends thread eller implements runnable. Dette er to forskjellige metoder å bruke denne logikken.

Collections:

List: List interfacet gjør det mulig å lagre verdier i lister. Vi har arraylist som er en del av det. Der er det mulig å lagre data internt som arrayer. Vi har også Linked List som lagrer samme type data som en arraylist, men den har en oversikt over hva som er forrige og neste element i listen.

HashMap: Et hashmap implementerer map interfacet. I et hashmap så er to elementer koblet sammen, hvor den ene er en Key som må være unik og den andre er en Value som ikke må være unik. Når vi skal hente ut en verdi i et hashmap så henter vi det ut på Key’en ikke på posisjonen slik vi gjør i en arrayList. Hashmap kan være nyttig når du skal lagre data om personer eller lignende hvor alle er unike.

Queue: Queue fungerer som navnet, altså som en kø. Det er en liste for lagring av elementer hvor det første elementet som blir lagt inn er det første som blir tatt ut. Så hvis du skal legge til ett element blir det lagt til bakerst i køen, men det blir også da det siste elementet som kan bli tatt ut. Dette baserer seg på FIFO, First In First Out.

Stack: Stack er en bunke med elementer. Du kan se på det som en bunke med kort, hvor du legger de oppå hverandre. Det øverste elementet blir da det siste du la til og det første du kan ta ut igjen. Det baserer seg på LIFO, Last In First Out.

**Oppgave 2**

Jeg har valgt å fokusere på å sammenligne kontrollerne og repositoriene, siden det er disse som har blitt laget og endret på i oblig 4.

I PlanetControlleren så har jeg valgt å lage nye comparatorer i CelestialBody for å få sammenlignet og sortert planetene til et planetsystem. Her ser jeg at i forslaget, så har de valgt å bruke lambda og gjort det internt der, noe som kanskje hadde vært en bedre måte å løse oppgaven på. Siden det da er mer oversiktlig og mindre kode.

PlanetSystemControlleren har vi gjort mer eller mindre likt, ingen betydelige forskjeller der.

I UniverseRepository så har vi gjort det litt forskjellig. I forslaget så har det delt opp de to forskjellige solsystemene i to forskjellige metoder og kaller bare de i konstruktøren til repositoryet. Vi har gjort getPlanet metoden forskjellig. Jeg har valgt å bruke en for loop for å loope gjennom planetene i et planetsystem, for å finne den planeten vi er ute etter. Mens i løsningsforslaget har de valgt å kjøre getPlanetSystem metoden med .getplanet og planet navnet vi er ute etter. Dette sparer deg for kode og er en enklere måte å kjøre på enn det jeg valgteå gjøre.

Vi har gjort mye likt, men mye ulikt også. Det er flere steder hvor jeg kunne forenklet min kode til slik de har gjort i løsningsforslaget. Så det er noe jeg tar med meg til neste gang.