Almuthanna Jamal Aljajan

Teori

Oppgave 1.1

Polymorfisme (flere former) er et konsept i OOP og det vil si at objektene av sub-klasse typen kan behandles som objektene av super-klasse typen. Når det gjelder å opprette et objekt av en klasse, må vi først definere en referansevariabel objektet typen for å peke på adressen til objektet i minnet, men på grunn av polymorfisme kan vi deklarere en referansevariabel av superklasse som vil referere til objektet av sub-klasse-typen, og dette er veldig nyttig. Et eksempel på bruk av polymorfisme er hvis vi har klassene Planet og Star som arver fra klassen CeltialBody. og vi ønsker å definere en ArrayList som kan holde objektene av typen Planet og Star sammen. så på grunn av Polymorfisme kan vi oppnå dette ved å definere Arraylisten med typen av CeltialBody, så i dette tilfellet kan vi legge til ArrayListen elementene av typen Planet og Star (sub-klasse typer).

```
// CeletialBody klasse
public class CeletialBody {
}
//Planet klasse
public class Planet extends CeletialBody{
}
// Star Klasse
public class Star extends CeletialBody{
}
// i Main Klasse
ArrayList<CeletialBody> celetialBodies= new ArrayList<>();
celetialBodies.add(new Planet(););
celetialBodies.add(new Star());
```

NB: Eksempel er fra mine obliger.

Forelesning06_Uke04_Arv_Polymorfi_Array_ArrayList

https://www.tutorialspoint.com/java/java_polymorphism.htm

https://www.geeksforgeeks.org/polymorphism-in-java/

https://www.javatpoint.com/runtime-polymorphism-in-java

Oppgave 1.2

Primitive datatyper som finnes i java er boolean, char, int, short, byte, long, float og dobbel.

```
byte myByte = 100; // brukes for å holde heltall fra -128 til 127
short myShort = 30000; // brukes for å holde heltall fra - 32768 til 32767
int myInt = 42; // brukes for å holde heltall fra -2^31 til 2^31-1
long myLong =23459786549654L; // brukes for å lagre heltall fra -2^63 til 2^63-1
float myFloat = 23.4444F; // brukes for å holde et flyttall med 7 desimaler
double myDouble = 234876557932.50; // brukes for å holde et flyttall med 10
desimaler
```

```java
boolean myBoolean= true; // brukes for å holde to verdier true eller false
String fornavn = "Almuthanna"; //String er en klasse i Java
char myChar ='A'; // brukes for å holde en enkelt karakter
```

Forelesning02_Uke02_Datatyper_Metoder_Klasser

https://www.javatpoint.com/java-data-types

https://www.geeksforgeeks.org/data-types-in-java/

Oppgave 1.3

Klasse

Defineres som en "blueprint" hvor vi lager objekter fra. For å definere en klass, må vi først spesifiserer access modifiers generelt vi definerer det som public. Deretter bruker vi class nøkkelord til å lage en klasse og spesifiserer navn på klassen som skrives med stor forbokstav. Til slutt, hvis den klassen arver fra en annen klasse så bruker extends nøkkelord og navn på klassen som arver fra og hvis den klassen implementerer et interface, så bruker vi implements nøkkelord og navn på interfacet. Typisk Kropen til en klass er bygd opp av: instanse og eller statiske variabler , konstruktør, get- og set- metoder ,instanse og eller statiske Metoder og å override toString() metode.

Objekt

Defineres som en instanse av klassen. For å opprette et objekt fra en klasse , må vi først deklarere en referense variabel av objekttypen som skal peke på addressen til objektet i minne. Deretter må vi bruke 'new' nøkkelordet som brukes til å lage objektet. Og det new nøkkelordet blir fulgt av å kalle en konstruktørfor å initialiserer det nye objektet

```java
//Eksempel på hvordan vi definerer en klasse:
public class Planet {
  private String name;
  public planet(String name){
      this.name=name;
  }
}
//Eksempel på hvordan vi oppretter et objekt:
 Planet planet = new Planet("Earth");
```

Forelesning01_Uke02_Introduksjon

Forelesning03_Uke03_Klasser_Objekter_Metoder

Forelesning04_Uke03_Klasser_Objekter

Forelesning09_Uke06_Abstract_Interface_Parprogrammering

https://www.geeksforgeeks.org/classes-objects-java/

https://www.w3schools.com/java/java_classes.asp

https://www.tutorialspoint.com/java/java_object_classes.htm

Oppgave 1.4

Imperativ programmering

Imperativ programmering er en stil å skrive og organisere koden i applikasjoner basert på å fortelle programmene hvordan vi vil gjøre noe steg for steg for å oppnå det ønskete målet

Funksjonell programmering

er et programmeringsparadigme som baserer seg på å uttrykke alt i et program i form av funksjoner. Det finnes flere programmeringsspråk som støtter det som JavaScript, Lisp..osv. De hoved konseptene i funksjonell programmering er:

Pure functions er funksjoner som retunerer data basert på parametrene som funksjonen tar.

Higher-order functions er funksjoner som tar andre funksjoner som parametre eller retunerer dem

Recursive Functions er funksjoner som kaller seg selv.

Deklarativ programmering

er et programmeringsparadigme som bygger på å beskrive hva man ønsker å gjøre / oppnå med programmet, men ikke hvordan man gjør det, så det er i motsetning til det imperative paradigmet

https://codeburst.io/declarative-vs-imperative-programming-a8a7c93d9ad2

https://www.computerhope.com/jargon/i/imp-programming.htm

https://www.geeksforgeeks.org/introduction-of-programming-paradigms/

https://en.wikipedia.org/wiki/Declarative_programming

https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0

https://wiki.haskell.org/Functional_programming

https://www.geeksforgeeks.org/functional-programming-paradigm/

https://www.tutorialspoint.com/functional_programming/functional_programming_introduction.htm

https://en.wikipedia.org/wiki/Functional_programming

Oppgave 1.5

Class

Defineres som en “blueprint” hvor vi operetter objekter fra.

Object (konseptet, ikke klassen)

Defineres som en instanse som operettes av klassen.

Instance variable

Instans variabler er variablene som tilhører til et spesifikt objekt av klassen i motsetning til de statiske variablene som tilhører til klassen. Initialisering av instans variabel er ikke obligatorisk, så hvis vi ikke initialiser dem, får de standard verdier for eksempel for tall er 0, og for objekt som String er den null. Det anbefales å definere disse variablene som private for å oppnå Innkapsling. Disse variablene opprettes når et objekt av klassen er opprettet og til å få tilgang til dem, må vi opprette et objekt av klassen først.

https://www.geeksforgeeks.org/instance-variable-final-java/

https://www.tutorialspoint.com/Instance-variables-in-Java

https://www.javatpoint.com/java-variables

Overloading

Overloading av metoder/konstruktører er et prinsipp i OOP som vil si at klassen kan ha to eller mange metoder med samme navn eller konstruktør. Men på betingelse av at disse metodene/konstruktørene må ha ulike parameterne. Parameterne kan være forskjellige, Hvis de er forskjellige i antall eller i datatyper.

https://www.javatpoint.com/method-overloading-in-java

https://www.geeksforgeeks.org/overloading-in-java/

https://beginnersbook.com/2013/05/method-overloading/

Overriding

Overriding metoder er også et prinsipp i OOP som vil si at det er mulig for sub-klassene å implementere på nytt og på en spesifikk måte en eller flere av metodene som allerede eksisterer i super-klassen. Når vi vil å override en metode, bør vi å benytte @Override annotering, før å override det.

Forelesning06_Uke04_Arv_Polymorfi_Array_ArrayList

https://www.geeksforgeeks.org/overriding-in-java/

https://www.javatpoint.com/method-overriding-in-java

Extends

extends er et nøkkelord som benyttes, når vi vil at en klasse (sub-klasse) skal arve fra en annen klasse (super-klasse), så dette nøkkelordet benyttes for å refere til arven mellom klassene.

Forelesning05_Uke04_IntelliJ, Arv

https://www.geeksforgeeks.org/inheritance-in-java/

https://www.tutorialspoint.com/java/java_inheritance.htm

private,public,(protected) (klasse,variabel,metode)

public, private og protected er tilgangsmodifikatorer som benyttes med variabler, metoder eller klasser for å begrense tilgangen til dem.

public

Når vi definerer variebler, metoder eller klasser som public, vil det si at de er tilgjengelige fra hvor som helst (alle steder) i programmet.

private

Når vi definerer variebler, metoder som private noe som vil si at de er tilgjengelige bare inni klassen som de er definert i. når det kommer for klasser i hoved sak, defineres det ikke som private, men det er mulig å definere nested klasser slik og i dette tilfellet vil det si også at tilgjengelige bare inni klassen som de er definert i.

protected

Når vi definerer variabler, metoder som protocted noe som vil si at de er tilgjengelige i samme pakke eller fra forskjellige pakker hvis sub-klasser befinner seg i forskjellige pakker.

Forelesning07_Uke05_ArrayList_UML

https://www.javatpoint.com/access-modifiers

https://www.geeksforgeeks.org/access-modifiers-java/

this og super

this

this er et nøkkelord som benyttes for å indikere til det aktuelle objektet. I hoved sak det brukes i konstruktørene og i set-metoder for å skille mellom instans variabler og parametere som har samme navn. I tillegg det kan også benytte til å kalle konstruktøren inni en annen konstruktøren i samme klasse.

Forelesning04_Uke03_Klasser_Objekter_Arv

https://www.w3schools.com/java/ref_keyword_this.asp

https://www.geeksforgeeks.org/this-reference-in-java/

https://www.javatpoint.com/this-keyword

super

super er et nøkkelord som benyttes til å indikere til klassen (super-klassen) som sub-klassene arver fra. vi kan benytte det til å få tilgang til instans variablene eller å kalle konstruktørene og metodene som finnes i super-klassen.

https://www.geeksforgeeks.org/referencing-subclass-objects-subclass-vs-superclass-reference/

https://www.javatpoint.com/super-keyword

Refaktorere

er et konsept som basert på å omstrukturere koden som allerede eksisterer i programmet, men uten å forandre hovedfunksjonen som utføres av denne koden. Dette kan oppnås, ved å fjerne den dupliserte koden, gjenbruk av koden og unngå lange metoder ved å dele den opp i flere deler. Dette vil gjør at debugging av koden blir enklere og at vi har bedre kontroll over det og gjøre for oss at det er mulig å bruke disse delene i andre metoder og dermed oppnå også gjenbruksprinsippet.

https://www.geeksforgeeks.org/ugc-net-ugc-net-cs-2018-july-ii-question-20/

https://dzone.com/articles/what-is-refactoring

https://en.wikipedia.org/wiki/Code_refactoring

Static (variabel, metode)

Static er et nøkkelord som brukes begge med variabler og metoder. Når variabler og metoder defineres med static, noe vil si at de hører til klassen, men ikke til et spesifikt objekt av klassen. I tillegg når vi definerer når variabler og metoder med static, så vi kan få tilgang til/kalle dem ved bruk av bare navnet til klassen uten at vi trenger til å operette en instans av klassen f.eks. Math.pow(), Math.PI…osv. Static variabler kan benytte for

egenskapene som er felles blant alle objektene. static metode kan ikke bruke this nøkkelordet, fordi det finnes ikke objekt som kan "this" referere til.

Forelesning08_Uke05_Static_Final

https://www.javatpoint.com/static-keyword-in-java

https://www.geeksforgeeks.org/static-keyword-java/

Final (variabel, metode, klasse)

final er et nøkkelord som kan benyttes med variabler, metoder og klasser. Når vi definerer en variabel med final, noe vil si at når denne variabelen får en verdi, denne verdien kan ikke forandres etterhvert. Variabler kan også defineres med static sammen med final og det kalles i dette tilfellet konstanter. Når en metode er definert med det final nøkkelordet, så metoden kan ikke overskrives og når en klasse er deklarert med det final nøkkelordet, så klassen kan ikke bli arves (extends).

Forelesning08_Uke05_Static_Final

Forelesning11_Uke07_Abstract_Final_Exception

https://www.geeksforgeeks.org/final-keyword-java/

https://www.javatpoint.com/final-keyword

Abstract (klasse, metode)

abstract er et nøkkelord som brukes for å definere abstrakte klasser og metoder. Når vi definerer klasser med abstract, noe vil si at det er ikke mulig å operette objekter av dem. Abstrakte klasser kan inneholde i tillegg til instans variabler, metoder og konstruktør, kan inneholde abstrakte metoder. I hovedsak benyttes de for å abstrahere dataene som er felles i sub-klassene.

Abstrakte metoder er metoder som deklarert med abstract nøkkelordet og med bare metode-definisjon (hode) og uten noe implementasjon (kropp). I tillegg de kan defineres bare inni abstrake-klasser. Når en absract klasse inneholder abstract metoder, så blir sub-klassen tvunget til å lage sine egne implementasjoner for disse metodene

Forelesning09_Uke06_Abstract_Interface_Parprogrammering

Forelesning10_Uke07_Interface_Anon

Forelesning11_Uke07_Abstract_Final_Exception

https://www.geeksforgeeks.org/abstract-classes-in-java/

https://www.javatpoint.com/abstract-class-in-java

Interface

Interfacet defineres som en kontrakt mellom interfacet og de klassene som skal implementere det. Interfacet deklareres ved å benytte interface nøkkelordet og det kan inneholde begge konstante variabler og abstrakte metoder. Alle metodene i interfacet er definert som standard med public og abstract. Når klassene vil implementere et interface, må de benytte implements nøkkelordet isteden av extends nøkkelordet. I dette tilfellet tvinger interfacet de klassene til å implementere alle de abstrakte metodene som er definert der. Når en abstrakt klasse implementerer et Interface, men lager ikke implementasjon av metodene som finnes der, så tvinger det de sub-Klassene til å gjøre det isteden av.

Forelesning09_Uke06_Abstract_Interface_Parprogrammering

Forelesning10_Uke07_Interface_Anon

Forelesning11_Uke07_Abstract_Final_Exception

https://www.geeksforgeeks.org/interfaces-in-java/

https://www.tutorialspoint.com/java/java_interfaces.htm

Anonymous Inner Class

Anonym indre klasser defineres som en type av nested klasser som ikke har navn. I hoved sak brukes det for å implementere et interface isteden av å operette en ny klasse for å oppnå dette, så det benyttes også for å forkorte koden. Når vi operetter en anonym indre klasse, vi egentlig definerer klassen og objektet av klassen samtidig. Gjennom kurset vi har brukte de for å implementere Comparator interface for å lage en spesifikk sortering og Handler interface for å håndtere requesten som kommer fra frontend isteden å lage en ny klasse for det.

```
Collections.sort(planets, new Comparator<Planet>() {
    @Override
    public int compare(Planet p1, Planet p2) {
        return p1.getName().compareTo(p2.getName());
    }
});
```

Forelesning10_Uke07_Interface_Anon

https://www.geeksforgeeks.org/anonymous-inner-class-java/

https://www.javatpoint.com/anonymous-inner-class

MVC (konseptet, og hver enkelt del)

MVC står for Model, View, og Controller og det er en teknikk som basert på å strukturere/organisere koden til programmet i forskjellige lag/pakker

Model

Model inneholder de klassene som vi bruker det for å generere dataene /objektene i programmet f.eks. i denne obligen klassene som er model-klasser er: Animal, Biom, Bird, Location….osv.

View

View er brukeren grensesnittet i programmet og det inneholder HTML, CSS og Vue filer som tar seg om hvordan grensesnittet vil ser ut. view i hovedsak har ansvar for å vise dataene som får fra backend og sende requestene basert på hva brukeren har gjørt til backend for å håndtere det.

Controller

Hovedfunksjonen til Controlleren er å oppnå en kobling mellom View og Model ved å ta de ønskete dataene fra backend og gi de videre til frontend eller ved å oppdatere datene i backend og sende de nye datene tilbake til frontend for å vise det. Controlleren på en måte som har ansvar for hvordan skal håndtere de requestene som kommer fra frontend f.eks. ved å operette en liste eller ved å linke til en ny webside.

Forelesning14_Uke09_Javalin_MVC_Lambda

https://www.tutorialsteacher.com/mvc/mvc-architecture

https://www.geeksforgeeks.org/mvc-design-pattern/

Exception

Exception er et objekt som blir kastet når en feil eller en uventet hendelse skjer i kjøretiden av programmet som ofte fører til at programmet vil stoppe/krasje. I hovedsak finnes det tri typer av exceptions:

Checked exceptions

er feiler som vi forventer at de skal skje, og vi må på en måte behandle det. For eksempel finnes det noen predefinerte metoder i Java som kaster exceptions. Disse feiler må vi håndtere ved å skrive koden som vi forventer at den skal kaste en feil inni try-blokket og i catch-blokket skriver vi koden som blir kallet når koden i try-blokket kaster en feil for håndtere det og å unngå at programmet krasjer

Unchecked exceptions

er feiler som vi ikke forventer at de skal skje i heltatt i kjøretiden av programmet og det er i hovedsak feil i koden/logikken for eksempel når vi forsøker å aksessere et element som ikke finnes i en array.

Errors

er seriøse feiler som vi kan ikke håndtere vi å bruke try-catch blokker for eksempel når systemet stopper eller når det er ikke nok plass på minnet

Forelesning11_Uke07_Abstract_Final_Exception

Forelesning16_Uke10_Filskriving

https://www.geeksforgeeks.org/exceptions-in-java/

https://www.tutorialspoint.com/java/java_exceptions.htm

https://www.javatpoint.com/exception-handling-in-java

Threads

Hovedfunksjonen med tråd er å utføre noen deler av koden i programmet samtidig på forskjellige tråder. Det finnes to måter for å oppnå dette. Den ene er ved å la klassen/anonym klassen implementerer Runnable interfacet og deretter implementerer vi også run () metoden ved å skrive koden som vi ønsker at det skal utføres parallelt inni metoden og starte tråden ved å kalle start() metoden i Main/Application klasse. den andre måte er ved å la klassen arver fra Thread klassen som implementerer også Runnable interfacet og dermed må vi også implementere run () metoden og deretter starte tråden ved å kalle start() metoden i Main/Application klasse.

Forelesning20_Uke13_Traader

https://www.javatpoint.com/creating-thread

https://www.tutorialspoint.com/java/java_multithreading.htm

https://www.geeksforgeeks.org/java-lang-thread-class-java/

Collections Framework

- **List**

  List er et interface som arver fra Collection interfacet og dette interfacet er implementeres av klassene som ArrayList og LinkedList og Vector og Stack. Dette interfacet har flere abstrakte metoder som add(), size(), remove() og clear()...osv, hvor de klassene som implementer List interfacet implementerer disse metodene.

Forelesning17_Uke11_JSON_Collections

https://www.geeksforgeeks.org/list-interface-java-examples/

https://www.javatpoint.com/java-list

- **Map**

  Hashmap er en klasse som er en del av Collection framework. Det holder dataene i form av (nøkkel, verdi), så når vi ønsker å definere en HashMap, må vi spesifisere typen til begge nøkkelen og verdien. For å få tilgang til verdiene må vi bruke nøkkelen for å referere til verdien ved å benytte get(nøkkel) metode og for å legge elementer til en Hashmap bruker vi metoden som heter put(nøkkel, verdi).

  Forelesning17_Uke11_JSON_Collections

  https://www.w3schools.com/java/java_hashmap.asp

  https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/

- **Queue**

  Queue er et interface som arver fra Collection interfacet og dette interfacet er implementeres av klassene som LinkedList og PriorityQueue. Det er basert på (First In First Out) konseptet, og det vil si at første element vil legger til Queue, er det første element som vil hentes fra Queue.

  Forelesning17_Uke11_JSON_Collections

  https://www.geeksforgeeks.org/queue-interface-java/

  https://www.javatpoint.com/java-priorityqueue

- **Stack**

  Stack er en klasse som er en del av Collection-framework. Det er basert på (Last In First Out) konseptet, og det vil si at siste element vil legger til Stack, er det første element som vil hentes fra Stack.

  Forelesning17_Uke11_JSON_Collections

  https://www.geeksforgeeks.org/stack-class-in-java/

  https://www.javatpoint.com/data-structure-stack