Teori

Oppgave 1 - Ord og begreper

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

- Exception

  Exception er en hendelse som skjer når kjører programmet, som får programmet til og slutte og lese. Da vil det komme opp en type error. Eller finere sakt ''Exception er en hendelse som ''forstyrrer '' den normale flyten i et programs instruksjoner mens programmet kjører''

  Kode eksempler:

```
public class Main {

    public static void main(String[] args) {

        String[] pets = {"dog", "cat", "monkey"};
        System.out.println(pets[3]);

    }
}
```

```
Main
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3.2\lib\idea_
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException Create breakpoint : Index 3 out of bounds for length 3
    at com.company.Main.main(Main.java:8)
```

  Det vil komme en feil meding fordi indexen starter på 0 og ikke 1.

  Første ordet som kommer opp i feilmedlingen er Exception det er fordi den prøver og kjøre koden innen for main metoden, og merket at (pets[3]) ikke eksisterer. Fordi i lista går det fra 0 og opp over. Og det er ikke et fjerde elemten i listen. På grunn av denne hendelsen så stoppet gjennomføringen av programmet det for er det exception.

- Tråd (Thread)

Tråd er en måte for Pcer å gjøre flere ting samtidig. Moderne CPUer har flere kjerner som utfører opperasjoner. Om man har en opperasjon som er uavhening av å ligge etter en annen kode, som et grensesnitt, er det mulig å legge den på sin egen tråd. Kode som kjører på forskjellig tråder har tilgang til samme minne(samme variabler, klasser og objekter). Noe som betyr at du kan ha flere koder som leser av det samme objektet.

Det som kan være vansklelig ved bruk av tråder, er å lage kode som er godt optimaliserer for dette. Siden programmet vanligvis kjører på en tråd, må man definere hva som skal gjøre på andre tråder.

Mulig å definere at bare en tråd kan endre på en kode. Da vil alle andre tråder stoppe opp, mens en annen tråd blir ferdig.

- Collections Framework
  - List

    List er som en vanlig liste som er nummerert fra 0 til noe. 0 er det første elementet i lista.

    0 – verdi

    1 – verdi

    Når man legger til en ny verdi i lista kommer det et nytt tall. Man kan da aksessere lister med og gå på navnet på lista. Så et tall så returnerer det verdien i lista.

        listeNavn[0] -> verdi

    Bare List har en satt størelse. Det vi si at man definerer at List skla være 20 lang, vil man ikke kunne legge til et 21. objekt. Fordelen med List er at det er raskere enn ArrayList.

    arrayList er en liste som ekspanderer data. arraylist er av typen List de arver fra list klassen. Men med ArrayList trenger ikke og ha en satt størelse. Det vil si at man kan definere en størelse på strarten, om man har definert en størelse på 20, kan man legge til et 21. objekt.

  - HashMap

    HashMap sin måte å lagre data på er litt forskjellig fra List. Her lagrer man alle dataene med en nøkkel og selve dataen. Fordelen med HashMap er når man skal lagre noe data kan man ha en enkel String som nøkkel. Om man har en Streng som nøkkel vil man kunne «søke» etter den strengen istedenfoe å måtte gå igjen alle elementene. Er mulig og lagre data med int som nøkkel, men da vil det oppstå litt det samme problemet som i List.

    <key, Value>

     <String, Planet>

    Verdien av et HashMap kan være hva som helst, Det viktigste er at nøkkelen er forskjellig fra alle elementer. Hvis du prøver å lagre et objekt med samme nøkkel som et annet, vil det bare overskrive det som stod i den orginalen nøkkel verdien.

    Eksempel på og hente ut verdi:

    listeNavn.get(«planet navn») -> planet

- Queue

  Queue lagrer data på «rader». Queue eaver fra LinkedList, noe som betyr at elementet ser/vet bare det som kommer før og etter. Dette gjør det ennkelt å finne de første og siste elementene, men det er vanskligere å jobbe med de elementene midt i, Queue er bare at man skal legge inn elementer som man skal ha ut igjen i samme rekkefølge, eller at man trnger å vite nabone/den som er etter til vært element.

  i0 -> i1 -> i2 -> i3

  i0 <- i1 <- i2 <- i3

- Stack

  Stack er litt som Queue. Forskjellen er at man bare har tilgang til toppen av listen. Det er mulig å søke etter et element og få indexen til det elementet, men for å kunne slette/fjerne elemente, blir elementene ovenfor også fjernet. Dette kan også bli brukt til å holde styr på en kø med elementer, bare at denne er raskere enn Queue.

Kodesammenligning

Sammenlignet med Even Langebråten sin kode. (Oppgave 2.4)

- Hente alle planetsystemer

  Han sjekker om planetsystemet hans er tomt og vis ikke retunerer han planetsystemene. Og vis det ikke er tomt så retunerer den en tom liste. Jeg henter ut hele listen.

- Hente ett spesifikt planetsystem

  De er like

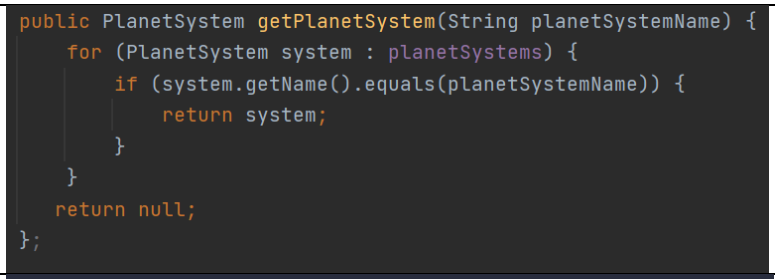
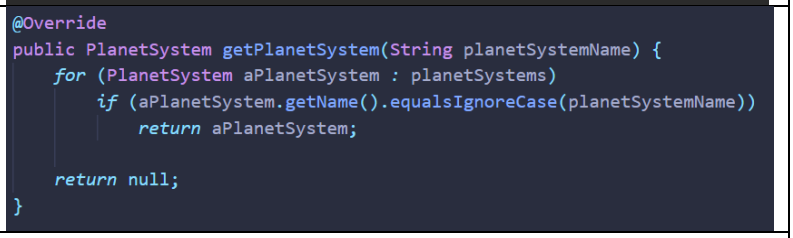
  Sammenligning med løsningsforslaget

  Oppgave 2.4

  - Hente alle planetsystemer

Min kode:	`public ArrayList<PlanetSystem> getAllPlanetSystem() { return planetSystems; };`
Løsningsforslag:	`@Override public ArrayList<PlanetSystem> getAllPlanetSystems() { return new ArrayList<>(planetSystems); }`
Sammenligning	Jeg har ikke override i denne men burde, i dette tilfelle har det ikke noe funkjonell betyding. I løsningsforslaget returnerer den et nytt objekt og kopier det samme inn i det nye objektet. Imens jeg retunerer objektet.

- Hente ett spesifikt planetsystem

Min kode:	
Løsningsforslag:	
Sammenligning	Her er det ganske likt uten om at jeg brukte .equals isteden for .equalsIgnoreCase det vi si at store og småe bokstaver har noe og si men det har det på min.