IN1010 Gruppetime Uke 4

Gruppe 2

Interface

- Et interface ligner litt på en abstrakt klasse
  - Vi kan ikke opprette *instanser* av et interface
  - Et interface har metodesignaturer, men ikke implementasjoner til metodene
- Hvis vi sier at en klasse *arver* fra en annen klasse, kan vi si at en klasse *implementerer* et interface
- Nøkkelord i Java: Implements
- Hvis en klasse implementerer et interface *må* man definere implementasjonen til interfacets metoder
- Kodeeksempel: Papir og Dokument

Hvorfor bruke grensesnitt?

- Java tillater ikke multippel arv, dvs. en klasse kan kun arve fra én annen
  - En klasse har ingen grense på hvor mange grensesnitt den kan implementere
- Nyttig hvis enkelte klasser på tvers av klassetrær har lignende egenskaper
- Lar oss samle objekter på samme måte som arv lar oss samle objekter
  - Pekere kan et interface som type
  - Kun objekter av klasser som implementerer interfacet kan refereres til av disse pekerne
- Bra for testing

Interface dere kjenner fra før

- *List* og *Map* er grensesnitt som brukes av objekter dere har brukt før
- ArrayList implementerer List, HashMap implementerer Map
- Det er mange ulike typer lister og maps, men disse objektene har ulike måter å lagre elementer på
- Vi vil at de skal ha de samme metodene, men det gir ikke mening at alle lister skal arve fra en liste-klasse og alle maps skal arve fra en map-klasse
- Titte i dokumentasjonen for List

Exceptions

- Exceptions er også objekter i Java
- Forskjell mellom checked og unchecked exceptions
  - Checked exceptions fanges ved kompilering
  - Unchecked exceptions oppstår under kjøring
- Vi kan lage våre egne exceptions
  - Kode eksempel: Unchecked AlleredeSignertException

a) Lag et interface, Drikkbar. Interfacet skal ha to metoder: drikkGlass(); og fyllOpp();

b) Lag deretter en klasse «Drikke» som implementerer grensesnittet Drikkbar. Drikke må holde på hvor mye drikke det er plass til i flasken (dl) og hvor mye drikke det er i flasken nå (dl). Maksstørrelsen på flasken settes i konstruktøren og vi kan anta at flasken er full når den opprettes.

c) Dersom metoden «drikkGlass» kalles skal flasken reduseres med 2 dl. Dersom det er mindre enn 2 dl igjen, skal flasken tømmes (til 0 dl). «fyllOpp» skal fylle flasken opp til maksstørrelse.

d) Utvid programmet ditt med subklassen Ol (øl). Subklassen skal være lik drikke, men i stedet for at flasken reduseres med 2 dl, skal den reduseres med en halv liter når et glass drikkes. Lag også en toString()-metode for både drikke og øl. *Hint: husk @Override der du mener det er riktig.*

e) Tegn klassehierarkiet for oppgaven.
f) Implementer en liten test av programmene dine.

a) Vi skal lage en enkel modell av en næringskjede. På toppen av næringskjeden finnes rovdyrene og på bunnen finnes planteeterne. Skriv to grensesnitt; ett for rovdyr og ett for planteetere. Rovdyrene skal kunne jakte, mens planteeterne skal kunne beskytte seg selv.

b) Skriv klassene Bjorn, Ulv, Elg og Sau som implementerer de riktige grensesnittene over. Metodene som må implementeres trenger kun å printe ut en beskjed om hva dyret gjør. Opprett objekter av klassene og legg alle rovdyrene i en array av rovdyr, mens du legger alle planteeterne i en array av planteetere. Test programmet ditt. *Diskusjon: hva gjør man med Bjørn; er den et rovdyr eller en planteeter, eller kan vi gjøre den til begge deler?*

c) Tegn opp klassehierarkiet med alle klasser og grensesnitt.

Gjør Trix-oppgave 05.05. om egne exceptions.