Introduction Til Programmering
Disposition 5
Inheritance and Dynamic Method Lookup

Joakim Iversen

12-11-2024

Disposition:

- Motivation
- Inheritance
- subtypes, substitution og polymorfisk variabler
- DML (Dynamic Method Lookup)

Kode eksempel: NewsFeed
DU BEHØVER IKKE SKRIVE ALLE FELT VARIABLER OG METODER IND I UML-DIAGRAM

Notes:

Motivation:

- Undgå kode duplikering
- Gør koden letter at rette
- Tilføje features uden at modifisere subklassen

Kapitel 10 - Improving Structure with Inheritance:

Main concepts:

- Inheritance
- Substitution
- Subtyping
- Polymorphic Variables

Notes:

Kode eksempel: Vil gøre brug af projektet **Network**.
Projektet har tre klasser:

1. NewsFeed
2. MessagePost
3. PhotoPost

Newsfeed indeholder en arrayliste til at holde på MessagePost og PhotoPost. Både MessagePost og PhotoPostindeholder nogle metoder der gør det samme - det kan derfor være i vores interresse at få lavet en "fælles" metode til nogle af disse.

TEGN KLASSE DIAGRAM UDEN INHERITANCE + KODE EKSEMPEL
Dårligt eksempel med MessagePost og PhotoPost - Mange ens metoder.

Inheritance:

Til at løse dette vil vi kunne gøre brug af nedarvning - Inheritance. Nedarvning tillader os at lave en klasse der kan "nedarve" dens metoder og felt variabler. Klassen der bliver nedarvet fra vil jeg kalde "fader-klassen", formelt kaldes den for "super-klassen" og klasser der nedarve ting fra den kaldes for "sub-klasser".

Måden man deklarer en sub- og super-klasse er ved at bruge det reserverede ord 'extends'.
Brug af nedarvning

MODIFICER KODE EKSEMPEL OG KLASSE DIAGRAM TIL BRUG AF NEDARVNINING Tilføj "extends", lav en post super-klasse

something **!HUSK!**

- KALD SUPER KLASSSEN MED

  `super(parametre)`

- Flyt fælles feltvariabler og metoder

Subtyping / Substitution:
Subtyping handler, skåret ind til benet, om hvilke typer der hænger sammen. Det kan samenlignes med når vi laver sub- og super-klasser.

Når vi bruger nedarvning vil vi nemlig kunne sige at vores subklasser også bliver en "sub type" af vores superklasse. Forstået på den måde, at alle steder vi vil skulle oplyse en type lig vores superklasse, vil vi kunne opgive en subtype (subklasse).

Lille subtype eksempel
Vi kan også tage et ekssempel med musik. Her genre en type og 80'er pop er klart den bedste subtype.

Overført til vores eksempel på tavlen kan vi sige at vores post er en type, og MessagePost og PhotoPost er subtyper af post. Dette vil sige at vi kan lave substitution, hvor vi sætter en variable defineret med typen Post lig med MessagePost eller PhotoPost.

LAV EKSEMPEL PÅ SUBSTITUTION MED POST OG MSGPOST

Vi kan egentlig sige at alle selvlavet klasser er sub-klasser og subtyper af hoved klassen "Objekt". Java laver automatisk denne klasse til en suertype til ens klasser, så det er ikke noget man selv skal tænke på. Det er også fra denne klasse vi har metoderne *toString(), hashCode(),* og *equals()*

Kapitel 11 - More about Inheritance:

Main concepts:

- Method polymorphism
- Static and dynamic type
- Overriding
- Dynamic method Lookup

Notes:

Vores kode eksempel fra sidst i kap 10 vil printe forkert. Vi vil ikke kunne se MessagePost message tekst eller PhotoPost image filename ikke er tilgængeligt i vores post klasse.

Dette skyldes at en inherince kun er en envejs kommunikation. Det er altså kun muligt for klasserne der nedarver at tilgå faderklassen, men ikke faderklassen at tilgå børnene.

Statisk- og dynamisk type:
Vores første løsning på dette problem er måske at flytte vores display metode tilbage til MessagePost og PhotoPost. Dette giver dog nogle fejl sm gør vi ikke kan compilie:

- MessagePost og PhotoPost kan ikke tilgå superklassens feltvariabler
- NewsFeed kan ikke finde display metoden mere.

Når vi snakker om dynamsike og statiske typer refferere vi til når en variable af en type egenligt er noget andet.

```
Post p1 = new MsgPost();
```

Er $p1$ her af typen Post eller MessagePost? Vi siger at den dynamiske type af $p1$ er en MessagePost og den statiske type er Post. Dette er da vi kan ændre hvad den peger på, men selve definationen af $p1$ kan ikke ændres og det er af typen Post.

Derfor får vi en fejl ved at flytte metoderne ned i MessagePost og PhotoPost, da compilieren kigger på de statiske typer, og ved at rykke metoderne er der ikke længere en dispaly metode i post klassen.

Derfor bliver vi nød til også at have en display metode i vores post klasse.

Overriding:
Hvis vi antager at display metoden nu er i alle tre klasser (Post, MessagePost, PhotoPost). Nu vil vi kunne kompilerer vores kode da Newsfeed nu kan finde display metoden i Post klassen.

Hvis vi så vil have MessagePost og PhotoPost til at printe noget forskelligt, kan vi Override dispaly metoden fra Post. Dette gør vi ved bare at lave en metode der hedder præcis det samme - her er det også god praktis at tilføje '@Override' over metoden for at sige hvad man gør.

Hvis vi vil have den til først at printe superklassens metode kan vi kalde den på følgende måde for PhotoPost:

```
public void disply(){
    super.display();
    System.out.println("  [" + filename + "]");
    System.out.println(" " + caption);
}
```

Dette vil først kalde metoden fra superklassen derefter kører metoden fra subklassen.

Dynamic Method Lookup:
Vi ændre hele måden vi tjekker efter en metode. En variables statiske typer har ikke noget at sige mere. Det første den kigger efter under gennemkørsel er den dynamiske type. Denne vil tage precedens over superklassens metoder.

Derfor vil den, hvis der er en, kalde dispaly metoden fra subklassen. Hvis der ikke er en der, vil den så søge opad indtil den finder en.

Det er det samme der sker med toString metoden fra objekt klassen. Hvis vi ikke overrider den i løbet af vores kode vil den gå til dens standard metode som skrevet i objekt klassen. Hvis den derimod er blevet overridet, vil den tage den overridet metode.

Method polymorphism: Vi vil her også kunne sige display metoden er polymorfisk ligesom Post typen, da den ændre sig alt efter hvilken subtype den kommer fra.

Der er altså forskel på følgende to metode kald:

```
MsgPost.display();
PhotoPost.display();
```