Videregående programmering i Java

Dag 5 - Model-View-Controller-arkitekturen

Model-View-Controller-arkitekturen (MVC)
Evt.: Rekursion

Læsning: VP 19

Rekursion (hvis interesse)

- Listning af filer i undermapper?

```
import java.io.*;
public class ListFilerRekursivt
{
	public static void main(String[] arg)
	{
		File kat = new File(".");   // det aktuelle katalog
		listKatalog(kat);
	}

	private static void listKatalog(File kat)
	{
		File[] filer = kat.listFiles();

		for (int i=0; i<filer.length; i++)
		{
			File f = filer[i];

			if (f.isDirectory())
			{
				System.out.println("Katalog "+f+":");
				listKatalog(f);
				System.out.println("Katalog "+f+" slut.");
			} else {
				System.out.println(f);
			}
		}
	}
}
```

Rekursion (hvis interesse)

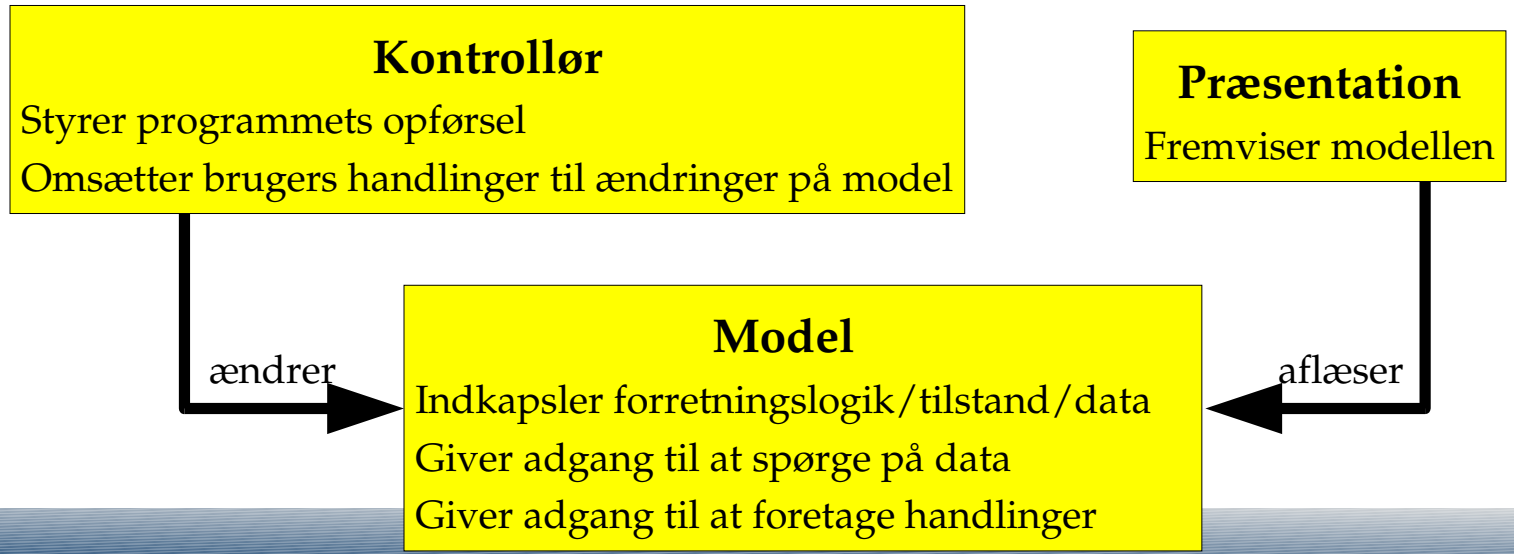
- Hvad skriver dette program ud?

```
public class Rekursionsopgave
{
  public static String kaldRekursivt(String s)
  {
    if (s.length()<=1) return s;

    String førsteTegn = s.substring(0,1);
    String resten     = s.substring(1);
    String resten2    = kaldRekursivt( resten );
    String detHele    = resten2 + førsteTegn;
    return detHele;
  }

  public static void main(String[] arg)
  {
    String resultat = kaldRekursivt("Hej verden!");
    System.out.println(resultat);
  }
}
```

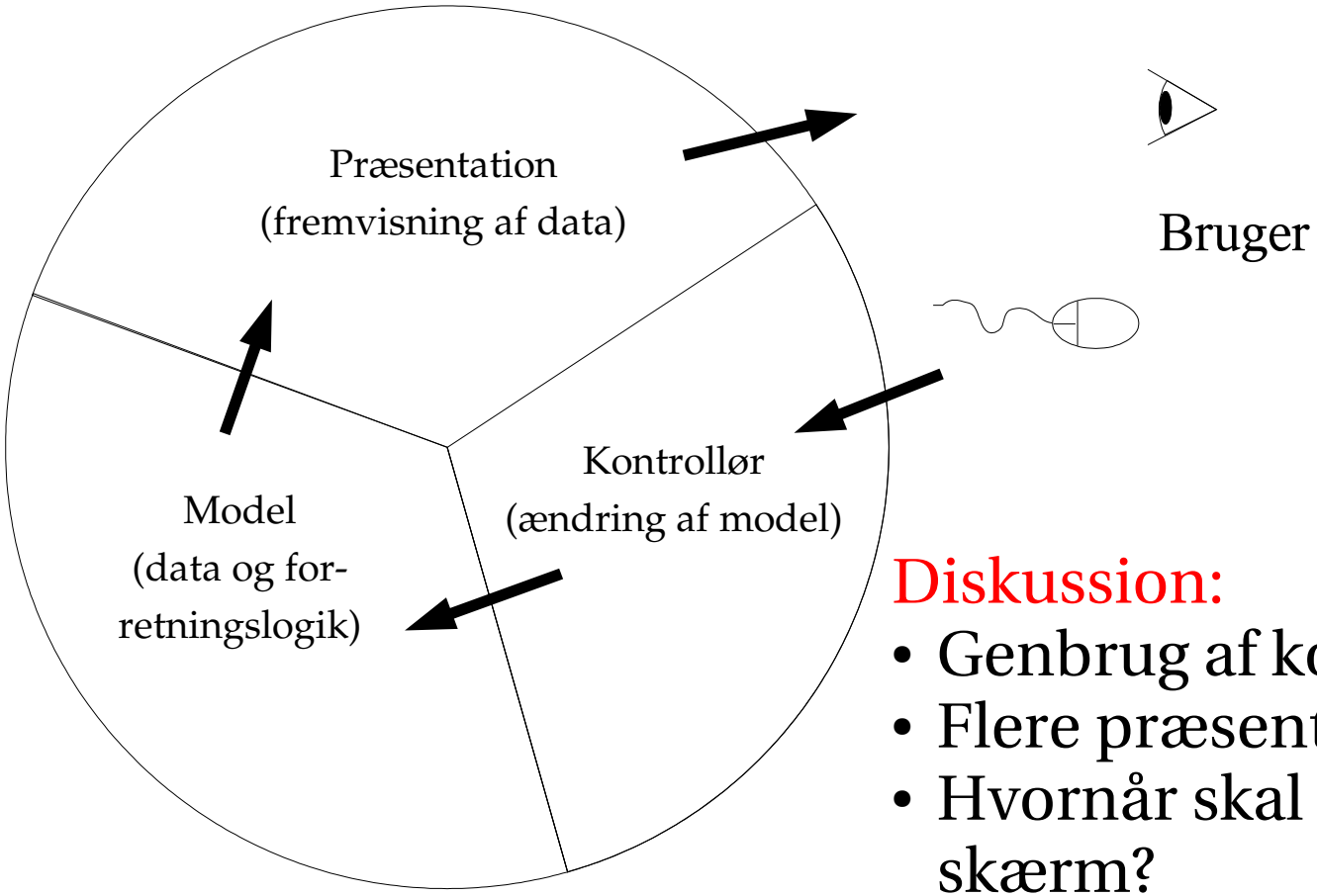
Model-View-Controller

Programmer med en brugergrænseflade kan inddeles i tre dele:

1. Forretningslogik: Data og de bagvedliggende beregninger (model)
   - En bankkonto har navn på ejer, kontonummer, kort-ID, saldo, bevægelser, etc.
   - Saldo kan ikke ændres direkte, men med handlingerne overførsel, udbetaling og indbetaling
2. Visning af informationer til brugeren (præsentation)
   - Bankkonti præsenteres meget forskelligt.
   - I en pengeautomat vises ingen personlige oplysninger overhovedet
   - I et netbank-system kan saldo og bevægelser ses (det kunne være en webløsning i HTML)
3. Mulighederne for at ændre i data gennem handlinger (kontrollør)
   - I en pengeautomat kan man kun hæve penge
   - I et netbank-system kan brugeren måske lave visse former for overførsel fra sin egen konto
   - Ved skranken kan medarbejderen derudover foretage ind- og udbetalinger



Model-View-Controller



Diskussion:

- Genbrug af kode?
- Flere præsentationer af samme data?
- Hvornår skal præsentation opdatere skærm?

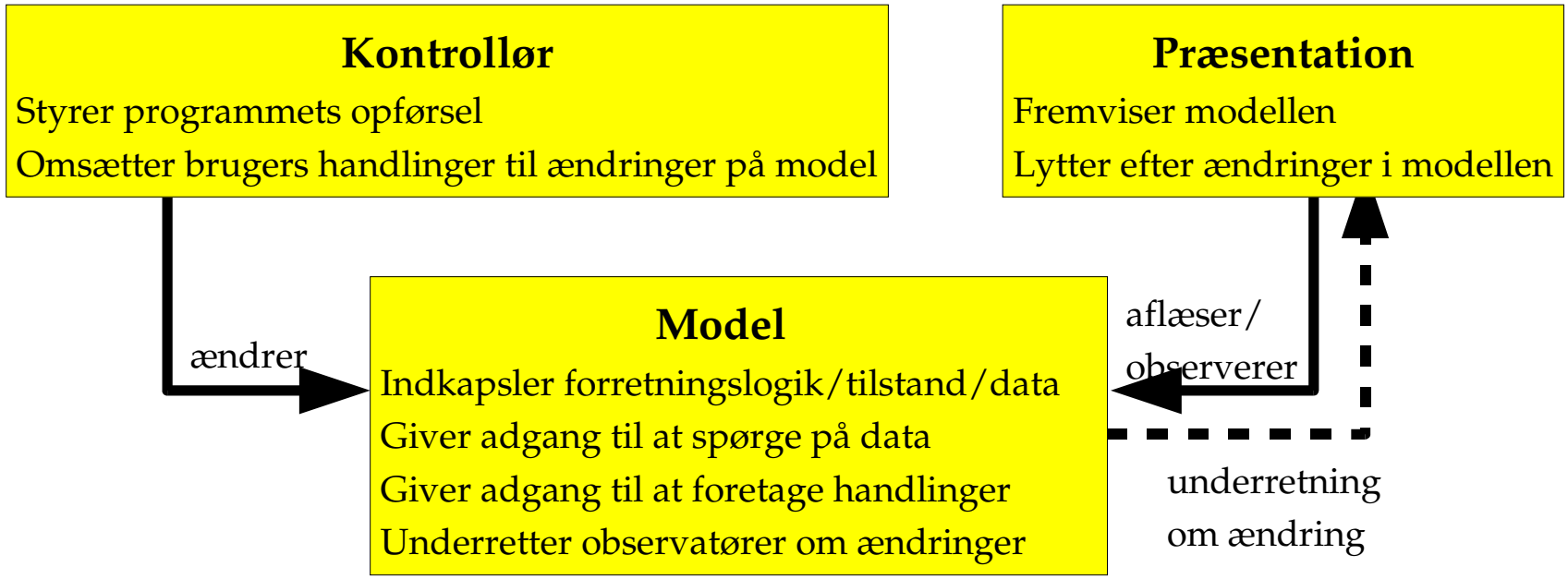
Muligheder for opdatering af præsentation:

A - Præsentationer undersøger modellen

B - Kontroldel underretter præsentationer

C - Modellen underretter præsentationer

C - Model underretter præsentation



C - Model underretter præsentation

Kontrollør
Styrer programmets opførsel
Omsætter brugers handlinger til ændringer på model

Præsentation
Fremviser modellen
Lytter efter ændringer i modellen

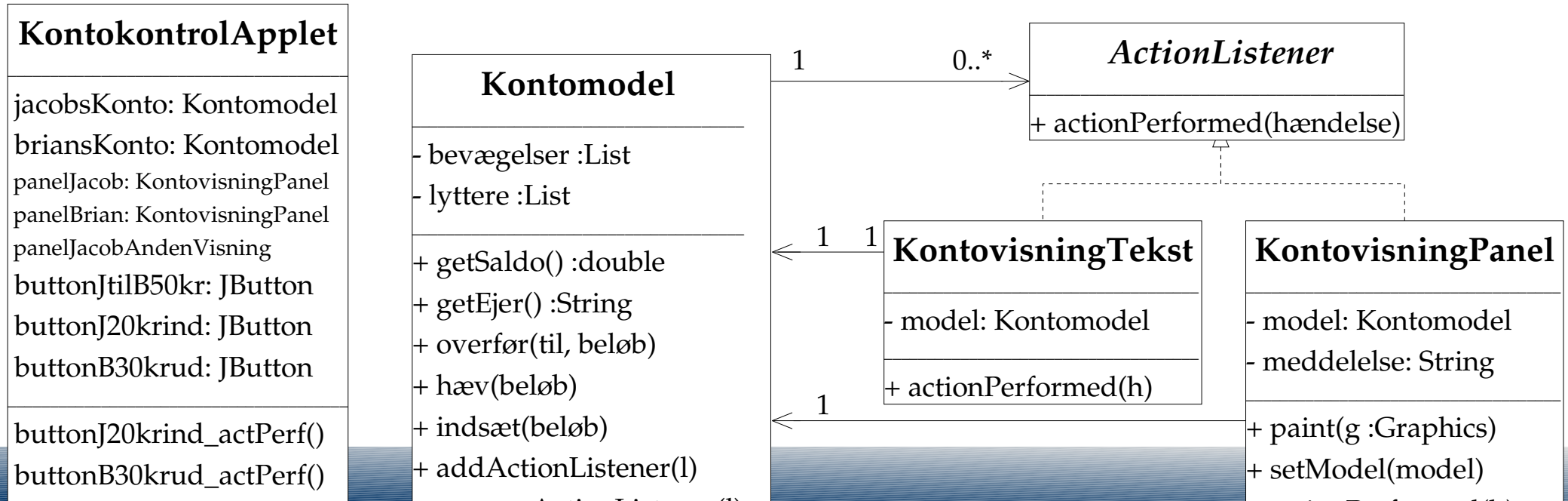
ændrer

Model
Indkapsler forretningslogik/tilstand/data
Giver adgang til at spørge på data
Giver adgang til at foretage handlinger
Underretter observatører om ændringer

aflæser/
observerer

underretning
om ændring

Eksempel:



Skabende designmønstre - igen

```
class BenytHjælp {
 private Hjælp h;

 public BenytHjælp() {
   // høj kobling - klient opretter et Hjælp-objekt
   //Hjælp h = new Hjælp();

   // fabrikeringsmetode leverer objekt til klienten
   h = Hjælp.opretHjælp();
 }

 public void gørNoget() {
   h.metode1();
   h.metode2();
 }
}
```

Skabende designmønstre - igen

```
class BenytHjælp {
 private Hjælp h;

 public BenytHjælp() {
   // høj kobling - klient opretter et Hjælp-objekt
   //Hjælp h = new Hjælp();

   // fabrikeringsmetode leverer objekt til klienten
   h = Hjælp.opretHjælp();
 }

 public void gørNoget() {
   h.metode1();
   h.metode2();
 }
}
```

Det kan være at:

- Det heller ikke er hensigtsmæssigt at biblioteksklassen ved hvordan Hjælp oprettes
- Hvad der skal oprettes afhænger af konfigurationsfil

Hvad hvis oprettelse/konfigurering skal være endnu mere dynamisk?

- Fabrikeringsmetode giver høj kobling dér hvor objektet i sidste ende oprettes

Løsninger

- Bind alt sammen i en sammenbygger-metode (eller i main()!)
- Slå (fabrikerings)objekt op i en registreringsdatabase (Service Locator)
- Overlad sammenbygningen til et framework (f.eks. Spring)
  (Inversion of Control = IoC, bedre ord er Dependency Injection)

Designmønstret Service Locator

```
class BenytHjælp {
 private Hjælp h;

 public BenytHjælp() {
   // centralt register holder styr på fabrikker
   hf = (HjælpFabrik) ServiceLocator.lookup("hjælp");
   h = hf.opretHjælp();
 }

 public void gørNoget() {
   h.metode1();
   h.metode2();
 }
}
```

HjælpFabrik er et interface.
Den rigtige fabrik-klasse er f.eks. lagret i en konfigurationsfil (fabrikker.properties) og instantieres med f.eks. dynamisk binding:

```
hjælp=MinEgenHjælpFabrik
```

- Slå (fabrikerings)objekt op i en registreringsdatabase
  - (endnu) mindre gennemskuelig kode
  - Programmet bliver afhængig af (bundet til) ServiceLocator
  - Rode med konfigurationfiler

Designmønstret Service Locator

```
class BenytHjælp {
 private Hjælp h;

 public BenytHjælp() {
   // centralt register holder styr på fabrikker
   hf = (HjælpFabrik) ServiceLocator.lookup("hjælp");
   h = hf.opretHjælp();
 }

 public void gørNoget() {
   h.metode1();
   h.metode2();
 }
}
```

HjælpFabrik er et interface.
Den rigtige fabrik-klasse er f.eks. lagret i en konfigurationsfil (fabrikker.properties) og instantieres med f.eks. dynamisk binding:

```
hjælp=MinEgenHjælpFabrik
```

```
class ServiceLocator {
 private static Properties fabrikklasser = new Properties();
 static {
   fabrikklasser.load(new FileInputStream("fabrikker.properties"));
 }

 public Object lookup(String navn) {
   String fabrikNavn = fabrikklasser.getProperty(navn);
   Object fabrik = Class.forName( fabrikNavn ).newInstance();
   return fabrik;
 }
}
```

Dependency Injection / Injektion af afhængigheder

```
class BenytHjælp {
 private Hjælp h;

 public setHjælp(Hjælp h) { this.h = h; } // framework kalder (injektion)

 public BenytHjælp() { }     // gør intet, h bliver initialiseret udefra!

 public void gørNoget() {
   h.metode1();
   h.metode2();
 }
}
```

- Overlad sammenbygningen til et framework (f.eks. Spring)
- Inversion of Control = IoC
  - Frameworket kontrollerer sammenbygningen, ikke programmet
  - Bedre ord er Dependency Injection
    - Injektion (oprettelse) v.hj.a.:
      - Konstruktører
      - get- og set-metoder
      - Implementeringer af interfacet
  - Programmet forbliver uafhængigt og ubundet
    - ... og ubrugeligt uden frameworket (?)