Oppgave 2.1 - Arv

Hvis vi ser nærmere på **Planet** og **Star** så ser vi at disse har flere instansvariabler som er identiske. Begge disse klassene kan sees på som himmellegemer eller "Celestial Body". En **Planet** "*is a"* **Celestial Body**, og en **Star** "*is a*" **Celestial Body**.

Vi ønsker derfor å abstrahere de instansvariablene som er felles til en ny klasse **Celestial Body**.

1. Abstraher (flytt) instansvariablene som er felles fra **Star** og **Planet** til en ny klasse **Celestial Body**
2. Flytt metoder du mener hører med i **Celestial Body**
3. Lag kontruktør i **Celestial Body**
4. Sett **Planet** og **Star** til *extends* **CelestialBody**
5. Pass på å kalle *super*-konstruktøren i **Planet** og **Star**
6. Verifiser at all kode vi har i *Main.java* fortsatt kjører som den skal

Oppgave 2.2 - Navn

Det er ønskelig å kunne hente ut en planet fra et PlanetSystem basert på navn. Lag en metode som gjør dette.

Oppgave 2.3 - Konstanter

I oblig 2 lagde vi noen metoder for å kunne hente ut verdier i astronomiske enheter fremfor i "vanlige" enheter som kg og km. Denne konverteringen tok for seg noen satte verdier for disse. F.eks. 1 Rjup = 71492km.

Lag disse verdiene som konstanter (static final) i sine respektive klasser, og benytt disse konstantene i metodene du lagde i.e. (getMassInMjup()).

Oppgave 2.4 - Naturlig satellitt og arv

I astronomi har vi et konsept som går på at objekter går i bane rundt andre objekter. Disse kalles satelitter. En *naturlig* sattelitt er f.eks. en måne eller planet som går i bane rundt en planet eller stjerne. F.eks. er jorden en satelitt fordi den går i bane rundt solen. På samme måte er månen en satelitt fordi den går i bane rundt jorda.

Vi ønsker å kunne gjøre enkle beregninger av disse banene, og vil derfor å introdusere konseptet med satelitter. Vi skal derfor lage klassen **NaturalSatellite**.

a) - Naturlig satellitt data

De aller fleste av disse banene går i en form for ellipse. For å kunne beregne banen og hastigheten til en naturlig satelitt, trenger vi noen datapunkter. Disse er:

- *semi-major axis* - denne gir den lengste avstanden fra brennpunktet i en ellipse til yttersiden, altså fra planeten til sjernen den sirkler rundt (dette er ikke heeelt korrekt, men vi kan forholde oss til det slik)
- *eccentricity* - Et tall mellom 0 og 1 som sier noe om hvordan ellipseformen er
- *orbital period* - Hvor lang tid det tar for å sirkle en runde (i jordens dager)
- *centralCelestialBody* - CelestialBody'en denne naturlige satelitten går i bane rundt

Lag klassen **NaturalSatellite** med de nevnte datapunktene, samt get- og set-metoder.

b) - Naturlig satelitt arv

Hvis vi sier at en **NaturalSatellite** "is a" **CelestialBody**, og **Planet** "is a" **NaturalSatellite**. Hvordan blir det nå naturlig å sette opp arvehierarkiet?

Gjør de naturlige endringene for å få dette nye arvehierarkiet til å fungere som tiltenkt. Det vil bli nødvendig å endre på konstruktører.

Gjør endringer i *Main.java* slik at planetene som lages i planetsystemet får de nye dataene vi har spesifisert.

Oppgave 2.5 - Avstand

Vi ønsker å kunne finne avstanden mellom en naturlig sattelitt og objektet det går i bane rundt. For å gjøre dette kan vi benytte Kepler's orbit formula.

$$r = \frac{a(1 - e^2)}{1 + e\cos\theta}$$

- r = avstanden i **AU** (AstronomicalUnits - 1AU = gjennomsnittlige avstanden mellom jorda og sola)
- a = semi-major axis
- e = eccentricity
- θ = vinkelen (true anomaly) (mellom den posisjonen vi vil ha avstanden for, og punktet hvor den naturlige satelitten er nærmest objektet det går i bane rundt, denne avstanden kalles forøvrig periapsis)

Hint: Man kan benytte Math klassen for å hjelpe til med noen av utregningene her. Math.cos() (denne tar verdi i radianer), og Math.toRadians() f.eks..

a) - Avstandsimplementasjon

1AU =149 597 871km. Hvordan vil det være fornuftig å definere denne i koden?

Lag en metode for å regne ut avstanden, ***distanceToCentralBody(double degrees)***. Denne bør returnere avstanden i ***km***.

b) - Avstandsutregning

I *Main.java* benytt denne metoden, og finn avstand mellom Jorda og Sola ved 0, 90, 180, 270 og 360 graders vinkel.

Hvis du har implementert formelen og konverteringene riktig vil du få ca:

Earth has a distance of 147054707km to the Sun at 0 degrees
Earth has a distance of 149554637km to the Sun at 90 degrees
Earth has a distance of 152141034km to the Sun at 180 degrees
Earth has a distance of 149554637km to the Sun at 270 degrees
Earth has a distance of 147054707km to the Sun at 360 degrees

Oppgave 2.6 - Hastighet

Vi ønsker også å finne hastigheten til en naturlig satelitt. Dette kan vi benytte den underliggende formelen til.

$$v = \sqrt{\frac{GM}{r}}$$

- v = hastigheten i **m/s**
- G = Gravitational constant
- M = massen til "central body", altså objektet den naturlige satelitten sirkler rundt
- r = avstanden i **meter**

a) - Hastighetsimplementasjon

Lag en metode for å regne ut hastigheten **orbitingVelocity(double distance)**. Denne skal returnere hastigheten i **km/s**.

Det blir nødvendigvis noe konvertering fra km til meter, og m/s til km/s for å tilpasse utregningene til formelen. Dette er bare å dele og gange på 1000 respektivt.

b) - Hastighetsutregning

I *Main.java* benytt denne metoden, og finn hastigheten til Jorda ved 0, 45, 90, 135 og 180 graders vinkel.

Hvis du har implementert formelen og konverteringene riktig vil du få ca:

At a distance of 147054707km, Earth has a velocity of 30.04km/s
At a distance of 147778223km, Earth has a velocity of 29.97km/s
At a distance of 149554637km, Earth has a velocity of 29.79km/s
At a distance of 151374279km, Earth has a velocity of 29.61km/s
At a distance of 152141035km, Earth has a velocity of 29.54km/s

Oppgave 2.7 - Another String

@Override metoden *toString()* i klasser som ikke har det allerede.