Mönster och datastrukturer, del 2

Data- och informationsvetenskap: Objektorienterad programmering och modellering för IA

Johan Holmberg 2019-10-22

MALMÖ UNIVERSITET

Dagens agenda

- Förra föreläsningen
- Datastrukturer

MALMÖ UNIVERSITET

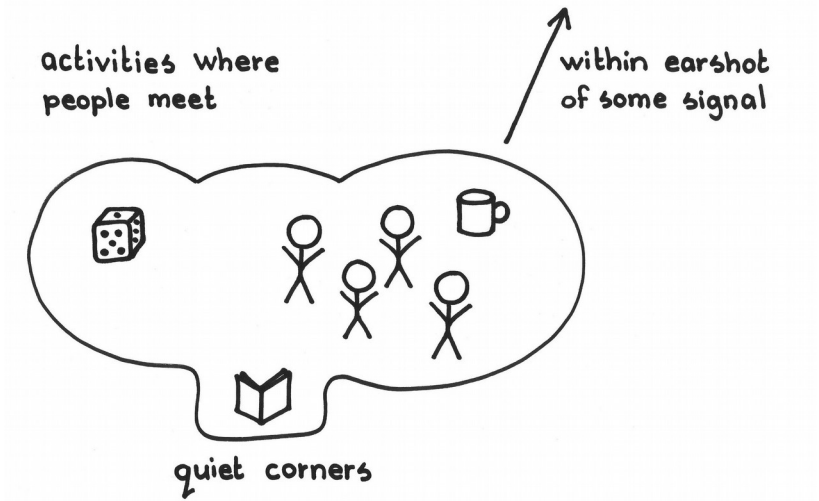
Vad är ett mönster?

Mönster är generella lösningar på vanligt förekommande problem.

För arkitektur: Alexander et al. 1977:

> ”Each pattern describes a problem which occurs over and over and over again in our environment, and then describes the core of a solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”

MALMÖ UNIVERSITET



A place to wait
(exempel från Alexander et al. (1977))

…in any office, or workshop, or public service, or station, or clinic, where people have to wait – I*nterchange (34)*, *Health Center (47)*, *Small Services without Red Tape (81)*, *Office Connections (82)*, it is essential to provide a special place for waiting, and doubly essential that this place not have the sordid, enclosed, time-slowed character of ordinary waiting rooms.

MALMÖ UNIVERSITET

Olika typer av mönster

Analys-mönster
Beskriver koncept som är viktiga för att modellera krav.

Design-mönster
Beskriver struktur och interaktion mellan mindre komponenter i koden.

Arkitekturiella mönster
Beskriver hur de större komponenterna i ett system är strukturerade i förhållande till varandra.

Anti-mönster
Hur man inte bör göra – lösningar som visat sig vara olämpliga på olika sätt.

Designmönster

Designmönster beskrivs strukturerat. Mallarna för dessa varierar lite beroende på varifrån de kommer. Generellt bör en beskrivning av ett mönster innehålla:

- Namn
- Problembeskrivning
- Kontext
- Krafter/Forces
- Lösning

Olika typer av designmönster

Creational patterns – skapa instansobjekt

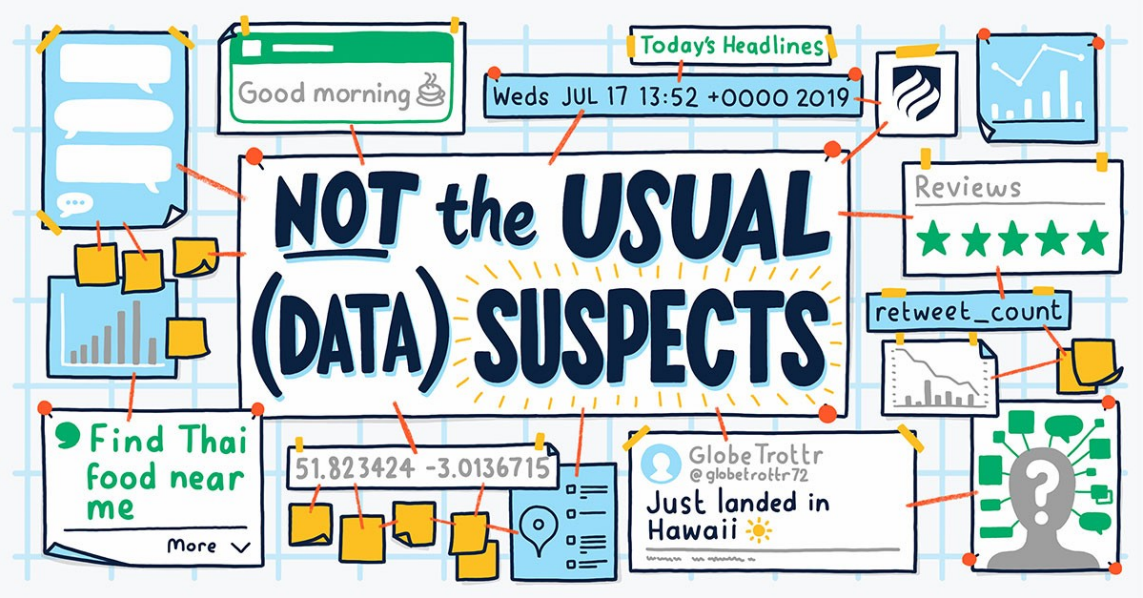
Structural patterns – design av klasser och relationer

Behavioural patterns – kommunikationsmönster mellan objekt

Ursprung i GoF:s *Software Patterns*

Datastrukturer

MALMÖ UNIVERSITET



Datastrukturer, vad är det?

Olika sätt att representera information

Dagens föreläsning avhandlar några vanliga datastrukturer.

MALMÖ UNIVERSITET

Datastrukturer

Information i ett modern datorsystem är i grunden representerat av en serie binära tal (1/0, sant/falskt, etc).

Vi behöver ordna och tolka dessa binära tal för att kunna representera olika former av data. Vi kallar dessa representationer för *datastrukturer*.

Först och främst! Python...

I Python är alla datatyper och datastrukturer egentligen objekt. För vårt vidkommande kan vi dock bortse från det.

MALMÖ UNIVERSITET

Objekt

Objekt är data i minnet som refereras av en pekare eller en *referens*. Detta kan vara enkla variabler, men även komplicerade datastrukturer eller funktioner.

I dagligt tal är objekt data som ordnas enligt ett givet mönster. Hos *klassobjekt* bestäms denna ordning av dess *klassdeklaration*.

Datastrukturer

Vi delar upp datastrukturer i *primitiva* och *abstrakta datatyper*.

Primitiva datatyper är datatyper som (generellt sett) tar upp **en** minnesplats.

Abstrakta datatyper tar upp **två eller fler** minnesplatser.

Atomärer

MALMÖ UNIVERSITET

Atomära datastrukturer

Med annat namn: **primitiver**

Enkla datarepresentationer som ryms i en minnescell (vanligtvis 4 eller 8 bytes). Kan i en del fall ta upp flera minnesceller, beroende på vilket språk som används.

Alla primitiver är *egentligen* samma sak: binära talsekvenser. Det enda som skiljer dem åt är hur vi väljer att tolka och använda dem.

Heltal – integers

Heltal (eng. *Integer*) är naturliga tal (exempelvis 42, 0, -13). En helt grundläggande datatyp som kan användas för att representera de flesta andra datatyper.

I Python: *int*

```
>>> type(43)
<class 'int'>
```

MALMÖ UNIVERSITET

Booleska värden – booleans

En datatyp som kan anta två värden: sant eller falskt. Representeras i många språk av 0 för sant, -1 för falskt. I Python är de uppräknade värden (se nästa bild).

I Python: *bool*

```
>>> type(True)
<class 'bool'>
```

MALMÖ UNIVERSITET

Uppräkningstyper – enums

Uppräkningstyper är heltal kopplade till en uppsättning namn. Detta kan användas till att representera exempelvis en färgpalett eller liknande.

Flyttal – float

Flyttal (eng. *Float*) är decimaltal. De representeras som produkten av två heltal: *signifikanden* (eller *mantissan*) och *exponenten*.

I Python: *float*

```
>>> type(3.14)
<class 'float'>
```

MALMÖ UNIVERSITET

Tecken – characters

Tecken (eng. *Character* eller *Char*) är ett binärt heltal som representerar ett tecken i en *teckenuppsättning*. Den nu vanligaste teckenuppsättningen är *Unicode*.

I Python:

```
>>> chr(2354)
'ल'
```

MALMÖ UNIVERSITET

Referenser

Referenser är platser i minnet som pekar på objekt. Dessa kan ses som “namn” på objekt. Därför kan flera referenser peka på *samma objekt*.

ac8647
(en referens)

Gamling
(en referens)

Johan
(en referens)

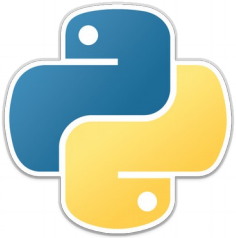
Objektet

MALMÖ UNIVERSITET

Referenser, forts

Referenser kan *pekas om* till nya objekt. Detta gör att samma referens kan referera till olika objekt i olika stadier av dess livslängd.

```
>>> a = [0]
>>> b = a
>>> print(b)
[0]
>>> b.append(2)
>>> print(a)
[0, 2]
>>> a = []
>>> print(b)
[0, 2]
```



MALMÖ UNIVERSITET

Abstrakta datatyper

MALMÖ UNIVERSITET

Abstrakta datatyper

Mer avancerade datarepresentationer som tar upp flera minnesceller. Dessa kallas allmänt för *datastrukturer*.

Alla typer av abstrakt sammansatta data är *egentligen* samma sak: sekventiellt ordnade primitiver. Det enda som skiljer dem åt är hur vi väljer att tolka och använda dem.

Vektorer (arrays)

En vektor (eng. *Array*) är en ordnad, sekventiellt lagrad lista av objekt. Varje värde i vektorn är indexerbart.

Tvådimensionella vektorer kallas ofta för *matriser*.

I Python saknas egentligen vektorer, men motsvaras av listor:

```
>>> a = [1,2,4,8]
>>> a[2]
4
```

MALMÖ UNIVERSITET

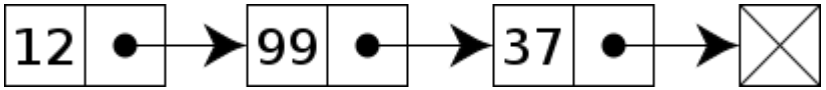
Strängar

Ett specialfall av vektorn är strängarna (eng. *String*). Dessa är vektorer av tecken. Således har strängar samma egenskaper vektorer, vilket innebär att varje tecken i strängen är identifierbar med hjälp av ett index.

I Python och många andra språk är *String* en klass med en massa trevliga metoder som vi kan använda för att manipulera strängen.

Listor

Listor är precis som vektorer sekventiellt ordnade objekt, men med skillnaden att dessa objekt har en relation till sina grannar. Den enklaste listtypen är den *länkade listan*, där en *nod* består av en primitiv följd av en referens till nästa nod i listan.



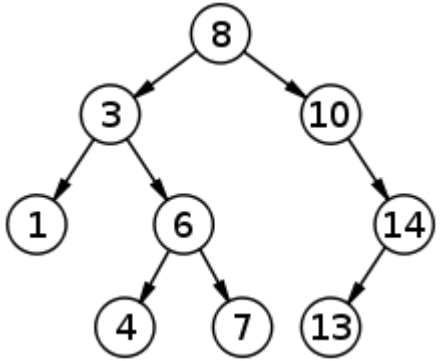
MALMÖ UNIVERSITET

Grafer

Grafer är, precis som listor, datastrukturer där de ingående objekten, *noderna*, har relationer till varandra. Relationerna, som anger grannar (och oftast har en riktning) kallas för *bågar* (eng. *edges*). Ett exempel på grafer är de tillståndsdiagram som vi arbetat med i kursen.

Grafer är bra för att representera relationer mellan objekt.

Träd



Träd är, precis som grafer, datastrukturer där de ingående objekten, *noderna*, har relationer till varandra. Till skillnad från graferna har bågarna i ett träd alltid enbart en riktning, utgående från rotnoden. Varje gren i trädet är i sig ett träd.

Träd kan, om de är sorterade, göra sökningar väldigt effektiva. Dessa trän kallas då för *sökträd*. Exempel: https://www.youtube.com/watch?v=5TQIOXHGKuM

MALMÖ UNIVERSITET

Associativa vektorer

En associativ vektor (eng. *Associative array* eller *Map*) är en samling objekt, benämnda *värden*, där varje värde identifieras med en *nyckel*.

I Python motsvaras detta av *dictionary*. Exempel:

```
>>> cat = {
... "name": "Zorro",
... "age": 7
... }
>>> print(cat["name"])
Zorro
```

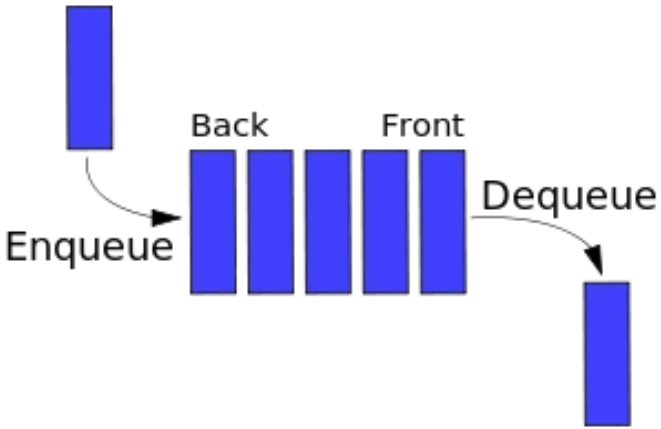
MALMÖ UNIVERSITET

Hashmaps

Ett exempel på en associativ vektor är *hashmap*, där varje värde representeras av ett heltal, ungefär som i en vektor. Detta heltal är en *hash*, vilket innebär att det räknas ut baserat på egenskaperna hos det objekt det representerar. Den underliggande datastrukturen är ofta ett sökträd.

Köer

En kö (eng. *Queue*) är en sekventiellt ordnad datastruktur, i vilken insättningar och uttag sker i ordningen *först in, först ut*.



MALMÖ UNIVERSITET

Stackar

En stack (eng. *Stack*) är precis som kön en sekventiellt ordnad datastruktur, men där insättningar och uttag sker enligt principen sist in, först ut.

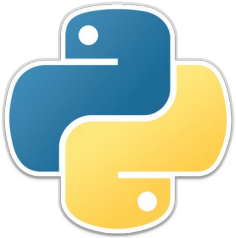
I Python finns denna funktionalitet inbakad direkt i listorna:

```
>>> a = [1,2,4,8]
>>> a.append(16)

>>> print(a)
[1, 2, 4, 8, 16]

>>> a.pop()
16

>>> print(a)
[1, 2, 4, 8]
```



MALMÖ UNIVERSITET