Trinn 9 – Interface, polymorphism, casting

I forrige uke lærte vi om abstraksjon. Jeg nevnte at vi kan utføre abstraksjon på to måter i Java: Abstrakte klasser og interface. Så denne gangen er det interface sin tur. Men når vi jobber med abstraksjoner blir det helt naturlig å lære om polymorphism og casting også!
Også denne gangen har jeg laget en lengre intro-video, så se gjerne den før du begynner.
Mål for dette trinnet:

- Jeg kan benytte interface, og jeg skjønner hva det innebærer.
- Jeg forstår hva polymorhism er.
- Jeg vet hvordan jeg kan benytte casting.

Relevante kapitler i pensumboka:
- Kapittel 11: Using interfaces
- Kapittel 15: Polymorphism explained
- Kapittel 16: Polymorphism in action

Figurene skal kunne flyttes. Det betyr at de også må ha informasjon om hvor de er. For en sirkel trenger vi bare informasjon om ett punkt (sentrum av figuren). For rektangler og kvadrater trenger vi å vite plasseringen på to punkter (for eksempel topp-venstre hjørne og bunn-høyre hjørne). Punkter har en plassering i et koordinatsystem med verdi for x og y. Figurer skal kunne flyttes opp, ned, høyre og venstre.

Oppgave 1

Lag et interface *Movable*. Metodene i interfacet skal være:
- *moveUp(double distance)*
- *moveDown(double distance)*
- *moveRight(double distance)*
- *moveLeft(double distance)*

Oppgave 2

Lag en klasse *MovablePoint* med to fields: *x* og *y* (begge av type double).
Klassen skal implementere *Movable*. *moveUp* skal øke *y* med angitt verdi, *moveDown* skal minke *y* med angitt verdi, og tilsvarende for begevelser i x-retning (moveRight/Left). Hvis du trenger litt hjelp, så har du hvit tekst på hvit bakgrunn nedenfor:

Husk å lage en toString-metode også

Oppgave 3

Vi ønsker at alle figurer våre skal ha en posisjon i et koordinatsystem. Vi vil altså vite hvor på x-aksen og y-aksen figurene befinner seg. En utfordring er at figurene er forskjellige. For sirkler trenger vi bare å vite et punkt (senteret i sirkelen). For rektangler og kvadrater trenger vi å vite plasseringen på to punkter (for eksempel topp-venstre hjørne og bunn-høyre hjørne).
La oss forsøke å benytte klassen *MovablePoint* fra oppgave 2.

Bruk *MovablePoint* når du sørger for at en sirkel har en plassering. Hvis du tenker «Hæ, hvordan i huleste kan jeg vite hvordan jeg skal gjøre det?!?» Da kan du få noen hvite tips på hvit bakgrunn nedenfor.

Oppgave 4

Sørg for at også figurene Rectangle og Square har en plassering. Husk at begge må ha to punkter: Et for topp-venstre hjørne av figuren, og ett for bunn-høyre.

Oppgave 5

Vi ønsker at **alle geometriske figurer skal være *Movable***. Har du noen idé om hvordan vi kan få til det? Tygg litt på den….
Har du tenkt ferdig? Helt sikker?

Jeg tenker det kan være lurt å la Shape implementere *Movable*. Alle geometriske figurer er jo en *Shape*.

Da må det vel være bedre å la *Shape* implementere *Movable* enn å la alle subklassene til *Shape* gjøre det?
Endre *Shape* slik at den implementerer interfacet *Movable*.
Hva skjedde?

Du vil se at vi ikke får noen kompileringsfeil i klassen *Shape*. Den er jo abstrakt. Men subklassene får problemer. Vi sier jo nå at alle *Shapes* må implementere metodene fra interfacet *Movable*. Men det er jo ikke tilfelle (enda).

Så hvordan skal vi fikse det? Tenk på den, du!

Det hadde jo vært digg å kunne implementere metodene i Shape. For da kunne det gjelde for alle subklasser. Men vi har et problem med at noen figurer har ett punkt for posisjon, og noen har to. La oss derfor forsøke oss på å implementere metodene i *Circle* først.
Endre *Circle* slik at metodene fra interfacet *Movable* blir implementert. Vanskelig? Da får du hjelp av hvit tekst på hvit bakgrunn nedenfor. Men forsøk å finne det ut på egen hånd først

Oppgave 6
Sørg for at både *Rectangle* og *Square* implementerer interfacet *Movable*.

Oppgave 7
Puh, nå er det vel ikke mer å gjøre? Jo, vi må jo teste om dette fungerer. Og da kan vi i samme slengen se på litt bruk av polymorfi! Bruk klassen *Test* til å lage metoder som tester ut koden. Opprett ett objekt av hver type (*Circle*, *Rectangle* og *Square*). Putt disse i en *ArrayList*.
Hmm, objektene er jo av forskjellige typer. Hvilken type skal vi benytte når vi oppretter *ArrayListen*? Tenk litt på den, og få hvit-tekst-hjelp nedenfor om nødvendig.

Gå igjennom alle elementene i ArrayListen og flytt elementene i en eller annen retning. Gå igjennom elementene på nytt og sjekk om ny posisjon samsvarer med forventingene.

Oppgave 8
Lag en unik metode for hver type (*Circle*, *Rectangle* og *Square*) – altså en metode som bare finnes i de enkelte klassene. Eksempel for *Circle*:

```
public void uniqueCircleMethod(){<SOUT noe>}.
```

Opprett ett objekt av hver type, men denne gangen skal du *upcaste* objektreferansene når objektene opprettes. Eksempel:

```
Shape s1 = new Circle(1, Color.PINK, true, new MovablePoint(0.0, 0.0));
```

Sjekk hvilke metoder du har tilgang til via objektreferansene. Har du tilgang til de unike metodene? Putt objektene i en ArrayList. Gå igjennom alle objektene i listen. For hvert objekt, skriv ut (SOUT) informasjon om areal og omkrets. Deretter skal du kalle på den unike metoden for objektet (hint: *instanceof*).

Ekstraoppgave 1
Ettersom vi for rektangler og kvadrater her definert to hjørner (*topLeft* og *bottomRight*), så kunne det være på sin plass å validere at objektene som opprettes av disse typene overholder visse regler for plassering. Det gir for eksempel ikke mening at et *topLeft* hjørne ligger plassert til høyre for *bottomRight*. Sørg derfor for at objekter opprettes kun hvis plassering av hjørnene gir mening.

Ekstraoppgave 2 (Ekstra vanskelig)
Klarer noen denne? Jeg har i hvert fall ikke trengt å forsøke –

Samme tankegang som forrige ekstraoppgave. I tillegg til å validere plassering skal du validere at lengdene samsvarer med plasseringen.