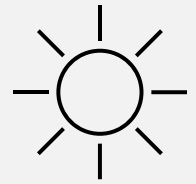
GRUPPE 11 IN1010 – UKE 6

Sivert Fjeldstad Madsen

I DAG

- Beholdere og generiske klasser
- Klasseparameter/typeparameter
- Indre klasser
- Exceptions
- Oppgaver



BEHOLDERE

- Når vi koder bruker vi beholdere hele tiden
- Java har mange innebygde beholdertyper
  - Arrays, ArrayList, HashMap, PriorityQueue, HashSet, osv…
- Av og til kan det likevel hende vi ønsker å designe beholdere selv
  - Kan hende vi trenger spesiell funksjonalitet
- Veldig mange måter å lage beholdere på!



GENERISKE KLASSER

- Når vi designer en beholder i Java ønsker vi gjerne at den skal kunne holde på objekter av en hvilken som helst klasse
  - Upraktisk om beholderen vår f.eks bare kan holde på Strenger
- Vi må da bruke det som kalles en *generisk klasse*
  - Altså en klasse som tar inn en annen klasse som «parameter» til *klassen*
- Dette gir oss mulighet til å lage beholdere som kan ta i mot hvilken type objekt som helst



KLASSEPARAMETER/TYPEPARAMETER



- Oppgis i signaturen til klassen:
- class Beholder<**T**> { }
- T-en står for type, og må erstattes med en bestemt type når vi oppretter instanser av klassen:
- Beholder**<String>** b = new Beholder<>();
- Finnes andre vanlige plassholdere i tillegg til T
  - E – Element, K – Key, V –Value, N - Number
- Har vi vært borte i klasseparametere før?

INDRE KLASSER

- Når vi designer egne beholdertyper trenger vi ofte en egen type objekt for å holde på datapunktene vi legger inn i beholderen
- Siden disse objektene ikke skal brukes (eller synes) i noe annet enn beholderklassen vår, kan det være naturlig å opprette dem som en **indre klasse**



```
class YtreKlasse {
    public YtreKlasse() { … }
    protected class IndreKlasse {
        protected IndreKlasse() { … }
    }
}
```

EXCEPTIONS



- Når vi designer beholdere er det naturlig å kunne gi gode feilmeldinger til den som benytter seg av dem
- Java sine egne feilmeldinger kan være misvisende og forvirrende dersom man ikke vet hvordan beholderen fungerer
- En egen exception lages som en klasse, som aller *helst* arver fra en av Java sine innebygde exception-klasser
  - F.eks **Exception** eller **RunTimeException**
- Konstruktøren bør ta inn relevant informasjon om feilen som har skjedd, og lager en feilmeldingsstreng som sendes til super-konstruktøren

«KASTE» EXCEPTIONS

- Dersom man ikke ønsker å behandle feilen akkurat der den forekommer, kan man «kaste» den tilbake til der metoden ble kalt
  - «Fanger» da feilmeldingen med en try/catch-blokk
- Dette gjøres med nøkkelordet **throws:**



```
public hentCelle(int r, int k) throws UtenforRutenettException {
    if (r < 0 || r >= rader || k < 0 || k >= kolonner) {
        throw new UtenforRutenettException(r, k);
    } else { … }
}
```

Pass på at du faktisk håndterer feilmeldingen – ikke la mainmetoden kaste exceptions!

OPPSUMMERING



- Generiske klasser og typeparametere lar oss lage klasser som kan behandle objekter av en hvilken som helst klasse
- Indre klasser kan lønne seg for objekter som kun skal brukes av én annen klasse
- Egne excceptions kan gi bedre feilmeldinger til brukere av koden din

- Ukas tips:
  - Dersom du skal tilegne noe *enten* en verdi *eller* en annen verdi, kan dette gjøres med en «short hand»-if blokk:
    - String resultat = (tall > 100) **?** «Tallet er stort.» **:** «Tallet er lite.»;