Objektorientert programmering i Python

IN1000

Høst 2020 – uke 12

Siri Moe Jensen

Planlagt innhold uke 12

Opprinnelig planlagt forelesning ligger fra uke12-siden sammen med det som ble gjennomgått:

- Ulike typer feil
  - Syntaksfeil
  - Logiske feil
- Unntak
  - Feil som kan fanges opp
  - Håndtering av unntak
- Større programmer
  - Refaktorering
  - Enhetstesting
  - Dokumentasjon

Dette blir det i stedet..

- Resten av semesteret
- Grafisk fremstilling av datastrukturer (klassediagrammer, objektdiagrammer, datastruktur-tegninger, ..)
- Noen sentrale begreper og temaer i IN1000. Mentimeter.

Resten av semesteret

on. 4. nov.	10:15–12:00	2 * 45 minutter (ca) på Zoom	Digital undervisning Zoom	▪ S. A. M. Jensen ▪ G. K. Sandve	Grafisk fremstilling av datastrukturer Mentimeter oppgaver om sentrale punkter i pensum. Ressurser for uke 12
on. 11. nov.	10:15–12:00	2 * 45 minutter (ca) på Zoom	Digital undervisning Zoom	▪ S. A. M. Jensen ▪ G. K. Sandve	Informasjon og tips til eksamen og eksamensforberedelser.
on. 18. nov.	10:15–12:00	2 * 45 minutter (ca) på Zoom	Digital undervisning Zoom	▪ S. A. M. Jensen ▪ G. K. Sandve	Gjennomgang av prøveeksamen.

Å tegne en struktur

Modell (fra uke 1): En forenkling!!

1. Vi har tegnet **objekter** og referanser for å se hva som skjer **under kjøring**. Disse viser (et utdrag av) enheter som finnes i minnet på et gitt tidspunkt under kjøring: objekter

- "Snapshot"!

=> **uformelle tegninger**, forståelse, eksperimentering, hvilke instruksjoner trengs

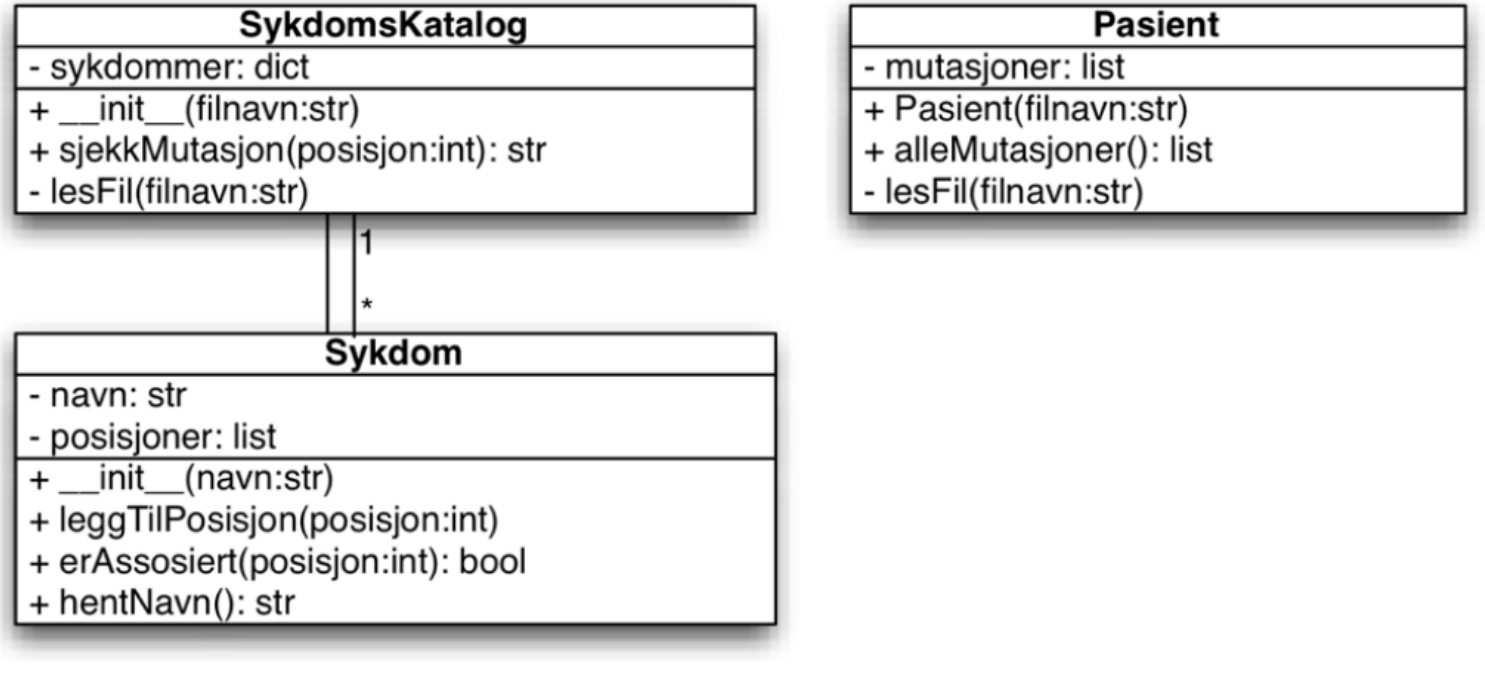
2. Kan også være nyttig å dokumentere klasser med et **klassediagram**

- Viser klassedefinisjonen(e), altså hvordan *programkoden* ser ut. Hva er definert i klassen: Potensialet
- Viser ingen objekter og dermed ingen instansvariabler som kan ha verdi
- Brukes ofte til kommunikasjon med andre: Spefikasjon, dokumentasjon

$\Rightarrow$ Nyttig med mer formell notasjon: UML (Unified Modelling Language)

Fra forrige ukes forelesning

Alle tre klassene i UML



Hva sier oppgaven i oblig 7?

Oppgave 1 Klassen **Sang**

Du skal skrive en klasse **Sang** som en egen modul i filen sang.py, med følgende grensesnitt:

Klassen skal ha en *konstruktør* med parametere for tittel og artist (i tillegg til **self**). Konstruktøren skal opprette instansvariabler **_artist** og **_tittel**. Disse er strenger som får verdi fra parameterne. Instansvariabelen **_artist** er en streng som består av en eller flere bokstavsekvenser atskilt av blanke tegn. Eksempler: "Simon and Garfunkel", "The Rolling Stones", "Prince".

Klassen skal dessuten tilby følgende metoder i grensesnittet:

spill "spiller av" musikken i sangen den kalles for – i dette programmet betyr det at den skriver meldingen "Spiller <info om tittel og artist>" ut på terminalen. *Ligger også en frivillig utvidelse på obligsiden, med denne utvidelsen kan du få faktiske sanger til å bli spilt av på PC'en din.*

sjekkArtist med parameter **navn** (en streng) på samme form som instansvariabelen **_artist**. Metoden returnerer **True** dersom ett eller flere av navnene i strengen **navn** finnes i **_artist**, ellers **False**.

sjekkTittel med parameter **tittel** (en streng). Metoden sjekker om oppgitt tittel er den samme som i instansvariabelen og returnerer **True** ved likhet, ellers **False**. Titlene skal defineres som like uavhengig av små/ store bokstaver.

sjekkArtistOgTittel med parametere **artist** og **tittel**. Metoden returnerer **True** dersom både tittelen og artisten (samme regler som i de tilsvarende sjekk-metodene) stemmer med sangens instansvariabler, ellers **False**.

Oppgave 2 Klassen **Spilleliste**

I filen *spilleliste.py* finner du definisjon av klassen **Spilleliste**. Konstruktøren er ferdig skrevet, mens metodene under skal ferdigstilles av deg (se rammen med **Frivillig utvidelse** ovenfor for utvidelser). Det forventes at du bruker metodene til Sang objektet der det er mulig.

lesFraFil med parametere **self** og **filnavn**. Metoden åpner den oppgitte filen, og leser inn data om sanger – en linje per sang, på formatet

```
tittel;artist
```

Siden både tittel og artist kan inneholde blanke tegn, brukes semikolon som skilletegn. Merk også at det er lurt å fjerne tegn for linjeskift på slutten av hver linje. En linje kan for eksempel leses slik:

```
alleData = linje.strip().split(';')
```

Metoden skal opprette nye **Sang**-objekter etter hvert som filen leses, og legge disse inn i spillelisten. Filen lukkes til slutt.

leggTilSang med parametere **self** og **nySang**, der **nySang** er det nye objektet som skal legges til spillelisten.

fjernSang med parametere **self** og **sang**.

spillSang med parametere **self** og **sang**.

spillAlle med parameter **self**, som spiller hver enkelt sang i listen.

finnSang med parametere **self** og **tittel**, som leter gjennom listen av sanger etter en med oppgitt tittel og returnerer den første den finner. Finnes ikke tittelen i spillelisten returneres **None**.

hentArtistUtvalg med parametere **self** og **artistnavn**. Metoden går gjennom alle sanger i spillelisten og tar vare på de som har en eller flere navn fra parameteren **artistnavn** i navnet på artisten. Disse returneres i en liste med sanger.

Klasser med relasjoner

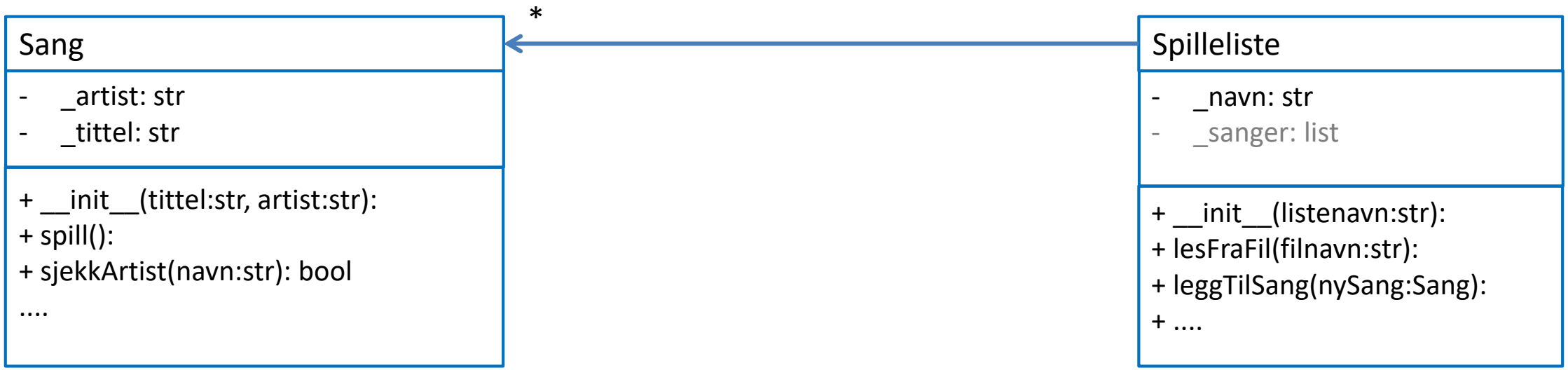
Oppgave 1 Klassen Sang

Du skal skrive en klasse **Sang** som en egen modul i filen sang.py, med følgende grensesnitt:

Klassen skal ha en *konstruktør* med parametere for tittel og artist (i tillegg til **self**). Konstruktøren skal opprette instansvariabler **_artist** og **_tittel**. Disse er strenger som får verdi fra parameterne.

```
from sang import Sang

class Spilleliste:
    def __init__(self, listenavn):
        self._sanger = []
        self._navn = listenavn
```



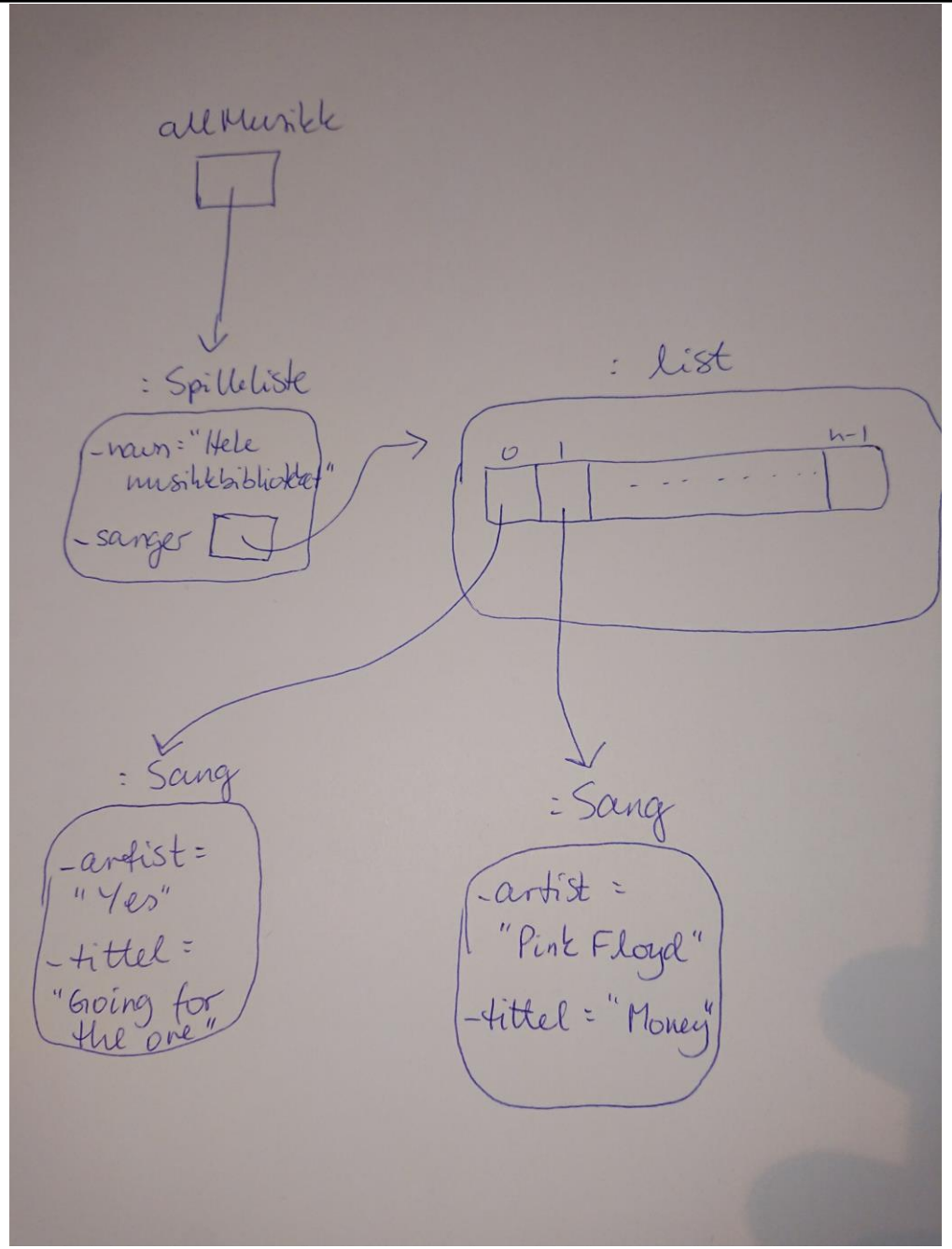
Klasser med relasjoner

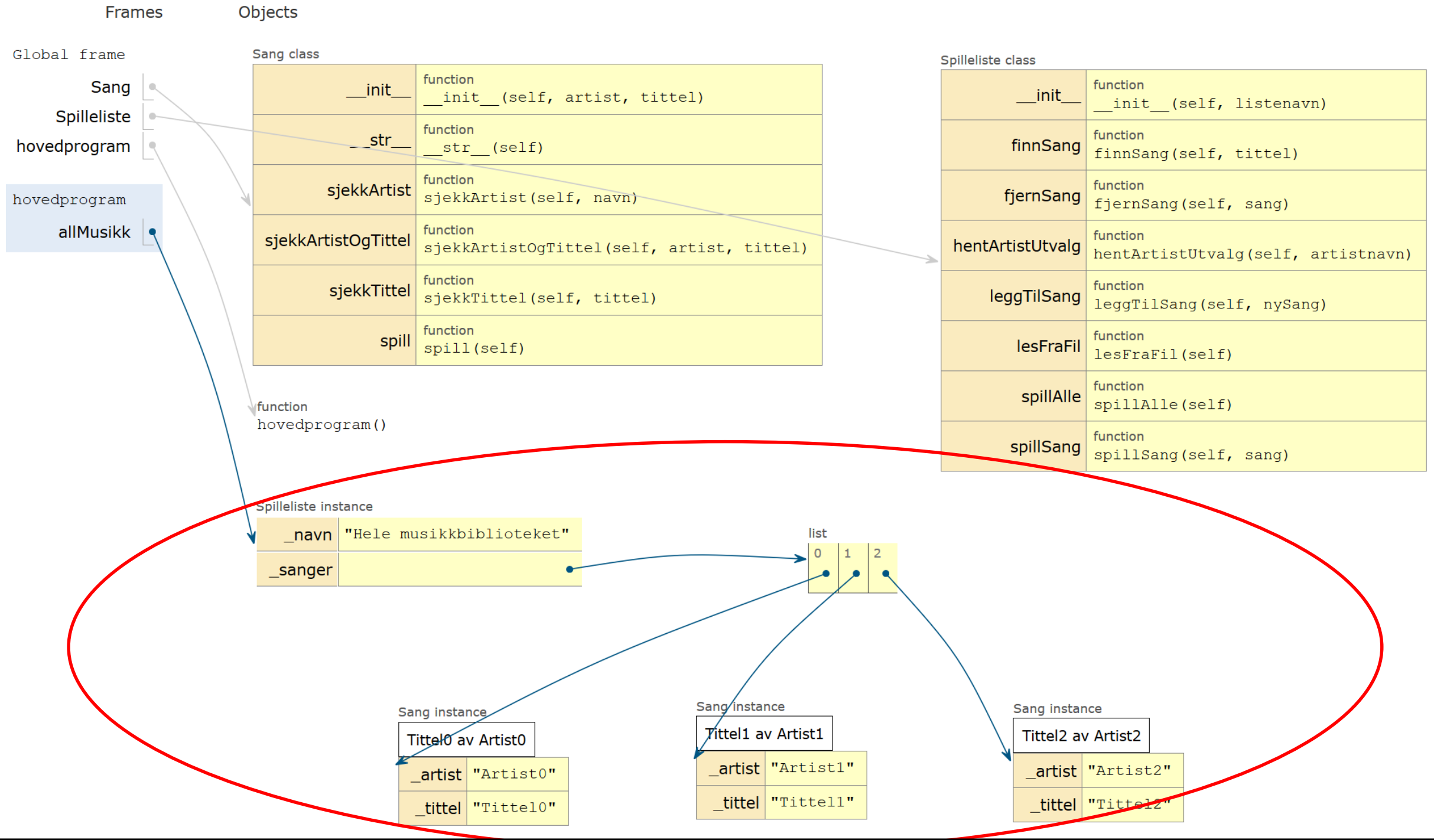
- Denne figuren sier noe om programmet vårt, mer spesifikt *klasse-definisjonene*
- Hvilke attributter (implementert som instansvariabler) og metoder definerer klassene?
- Hvilke data *kan* objekter representere*, når de opprettes?*
- Her finnes ingen objekter – kun klassene som beskriver mønstere for objekter
- Det vil si: Ingen instansvariabler finnes, og de kan dermed heller ikke ha verdi

Datastruktur-tegning oblig7

datastrukturen i programmet *spillelistetester* slik den ser ut etter at en spilleliste er lest inn fra filen musikk.txt. Du trenger ikke tegne mer enn to Sang-objekter.

Bruk notasjonen fra forelesningene, med variabler som bokser, objekter som sirkler eller bokser med runde hjørner, og referanser som piler fra en variabel til et objekt. Du kan tegne strenger som innhold i enkle variabler, mens en liste tegnes som et objekt.




Frames
Objects
Global frame
Sang
Spilleliste
hovedprogram
hovedprogram
allMusikk
Sang class
__init__
function
__init__(self, artist, tittel)
__str__
function
__str__(self)
sjekkArtist
function
sjekkArtist(self, navn)
sjekkArtistOgTittel
function
sjekkArtistOgTittel(self, artist, tittel)
sjekkTittel
function
sjekkTittel(self, tittel)
spill
function
spill(self)
Spilleliste class
__init__
function
__init__(self, listenavn)
finnSang
function
finnSang(self, tittel)
fjernSang
function
fjernSang(self, sang)
hentArtistUtvalg
function
hentArtistUtvalg(self, artistnavn)
leggTilSang
function
leggTilSang(self, nySang)
lesFraFil
function
lesFraFil(self)
spillAlle
function
spillAlle(self)
spillSang
function
spillSang(self, sang)
function
hovedprogram()
Spilleliste instance
_navn
"Hele musikkbiblioteket"
_sanger
list
0
1
2
Sang instance
Tittel0 av Artist0
_artist
"Artist0"
_tittel
"Tittel0"
Sang instance
Tittel1 av Artist1
_artist
"Artist1"
_tittel
"Tittel1"
Sang instance
Tittel2 av Artist2
_artist
"Artist2"
_tittel
"Tittel2"

Unified Modeling Language (UML)

UML

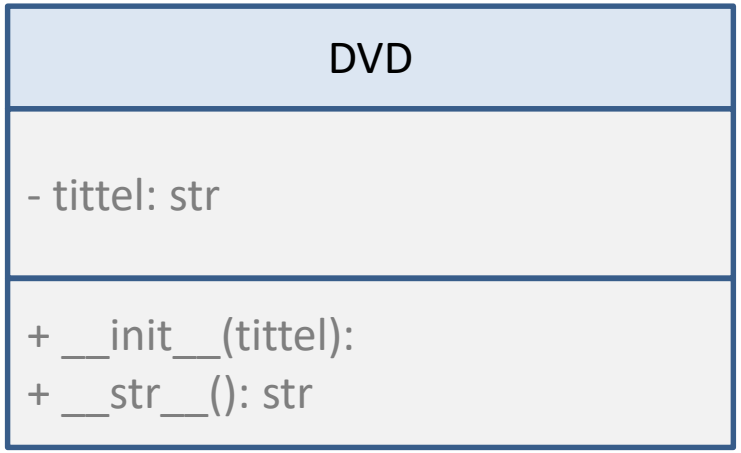
+ En formell standard og mest kjent: Kan leses av "alle"
+ Det finnes verktøy for bl.a. å enkelt lage diagrammer

- Standarden er *stor* og omfatter 20+ ulike typer diagrammer
- Kan være mer eller mindre detaljerte, og vise ulike aspekter av en spesifikasjon/ implementasjon
- Vi bruker en praktisk tilpasning/ forenkling av *klassediagrammer*

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

Enkeltklasse

- Alltid med: Klassens navn
- Etter behov/ plass: Datarepresentasjonen
- Etter behov/ plass: Grensesnittet og evt non-public metoder

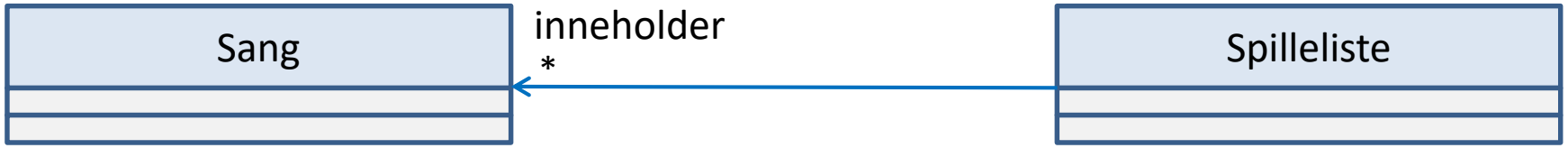


'-' (eller lås-symbol) angir **`non-public`**

'+' (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

$\Rightarrow$ viser ikke øyeblikksbilder av objekter under kjøring – bare *potensialet* i datarepresentasjonen!

$\Rightarrow$ Vi vil fortsatt bruke mer uformelle tegninger for å vise eksempler på hvilke objekter og referanser vi har på et bestemt sted i koden *under kjøring*

Tegn et klassediagram

Datasenter og regneklyngens bestanddeler

I denne oppgaven består et `Datasenter` av en eller flere regneklynger. En `Regneklynge` består av ett eller flere `Rack` (et kabinett med skinner) hvor mange *noder* kan monteres over hverandre. En `Node` er en selvstendig maskin med et hovedkort med én eller flere prosessorer og minne, i tillegg til en del andre ting. I denne oppgaven skal vi bare se på antall prosessorer (maks 2) og størrelsen på minnet. Du kan derfor anta at en node har en eller to prosessorer og et heltallig antall GB med minne.

Programdesign

Programmet skal designes med fire klasser som representerer noder, racks, regneklynge og datasenter. Et objekt av klassen `Datasenter` skal kunne referere til ett eller flere objekter av klassen regneklynge. Et objekt av klassen `Regneklynge` skal kunne referere til ett eller flere rack-objekter, der hvert rack-objekt igjen refererer til en eller flere node-objekter. Noen flere krav og tips til design av den enkelte klassen finner du videre i oppgaven.

Go to www.menti.com and use the code 22 85 89 5