

A decorative graphic consisting of a thin yellow circle. A thick black left square bracket is positioned on the left side of the circle, and a thick yellow right square bracket is on the right side. A horizontal bar with a gradient from olive green to white is overlaid across the middle of the circle.

Klasser og Objekter i Python

Uge 46

Learning Python: kap 15-16,
19-22.

[Klasser og objekter]

- En klasse beskriver en klump af samhørende funktioner og variable
- En klasse er en **beskrivelse**.
 - En kage form
- Klassens objekter er **instanser** af klassen.
 - De enkelte kager
- En programudførelse indeholder objekter, ikke klasser

[Opgave]

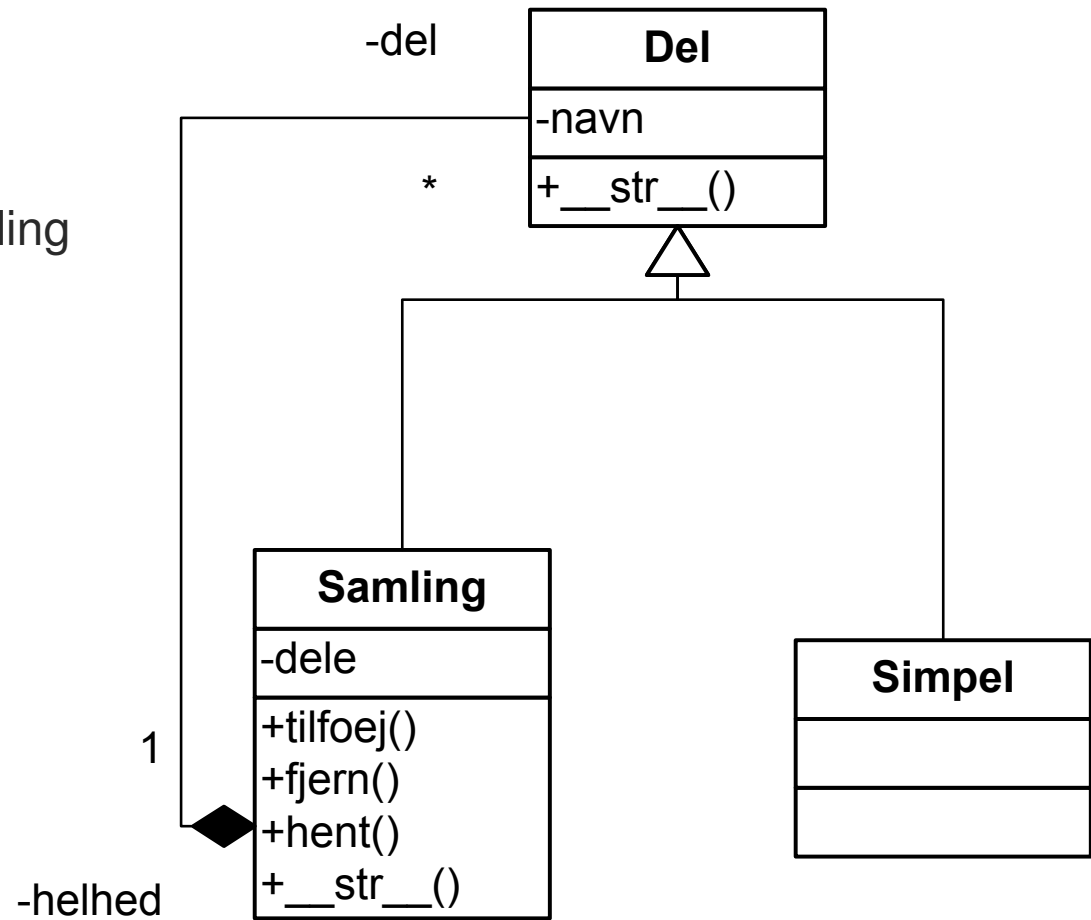
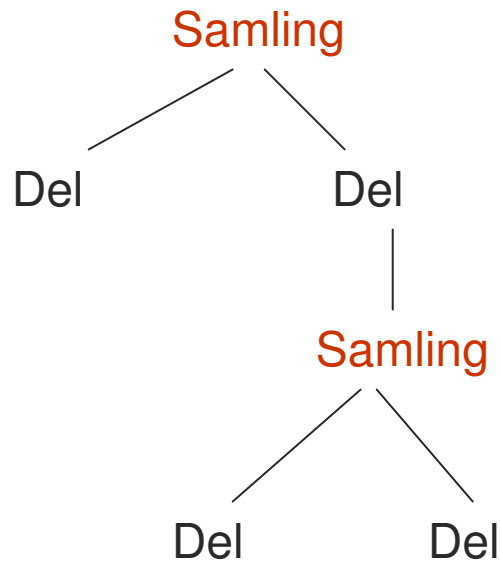
- [materialer\UML opgave 2 08.doc](#)

['Composite' mønstret]

- Composite mønstret kan bruges til at repræsentere hierarkier af helheder og dele
 - En bil består af karosseri, hjul og motor
 - Et karosseri består af tag, døre og bund
 - En motor består af stempler og karburator
- Composite mønstret er en abstraktion over alle den slags hierarkier

['Composite' mønstret]

- Rekursiv definition
- $\text{Samling} ::= \{\text{Del}\}^*$
- $\text{Del} ::= \text{Simpel} \mid \text{Samling}$



[Nedarvning]

- Subklassen arver variable og funktioner fra superklassen

Begge arver navn fra overklassen

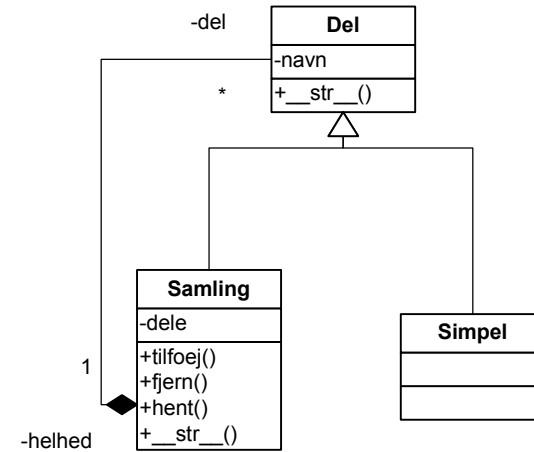
- **Simpel:**
 - navn
 - `__str__`

- **Samling:**
 - navn
 - dele
 - `__str__`
 - tilføj
 - fjern
 - hent

Samling tilføjer nye ting

Simpel bevarer superklassens `__str__`

Samling overskriver superklassens `'__str__'`



init og andre funktioner

```
class Del:  
    def __init__(self, navn):  
        self.navn = navn  
    def __str__(self, niveau = 1):  
        text = '\n' + niveau * '+' + self.navn + '  
        return text
```

superklasse

```
class Samling(Del):  
    def __init__(self, navn, dele = None):  
        Del.__init__(self, navn)  
        self.dele = dele  
    def tilfoej(self, enDel):  
        self.dele.append(enDel)  
    def fjern(self, enDel):  
        if enDel in self.dele:  
            self.dele.remove(enDel)  
    def hent(self, delNavn):  
        for d in self.dele:  
            if d.navn == delNavn:  
                return d  
        return None  
    def __str__(self, niveau = 1):  
        text = '\n' + niveau * '+' + self.navn  
        for enDel in self.dele:  
            text += enDel.__str__(niveau+1)  
        text += '\n' + niveau * '+'  
        return text
```

initialisering

funktioner

subklasse

```
class Sempel(Del):  
    def __init__(self, navn):  
        Del.__init__(self, navn)
```

subklasse

[Dannelselse af objekter]

Inits parametre

- `bil = Samling('bil',[Samling('karosseri'),Samling('hjul'),Samling('motor')])`
- `bil.hent('karosseri').tilfoej(Simpel('Tag'))`
- `bil.hent('karosseri').tilfoej(Simpel('Doere'))`
- `bil.hent('karosseri').tilfoej(Simpel('Bund'))`
- `bil.hent('hjul').tilfoej(Simpel('venstre forhjul'))` ← Kald af hjulsamlingens tiltøj-funktion
- `bil.hent('hjul').tilfoej(Simpel('venstre forhjul'))`
- `bil.hent('hjul').tilfoej(Simpel('hoejre forhjul'))`
- `bil.hent('hjul').tilfoej(Simpel('venstre baghjul'))`
- `bil.hent('hjul').tilfoej(Simpel('hoejre baghjul'))`
- `bil.hent('motor').tilfoej(Simpel('stempler'))`
- `bil.hent('motor').tilfoej(Simpel('karburator'))`
- `print bil`

[Output

- (bil
- (karosseri
- (Tag)
- (Doere)
- (Bund)
-)
- (hjul
- (venstre forhjul)
- (venstre forhjul)
- (hoejre forhjul)
- (venstre baghjul)
- (hoejre baghjul)
-)
- (motor
- (stempler)
- (karburator)
-)
-)

Definition af klasser:

- Class <klassenavn>(<evt. superklasse>):
 - <init-funktionen>
 - <funktioner der tilhører klassen>

klassenavn →

```
class Samling(Del):  
    """repræsenterer en samling af dele, enten samlinger eller simple"""  
    def __init__(self, navn, dele = None):  
        Del.__init__(self, navn)  
        self.dele = dele  
    def tilfoej(self, enDel):  
        if self.dele == None:  
            self.dele = []  
            self.dele.append(enDel)  
    def fjern(self, enDel):  
        if self.dele == None:  
            self.dele = []  
        if enDel in self.dele:  
            self.dele.remove(enDel)  
    def hent(self, delNavn):  
        for d in self.dele:  
            if d.navn == delNavn:  
                return d  
        return None  
    def __str__(self, niveau = 1):  
        """giver en tekstuel repræsentation af klassen  
        __str__ gør det muligt at printe klassen direkte"""  
        text = '\n' + niveau * '+' + self.navn  
        for enDel in self.dele:  
            text += enDel.__str__(niveau+1)  
        text += '\n' + niveau * '+'  
        return text
```

superklasse →

Init funktionen

Funktioner der tilhører klassen

Check om det er rigtigt: Introspektion

- Python indeholder en række funktioner der giver direkte adgang til dens indvolde
- `Objekt.__dict__`: en dictionary der gemmer objektets attributter og værdier
- `Dir(object)`: returnerer alle de attributter og metoder der er knyttet til objektet.

[Introspection]

- `def introspect(self):`
- `text = 'Dictionary: '`
- `text += '\n'+ str(self.__dict__)`
- `text += '\n'+ 'Attributes and functions: '`
- `text += '\n'+ str(dir(self))`
- `return text`

[Eksempel]

■ Bil (Samling):

○ Dictionary:

- {'dele': [<__main__.Samling instance at 0x00FC2BE8>, <__main__.Samling instance at 0x00FC2CD8>, <__main__.Samling instance at 0x00FC2D00>],
- 'navn': 'bil'}

○ Attributes and Functions :

- ['__doc__', '__init__', '__module__',
- '__str__', 'dele', 'fjern', 'hent', 'introspect', 'navn', 'tilfoej']

Nedarvede fra Python

■ Tag (Simpel):

○ Dictionary:

- {'navn': 'Tag'}

○ Attributes and Functions :

- ['__doc__', '__init__', '__module__',
- '__str__', 'introspect', 'navn']

Brugerdefinerede

[Generering af klasser]

- <klassenavn>(<inits parametre undtagen self>)

- Klasse definition

- class Simpel(Del):
- def __init__(self, navn):
- **Del.__init__(self, navn)**

Simpel's init kalder
superklassens init

Husk at give den
parametren self med!!

- Generering af klasse

- **Forhjul = Simpel('venstre forhjul')**

- `__init__` kaldes automatisk når klassen genereres

[Self]

- **self** henviser til det nydannede objekt
- Når man i klassedefinitionen vil henviser til objektets egne funktioner eller variable skal det ske via **self**
- `def getName(self):`
- `return self.name`

[Med og uden self]

- class Samling(Del):
- def __init__(self, navn, dele):
- Del.__init__(self, navn)
- self.dele = dele
- **stelnummer = 100**
- 'Stelnummer' er blot en lokal variabel i __init__ der forsvinder når funktionen har kørt. 'Dele' er derimod fast knyttet til objektet
- >>> bil.dele
- [<__main__.Simpel instance>, <__main__.Samling instance >, <__main__.Simpel instance >]
- >>> bil.stelnummer
- **Traceback (most recent call last):**
- **File "<pyshell#7>", line 1, in ?**
- **bil.stelnummer**
- **AttributeError: Samling instance has no attribute 'stelnummer'**
- >>>

Self bruges også til at referere til objektets egne funktioner

- class Eksempel:
 - def __init__(self,enListe):
 - self.minListe = enListe
 - def sommer(self):
 - resultat = 0
 - for i in self.minListe:
 - resultat += i
 - return resultat
 - def udskrivSum(self):
 - print self.sommer()
- etElement = Eksempel([1,2,3,4])
- etElement.udskrivSum()
- Resultat:
- 10
- >>>

Eksempel
-minListe
+summer()
+udskrivSum()

[Udeladelse af self]

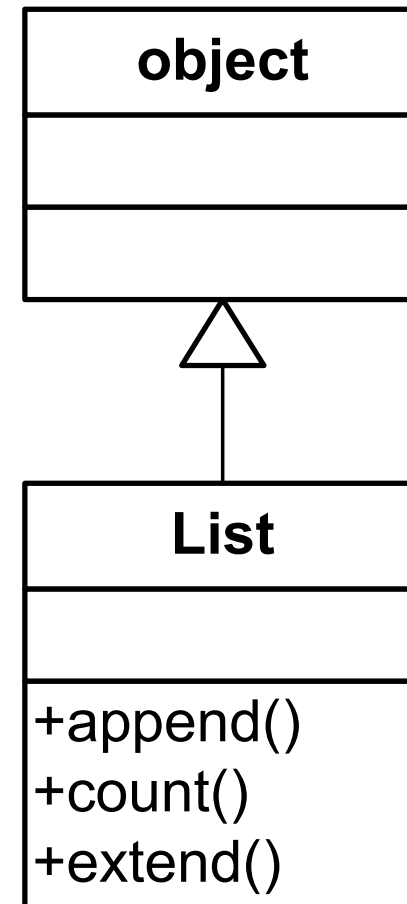
- `def udskrivSum(self):`
- `print summer()`
- Python ved ikke hvad 'summer' refererer til:
 - `print summer()`
 - `NameError: global name 'summer' is not defined`

[Benyttelse af klasser]

- Man kan godt bruge klasser's funktioner (**ubundne funktioner**) uden at lave en instans, men så skal de have en instans som argument:
- `etElement = Eksempel([1,2,3,4])`
- `Eksempel.udskrivSum(etElement)`

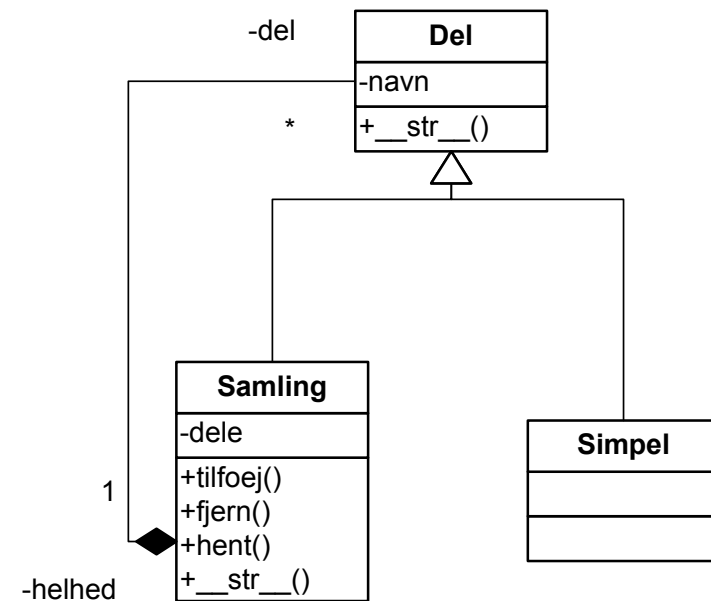
[Hele Python består af objekter]

- `>>> L = [1,2,3]`
- `>>> L.append(4)`
- `>>> L`
- `[1, 2, 3, 4]`
- `>>>`



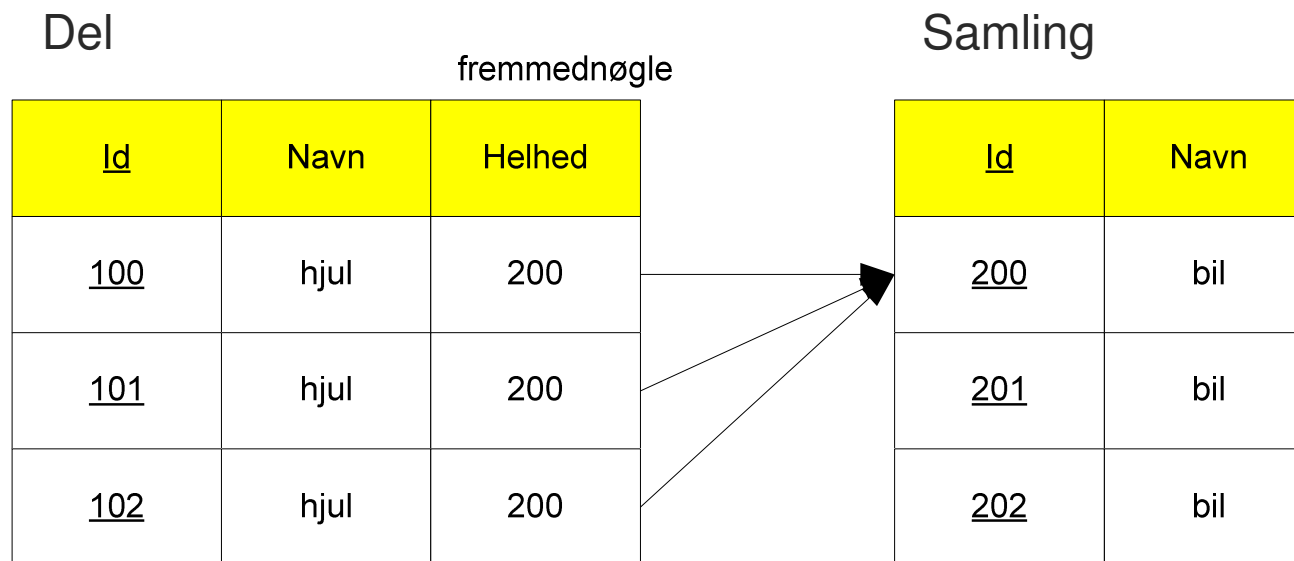
[Relationer mellem objekter]

- Der er en én-mange relation mellem en samling og dens dele
- En relation er et abstrakt begreb der kan implementeres på mange måder.
- SQL: som en relation
- Python: som en liste af objekter



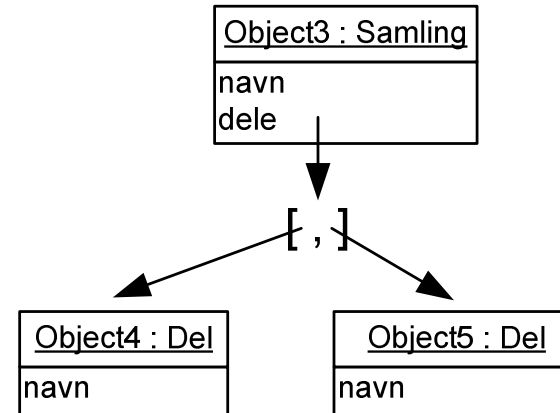
[SQL

- Helhedens primærnøgle benyttes som fremmednøgle i delen



[Python 1]

- `class Samling(Del):`
 - `def __init__(self, navn, dele):`
 - `Del.__init__(self, navn)`
 - `self.dele = dele`
-
- Helheden har en liste over de objekter der repræsenterer dens dele



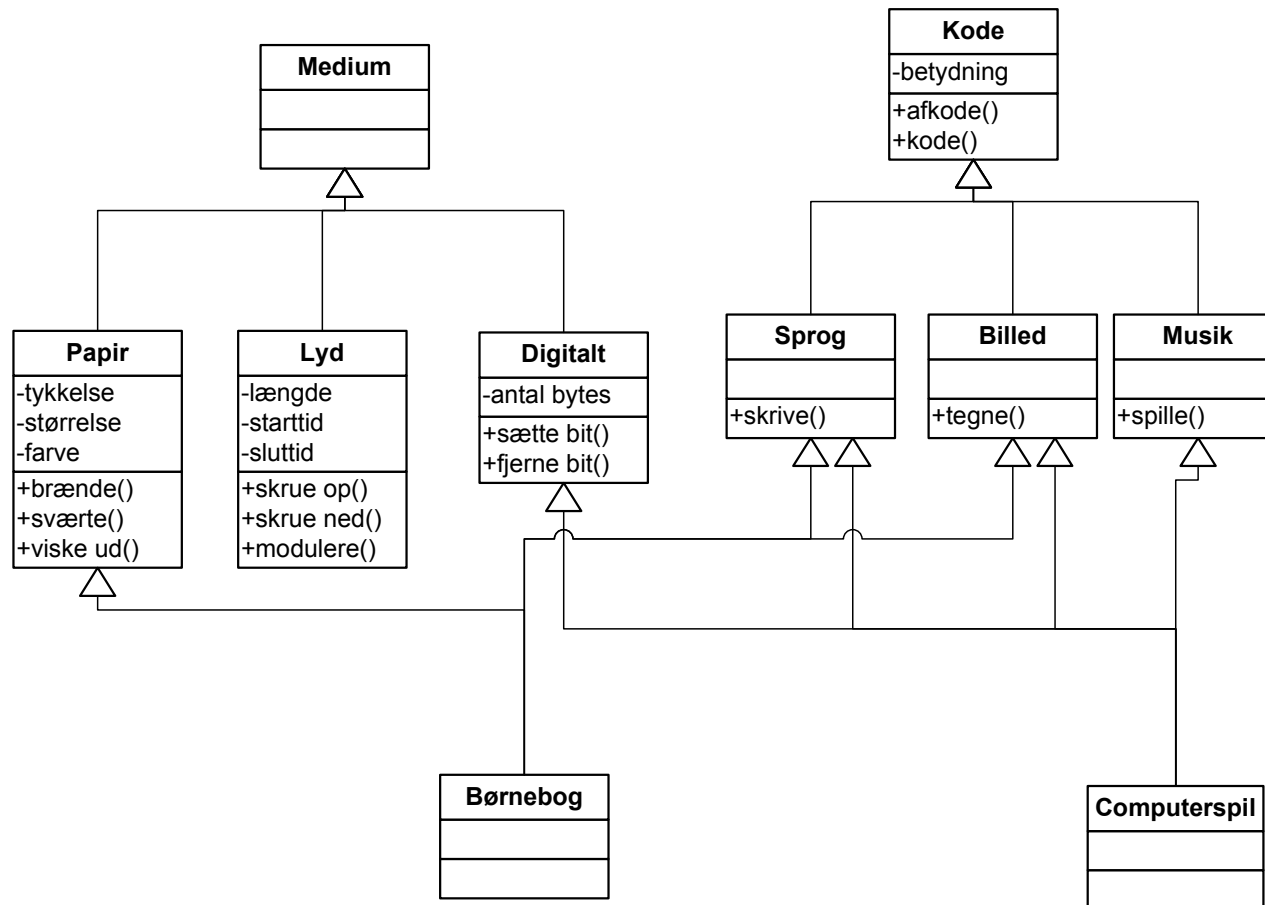
[Python 2]

- Den samme UML-relation kan implementeres på forskellig måde
- I stedet for en liste kan man bruge en dictionary

[Python 3]

```
■ class Samling(Del):
■     """repræsenterer en samling af dele, enten samlinger eller simple"""
■     def __init__(self, navn):
■         Del.__init__(self, navn)
■         self.dele = {}
■     def tilfoej(self, enDel):
■         self.dele[enDel.navn] = enDel
■     def fjern(self, delNavn):
■         if delNavn in self.dele.keys():
■             self.dele.pop(delNavn)
■     def hent(self, delNavn):
■         if delNavn in self.dele.keys():
■             return self.dele[delNavn]
■         else:
■             return None
■     def __str__(self, niveau = 1):
■         """giver en tekstuel repræsentation af klassen"""
■         text = '\n' + niveau * '+' + self.navn
■         for enDel in self.dele.values():
■             text += enDel.__str__(niveau+1)
■         text += '\n' + niveau * '+'
■         return text
```

[Multipel nedarvning]



[Kode]

```
class Medium:
    def __init__(self, navn):
        self.navn = navn
    def introspect(self):
        text = 'Id: '+str(id(self))
        text += '\nDictionary: '
        text += '\n'+ str(self.__dict__)
        text += '\n'+ 'Attributes and functions: '
        text += '\n'+ str(dir(self))
        return text

class Papir(Medium):
    def __init__(self,navn, tykkelse, stoerrelse, farve):
        Medium.__init__(self,navn)
        self.tykkelse = tykkelse
        self.stoerrelse = stoerrelse
        self.farve = farve
    def braende(self):
        pass
    def svaerte(self):
        pass
    def viske_ud(self):
        pass

class Lyd(Medium):
    def __init__(self,navn, laengde, starttid, sluttid):
        Medium.__init__(self,navn)
        self.laengde = laengde
        self.starttid = starttid
        self.sluttid = sluttid

    def skrue_op(self):
        pass
    def skrue_ned(self):
        pass
    def modulere(self):
        pass
```

```
class Digitalt(Medium):
    def __init__(self,navn, antal_bytes):
        Medium.__init__(self,navn)
        self.antal_bytes = antal_bytes

    def saette_bit(self):
        pass
    def fjerne_bit(self):
        pass

class Kode:
    def __init__(self,betydning=""):
        self.betydning = betydning

class Sprog(Kode):
    def __init__(self,betydning):
        Kode.__init__(self,betydning)
    def skrive(self):
        pass

class Billed(Kode):
    def __init__(self,betydning):
        Kode.__init__(self,betydning)
    def tegne(self):
        pass

class Musik(Kode):
    def __init__(self,betydning):
        Kode.__init__(self,betydning)
    def skrive(self):
        pass

class BoerneBog(Papir,Sprog,Billed):
    def __init__(self,navn,tykkelse,stoerrelse, farve,betydning):
        Papir.__init__(self,navn, tykkelse, stoerrelse, farve)
        Sprog.__init__(self,'eventyr')
        Billed.__init__(self,'eventyr')
```

[Hvad er der indeni?]

- `fyrtoejet = BoerneBog('Fyrtoejet','2mm','A4','hvid','eventyr')`
- `print fyrtoejet.introspect()`

- Id: 26353296
- **Dictionary:**
- `{'tykkelse': '2mm', 'stoerrelse': 'A4', 'betydning': 'eventyr', 'farve': 'hvid', 'navn': 'Fyrtoejet'}`
- **Attributes and functions:**
- `[... 'betydning', 'braende', 'farve', ... 'navn', 'skrive', 'stoerrelse', 'svaerte', 'tegne', 'tykkelse', 'viske_ud']`

[To typer af UML-diagrammer]

- Analyse diagrammer: repræsenterer den virkelige verden.
- Design diagrammer: repræsenterer IT-systemet.

[Fordelen ved UML]

- Vi kan beskrive de logiske forhold i problemområdet uden at skulle tage beslutninger om implementeringen
 - At kunne dette er jeres hovedkvalifikation.
- Har vi et godt UML diagram er der ikke lang vej til implementering for dygtige programmører.

[Opgave 9]

- ..\øvelser\øvelse 9.doc