Oppgave 1.1

--Refraktorere--

Refraktorering vil si at du vil ha så lite dupliserbar kode som mulig.

Eksempel på dette kan være at du extender fra en klasse som er abstrakt.

Den abstrakte klassen vil da inneholde variabler og metoder som også skal være et annet sted, men istede for å skrive det flere ganger så arver underklassen fra den.

Eks: class abstract Person{

private String etternavn;

private String name;

public Person(String name, String etternavn){

this.name = name;

this.etternavn = etternavn;

}

get/set metoder etc

}

class tyv extends Person{

public tyv(String name, String etternavn){

super(name,etternavn);

}

}

Istede for at vi da skriver inn variablene name og etternavn i tyv, så arver vi de istede. Dette gjør at når vi bruker konstruktøren til tyv, så får vi at vi skal skrive inn name og etternavn.

--Static(variabel,metode)--

Bruk av variabler eller metoder i en klasse uten at vi lager et objekt.

Variabler: private static int tall = 0; Alle objektene du lager etter vil ha den samme verdien som variabelen er. De får ikke noen egen verdi, selvom de har en egen instanse.

Metode: En statisk metode, er en metode som vil oppdatere seg eller være den samme uavhengig av objektene som er laget for klassen.

Eks:

class calc{

static int count = 0;

calc(){ //konstruktør

count++; //Legger til 1 i count

}

}

calc s = new calc(); // Gir 1

calc f = new calc(); // Gir 2

Uten static ville count begynt på 0 for begge og lagt til 1 i initialiseringen slik at begge hadde vært 1.

--Final (variabel, metode, klasse)--

Variabel: Kan ikke endres etter den er initialisert.

metode: Kan ikke @override fra den

Klasse: Kan ikke arve fra den

--Abstract (klasse, metode)--

Klasse: En klasse som er abstrakt går ikke an å lage et objekt av.

Eks: class abstract Person{}

Metode: En metode som er abstrakt har ingen return verdi,

og klassen som arver fra en abstrakt klasse må bruke denne metoden.

De må override metoden.

En vanlig klasse kan bare extende fra en abstrakt klasse.

Klassen som lager en abstrakt metode må være abstrakt.

Eks: public abstract int getAlder(LocalDate fodselsdato);

--Interface--

Har ingen implementasjon selv, så klasser kan implementere et interface.

Et interface kan kun inneholde abstrakte metoder.

Eks:

public interface TegnBar {

void tegn();

}

En klasse kan implementere mange interface.

Variablene vil alltid være public og final.

Oppgave 1.2

Jeg jobbet med Sigrid Gjøen

---------getSurfaceGravity()-------------------

public double getSurfaceGravity() {

final double g = 0.00000000006674;

double r = radius \* 1000;

return (g \* mass) / Math.pow(r, 2);

}

Helt den samme

---------getSmallestPlanet()--------------------

Planet getMin() {

Collections.sort(planets);

return (Planet) planets.get(0);

}

@Override

public int compareTo(Object obj) {

Planet planet = (Planet) obj;

if (radius > planet.radius) {return 1;}

if (radius < planet.radius) {return -1;}

if (mass == planet.mass) {return 0;}

if (mass > planet.mass) {return 1;}

return -1;

// 1. Returner tallet 1 om denne planeten er større enn den som kommer inn som argument

// 2 Returner tallet -1 om denne planeten er større enn den som kommer inn som argument

// 3. Retuner tallet 0 om planete er like store. Done.

}

Ho brukte implement Comparable i Planet klassen. Der måtte ho @Override compareTo(object o). I denne overriden, så sjekket hun planetene og sorterte de.

Lagde så en collection.sort i PlanetSystem og returnerte det første om det var minst og det siste om det var størst.

---------getMassinMSun()-------------------

public double getMsun() {

final double oneMsun = 1.98892E30;

return oneMsun / super.mass;

}

Så og si gjort det samme, eneste forskjell var at jeg skrev verdien i variablen Msun ved bruk av Math.pow og at jeg ikke skrev super.mass.

Oppgave 1.3

*CelestialBody*

