Introduksjon til objektorientert programmering

INF1001 Høst 2016
Uke 9

Etter uke 9 skal du

- Kunne designe og implementere en programstruktur med flere klasser
- Kunne etablere og manipulere objekter i (sammensatte) lister og dictionaries
- Kunne tegne klassediagrammer i UML
- Kunne skrive mellomstore programmer

Innhold

- Mentimeter test
- Objektorientert design – et eksempel
  - Hvordan jobber man?
  - Hva er god design?
- Mer verktøy
  - "Magiske" metoder (__lt__, __gt__, __str__, __repr__)
  - Dictionaries med referanser
- Formell dokumentasjon: UML klassediagrammer

Klasser som abstraksjoner

- En klasse representerer typisk «noe» i det domenet et program adresserer
- Nyttig innfallsvinkel til designprosessen
- Sentralt i et mer omfattende design er typisk relasjonene mellom objekter av de ulike klassene

=> Tegn!

God design: Nøkkelkonsepter

- **Løs kobling** mellom klasser: Hver klasse er mest mulig uavhengig, og kommuniserer med andre klasser via små, veldefinerte grensesnitt
- **"Samsvar" (cohesion)** : Om kodeenheter (klasser, prosedyrer, metoder) har ansvaret for veldefinerte, logiske enheter og oppgaver

Virkemidler

- Gjennomfør **innkapsling**: Kode som bruker en klasse skal kun aksessere objekter gjennom grensesnittet (ikke lese eller manipulere instansvariable direkte)
- **Duplisering** av kode bør unngås: Tegn på uheldig ansvarsfordeling mellom kodeenheter

I urtiden (80-tallet) var denne uunnværlig for å planlegge fjelltur mellom DNT-hytter og på T-merkete stier

Stikkord:
- Fjelltur
- Hytter
- Stier

Hva ønsker vi å tilby?

- Finne frem til hytter
  - Navn, antall senger, betjening
- Hvordan komme til en hytte eller mellom to hytter
- Hvor kan man komme fra en hytte på maks antall timer?

Hvilke «ting» handler dette om?

<tegning>

Designvalg

- Hytte som klasse?
  - hva med start- og sluttpunkter (stasjon, p-plass)
  - duplisering av kode
- Pekere til andre steder i et sted-objekt?
  - lite fleksibelt, tette koblinger

Klasser

```
Class Sted :
  def __init__(self, navn, hytte)
    self._navn = navn             # Sted-/ hyttenavn
    self._hytte = hytte           # Hytte eller annet sted?
  …
```

```
Class Rute :
  def __init__(self, fra, til, timer)
    self._fra= fra                # referanse til sted
    self._til= til                # referanse til sted
    self._timer = timer           # tid fra sted til sted
  …
```

Klassen Turplanlegger

```
Class Turplanlegger:
  def __init__(self,omradefil)
    self._omradefil = omradefil
    self._hytter = {}
    self._ruter = []
    <åpne og les data inn i datastrukturen>
  …

  <finn et sted>
  <finn alle ruter kortere enn xx som går til/ fra et sted>
```


Eksempler på alle disse i **navn.py**

"Magiske" metoder

- kalles med annet enn navn, eks
  __init__ kalles ved opprettelse dvs klassens navn
  __eq__ kalles ved bruk av likhetsoperator (==)
- tilsvarende for __lt__ og __gt__:
  __lt__
- Se tabell i kapittel 9.11.3: Oversikt over spesielle metoder i Python: Hvilket navn som brukes i klassedefinisjonen, hva de er tenkt å utføre, og hvordan de kalles

Eksempel fra uke 7

Eksempel: Metoden __eq__

```
class Rektangel :
  def __init__(self, len, bredde) :
    self._lengde = len
    self._bredde = bredde

  def __eq__(self, annen) :
    return (self._lengde == annen._lengde and
            self._bredde == annen._bredde)

r1 = Rektangel(8,6)
r2 = r1
print (r1 == r2)    # samme objekt, alltid True

r3 = Rektangel(6,4)
print (r3 == r2)    # ikke like
```

```
M:> rekt2.py
True
False
M:>
```


Eksempler på alle disse i **navn.py**

"Magiske" metoder II

Ofte nyttig å kunne hente ut innholdet i et objekt som en string, f eks for å skrive ut objektet. Da er disse metodene nyttige i klassen

__str__ returnerer en "menneskevennlig" representasjon av objektet ved kall på str(x)
(denne kalles bak kulissene ved print på et objekt)

__repr__ returnerer en unik og komplett representasjon av objektet ved kall på repr(x)
(denne kalles bak kulissene om du printer en liste med referanser til objekter – eller print på et objekt av en klasse uten __str__)

Samlinger av objekter

- Samlinger (collections) sentralt verktøy i programmering
- Ulike egenskaper – velger ut fra behov
- Så langt har vi sett på
  - Lister (List). Rekkefølge, nummerert
  - Mengder (Set). Unummerert, uten dubletter
  - Ordbøker (Dictionary). Par av nøkkel – verdi

Samlinger av objekter

- Mer presist: Samlinger av objekt-referanser
- Har sett flere eksempler
  - lister av referanser
  - klasser som inneholder lister av referanser (i tillegg til metoder og evt andre instansvariable)
- Dictionaries (ordbøker, maps) kan også ha referanser som verdier, typisk med en attributt (instansvariabel) som nøkkel

Bruk av dictionaries

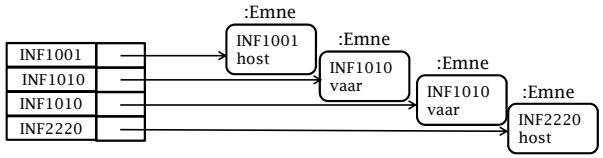
- For effektivt oppslag basert på nøkler
- Opprette og aksessere vha []
- Ville vært en nyttig struktur for representasjon av Steder i Turplanleggeren vår
- Kunne slått opp basert på stedsnavn.
- Eller lett gjennom for å finne alle som var hytter og hadde mer enn 10 sengeplasser

Eksempel: Informatikk-emner (kurs)

- Vi skal lage et program for å velge informatikk-emner
- Initielle krav: Kunne liste opp alle emner med id, antall poeng, tittel og høst eller vår-semester
- Velger en klasse Emne, med instansvariable som over
- Bruker List-klassen for å organisere Emne-objekter

Dictionary av objekter

- Eksempel: Emnekatalog
- Nøkkel (entydig): Emnekode
- Verdi: Referanse til et Emne-objekt



Eksempel: Dictionary med emner

```
emnekatalog = {}            #opprettet tom katalog

while <flere emner>
  <les inn emnekode, semester og poeng>
  nytt = Emne(emnekode,semester,poeng)
  emnekatalog[emnekode] = nytt
```


Å tegne en struktur

- Har så langt tegnet objekter. Disse viser (et utdrag av) enheter som finnes i minnet på et gitt tidspunkt under kjøring – objekter, med hver sine sett instansvariable som har verdier. "Snapshot"

  (bilde av en eller flere pannekaker)

- Kan også være nyttig å dokumentere klasser. Et klassediagram er en illustrasjon av klassedefinisjonen(e), altså hvordan programkoden ser ut. Man kan se hva som er definert i klassen og dermed *potensialet*, men det vises ingen objekter og dermed ingen instansvariabler som kan ha verdi.

  (bilde av en pannekakeoppskrift)

Unified Modeling Language (UML)

Å tegne datastrukturer kan være nyttig

- under planlegging (design) av et nytt program, før koding
- for å diskutere (alternative) løsninger med andre
- som dokumentasjon

For eget bruk: Mindre betydning *hvordan* man tegner. Ved skriftlig kommunikasjon kan det være nyttig å bruke en mer formell notasjon

En modell er alltid en forenkling – må vurdere hva som skal formidles i hvert tilfelle

- UML er den meste kjente standarden innen systemutvikling

Unified Modeling Language (UML)

UML
+ En formell standard og mest kjent: Kan leses av "alle"
+ Det finnes verktøy for enkelt å lage diagrammer

- Standarden er *stor* og omfatter ca 20 typer diagrammer
- Vi bruker en praktisk tilpasning av *klassediagrammer*

- Klassediagrammer er de vanligste til analyse og design, og kan "oversettes" direkte til objektorientere språk

UML klassediagrammer m/ assosiasjoner

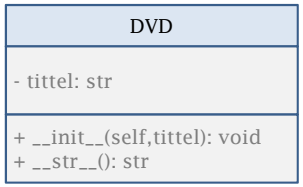
- Nyttig når vi jobber med datastrukturer med flere klasser og relasjoner mellom objekter
  - Design: hvilke klasser, hvilke relasjoner
  - Implementering: oversikt over pekere og objekter
  - (Dokumentasjon: Større krav til formalisme)

- To typer informasjon:
  - om hver enkelt klasse
  - "assosiasjoner" mellom klasser, uformelt: "peker til" (tar ikke stilling til nyanser som også kan uttrykkes i UML)

- Vurder i hvert tilfelle hvilken informasjon som trengs
  (eksamen: Les oppgaveteksten ☺)

Enkeltklasse

- Alltid med: Klassens navn
- Etter behov/ plass: Datarepresentasjonen
- Etter behov/ plass: Grensesnittet

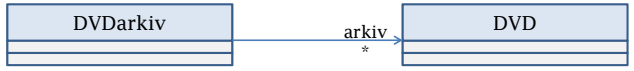


'-' (eller lås-symbol) angir **`private`**
'+' (eller åpen lås) angir **`public`**

Forhold mellom klasser I

- Uformelt: En *pil* fra A til B illustrerer behov (design) eller mulighet (dokumentasjon) for å navigere fra objekter av klassen A til objekter av klassen B

- Navnet på pilen sier noe om hva referansen representerer
- Tallet angir antall objekter som kan refereres fra ett objekt
  - * betyr 0 eller flere
  - 1..* betyr en eller flere
  - 0..2 betyr 0, 1 eller 2
  - 5 betyr alltid 5



Forhold mellom klasser II

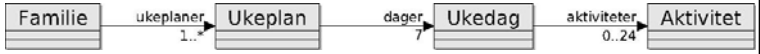
- Hvis det er referanser begge veier, bruker vi en *strek*
- Leses begge veier, tenker oss alltid at vi starter i ett objekt og ser hvor mange objekter dette kan referere til:
  - Én person eier 0 til mange biler
  - Én bil er eid av minst 1 person

- (vi ønsker å kunne finne eieren til et bilobjekt – og vi ønsker å finne ut hvilke(n) bil(er) en person eier)

Fra eksamen 2014

Du har påtatt deg å lage ukeplaner for hele familiens faste aktiviteter. For å gjøre det enkelt antar vi at alle aktiviteter starter på en hel time (kl 00.00, 01.00, 02.00 etc) og at de alle varer nøyaktig 1 time.

Du skal bruke klassene vist i dette UML klassediagrammet:



NB: Klassediagrammet er statisk

⇒ viser ikke øyeblikksbilder av objekter under kjøring – bare *potensialet* i datarepresentasjonen!

⇒ Vi vil fortsatt bruke mer uformelle tegninger for å vise eksempler på hvilke objekter og pekere vi har under kjøring