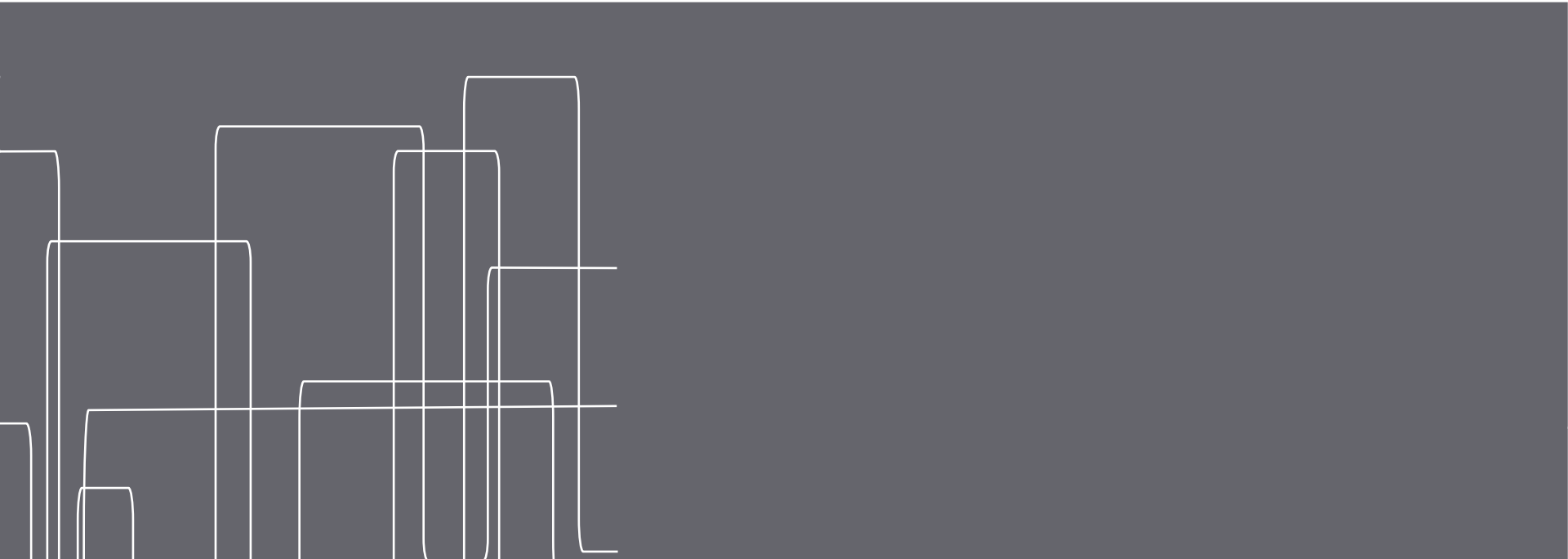
Objektorienterad Programkonstruktion

Föreläsning 13

17 jan 2017



Tentamen

- Del I, E – del
  - Flervalsfrågor
  - 20/25 krävs för godkänt, ger betyg E
  - Upp till 7 möjliga bonuspoäng
- Del II, Högrebetygsdel
  - 5 Problemfrågor, betygsätts med C eller A
    - 4 A + 1 C (eller bättre) ger A
    - 2 A + 3 C ger B
    - 4 C ger C
    - 2 C ger D
- Komplettering
  - 18-19p på E-delen ger FX, kan kompletteras till E
  - FX kan även kompletteras till D, C, B eller A om tillräckliga resultat finns på del II.

Exempelfrågor Del I

4. För varje påstående om Exceptions i Java, ange om det är sant eller falskt. 4 korrekta svar ger 2p, 3 korrekta svar ger 1p, 2 eller färre korrekta svar ger 0 p (2 p)

a) Kod som anropar en metod som kan kasta en eller flera Exception måste alltid inneslutas i ett try/catch-block.

b) En Exception är ett objekt som kan innehålla information om varifrån det kastades och varför.

c) Kod i ett finally-block exekveras bara om inga Exceptions har fångats.

d) Alla typer av Exceptions som man kan kasta finns redan definierade i Javas standard-API.

Exempelfrågor Del I

Punkterna a) till h) nedan beskriver olika designmönster. För varje beskrivning, ange namnet på det mönster som bäst passar beskrivningen. Välj namn från listan nedan. Notera att det finns fler namn än beskrivningar - alltså behöver inte alla namn användas. Varje rätt svar ger 1 poäng; totalt kan denna uppgift alltså ge 8 poäng.

Singleton MVC Composite Proxy Adapter Flyweight

Lock Observer Factory Builder Prototype Facade

a) Man skapar nya objekt O genom att skapa kopior av ett objekt A som redan existerar. (1 p)

b) Man skapar nya objekt O med anrop av en metod B() som beslutar vilken typ av objekt som skall returneras. (1 p)

c) Man skapar nya objekt O genom att först skapa ett objekt A. Objektet A kan man sedan modifiera och bygga på tills man är nöjd. Till sist låter man A returnera objektet O. (1 p)

…. osv ….

Exempelfrågor Del I

Kodsnuttarna a) till h) nedan beskriver olika designmönster. För varje beskrivning, ange namnet på det mönster som bäst passar beskrivningen. Välj namn från listan nedan. Notera att det finns fler namn än beskrivningar - alltså behöver inte alla namn användas. Varje rätt svar ger 1 poäng; totalt kan denna uppgift alltså ge 8 poäng.

Singleton MVC Composite Proxy Adapter Flyweight

Lock Observer Factory Builder Prototype Facade

```
public class D{
    public static GenericType getObject(int a){
        if(decisionFunction(a)){
            return new SpecificType1();
        }else{
            return new SpecificType2();
        }
    }
}
```

Exempelfrågor Del I

- För objekt av typen B, ange vilka av följande fält- och metodanrop som är tillåtna. Antag att alla fält är initialiserade. 0.5 p för varje korrekt svar. (4 p)

  a) myC.myFlt = 0.4;

  b) myCint.doStuff(2);

  c) myCint.doStuff();

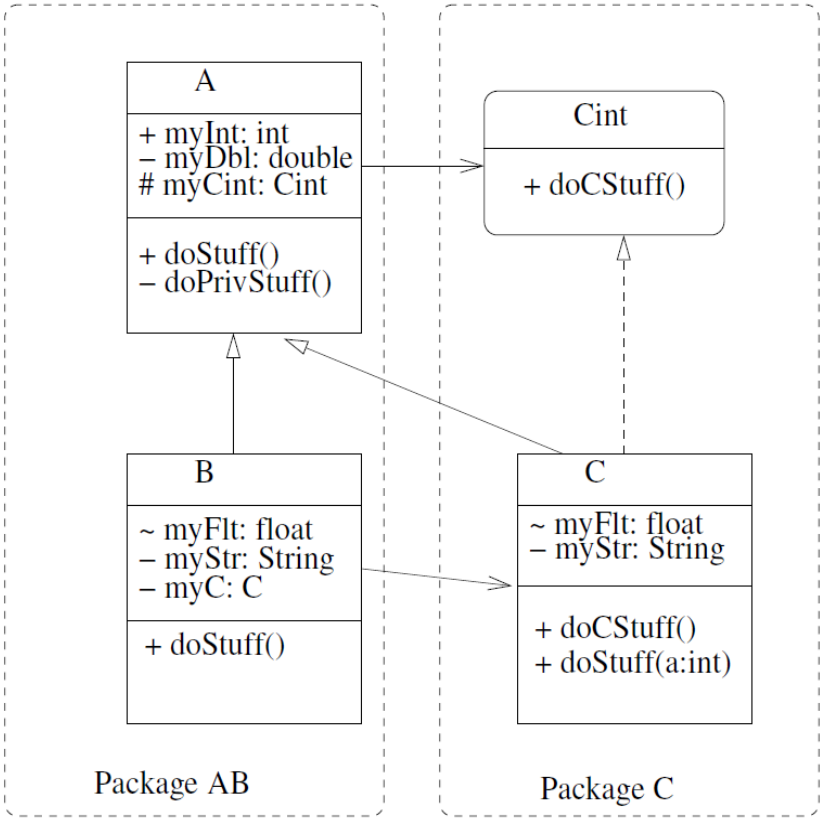
  d) myDbl = 2.1;

  e) myC.doCStuff();

  f) super.doPrivStuff();

  g) myC.doStuff((int)myC.myDbl);

  h) super.myInt = (int)myFlt;



Exempelfrågor Del I

- Nedan följer 5 kodexempel i Java. För varje kodexempel, ange om koden går att kompilera. Eventuella fel kan förväntas finnas inom den listade koden, dvs du kan anta att alla import-påståenden är korrekta, och att alla anrop av externa metoder fungerar som man förväntar sig. Varje korrekt analyserad programkod ger 1p. (5 p)

```
public class B6{
    private int a = 1;
    public static void main(String[] args){
        System.out.println(a);
    }
}
```

Exempelfrågor Del II

9. Förklara hur man kan använda Observer i MVC (2 p)

10. Förklara skillnaden mellan ett vanligt Observer-mönster och en publisher/subscriber. Vilka fördelar har det senare? (3p)

11. XML kan ibland vara ett bra format för att lagra data, men inte alltid. Förklara när det är bra, och när det passar dåligt, och ge exempel för båda fallen (2 p)

7. a) Förklara begreppet polymorfism, och på ge exempel på hur polymorfism kan underlätta för programmerare. (2 p)

b) Hur hanterar Java polymorfism? Vilka är tillåtna typer för att deklarera resp. instansiera variabler med? Vilka metodanrop är tillåtna? Hur/när avgörs vilken metod som ska anropas? (3 p)

Exempelfrågor Del II

- Antag att du har fått i uppgift att skriva ett program för att simulera partikelflöden i rörsystem. I programmet ska man kunna rita upp sina rörsystem, bestående av sammansättningar av färdiga rördelar som användaren kan välja från ett bibliotek. Man ska kunna modifera de färdiga rördelarnas sammansättning och egenskaper, och kunna spara både rördelar, delsystem och kompletta rörsystem till fil för att återanvända senare.

  Du har hittat en uppsättning förenklade differensekvationer för beräkning av partiklarnas rörelser i diskret tid, som du tänker använda tills din kollega har härlett något bättre. I din förenklade modell kan varje partikel uppdateras oberoende av de andra i varje steg, men man måste ha med väldigt många partiklar för att få realistiska resultat. Efter att en simulering har körts ska man dels kunna spela upp en animering av flödet, och dels kunna visualisera olika statistiska mått, som t.ex lokala partikeltätheter.

  Beskriv en bra struktur för programmet. Förklara vilka designmönster som används, i grova drag hur de implementeras (vilken information finns var, hur kommunicerar programmets olika delar med varandra, hur styrs olika programflöden, osv). Motivera varför din lösning är bra! För full poäng skall hänsysn ha tagits till alla delar som beskrivs ovan. Du får använda UML eller (pseudo-)kod i ditt svar om du tycker att det förenklar presentationen, men det är inte ett krav. (10 p)

Kloning

- Att skapa en exakt kopia av ett objekt. En kopia kommer att ha alla fält satta till samma värden som originalet
- Om **A** är en kopia av **B**, gäller oftast att:

```
(A == B) = false

A.equals(B) = true
```

- I Java finns metoden `clone()` definierad i klassen `Object`, och är alltså tillgänglig i alla klasser

  men

- I Object är metoden definierad så att den kastar en `CloneNotSupportedException` om inte den aktuella klassen implementerar gränssnittet `Cloneable`, och definierar en egen publik `clone()`-metod

Kloning

- Alla klasser anropar `super.clone()` i sin egen `clone()`-metod, så att `clone()` i `Object` anropas i slutändan, ex:

```
public Object clone() throws … {
    return super.clone();
}
```

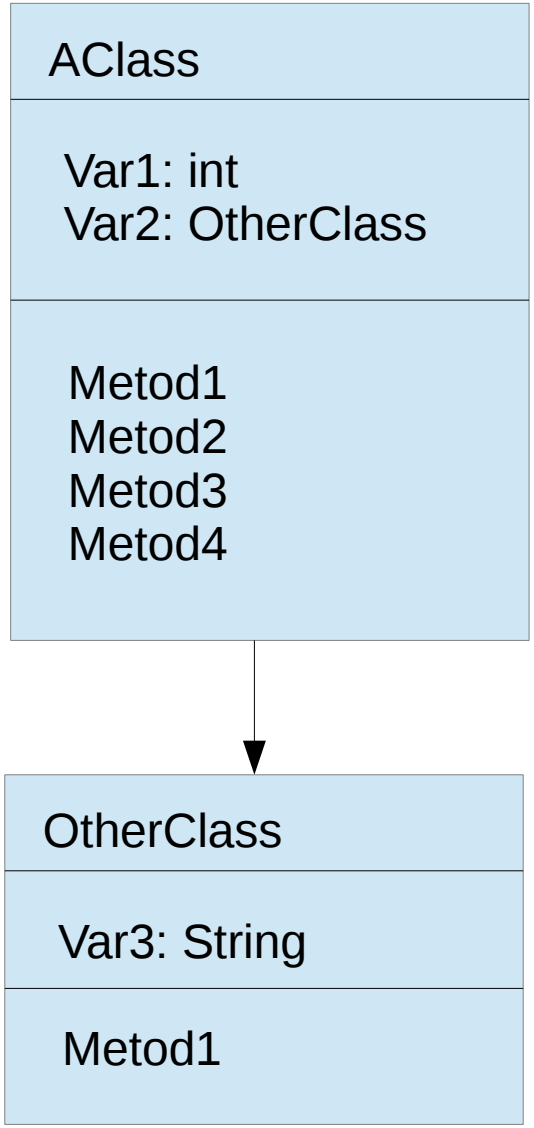
- Detta kommer att returnera en exakt kopia av det aktuella objektet
- Den kopia som skapas här är dock en s.k. **grund kopia** (eng: shallow copy)

Kloning

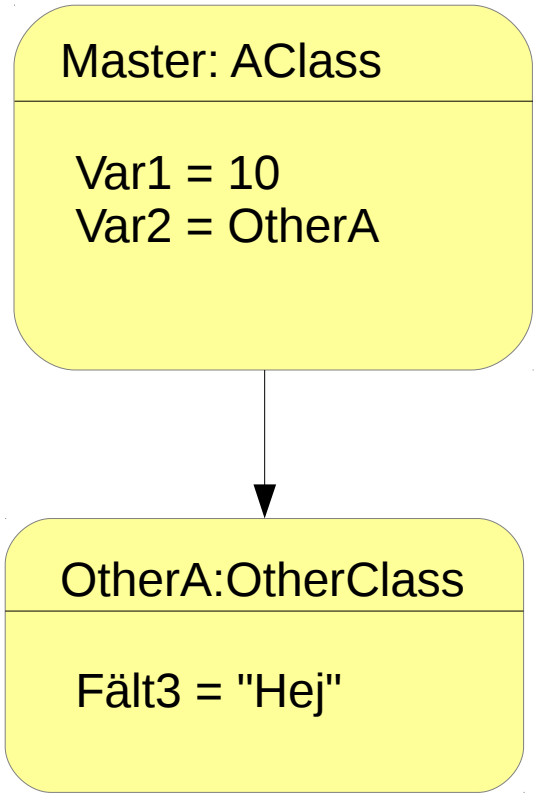
- En **grund kopia** kopierar alla fält exakt. Detta fungerar bra för primitiva datatyper

- En **djup kopia** (deep copy) gör dessutom kopior av objekt som refereras till
- Detta måste man implementera själv i sin clone()

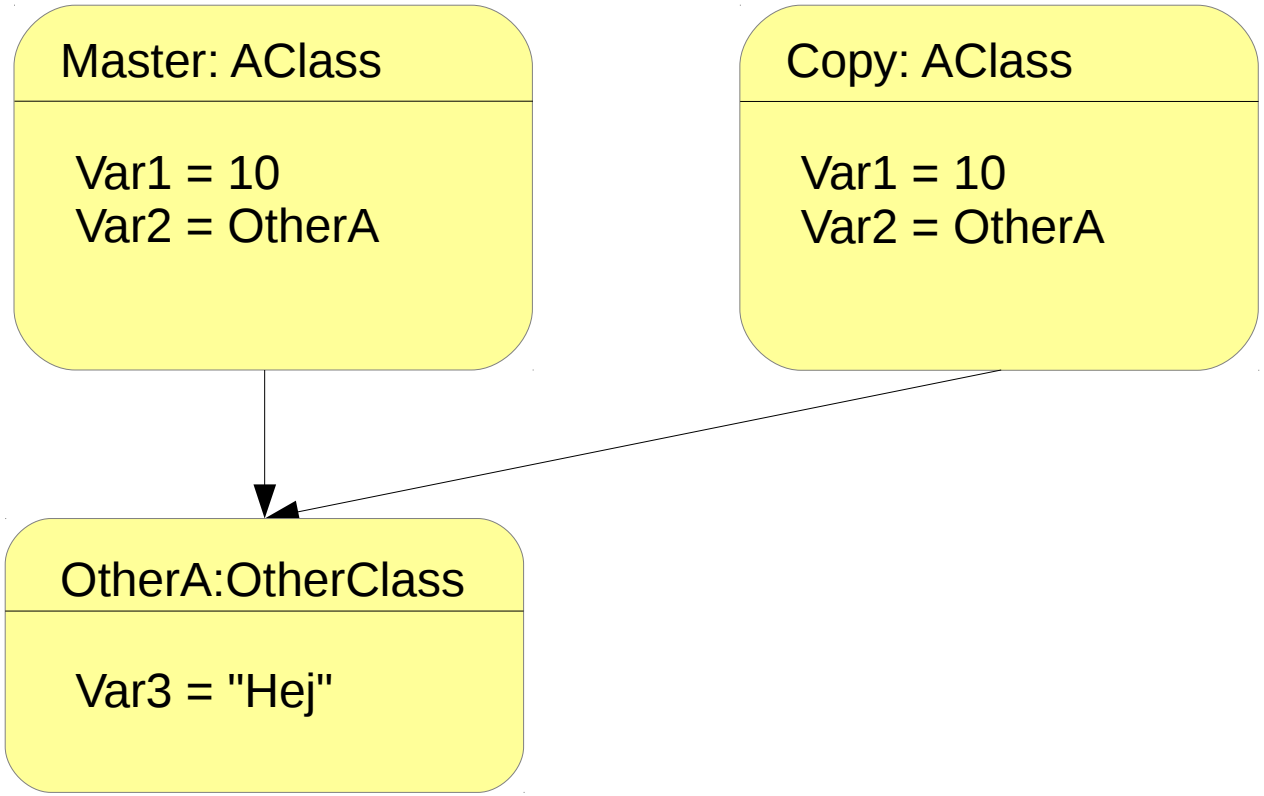
Kloning


AClass
Var1: int
Var2: OtherClass
Metod1
Metod2
Metod3
Metod4
OtherClass
Var3: String
Metod1

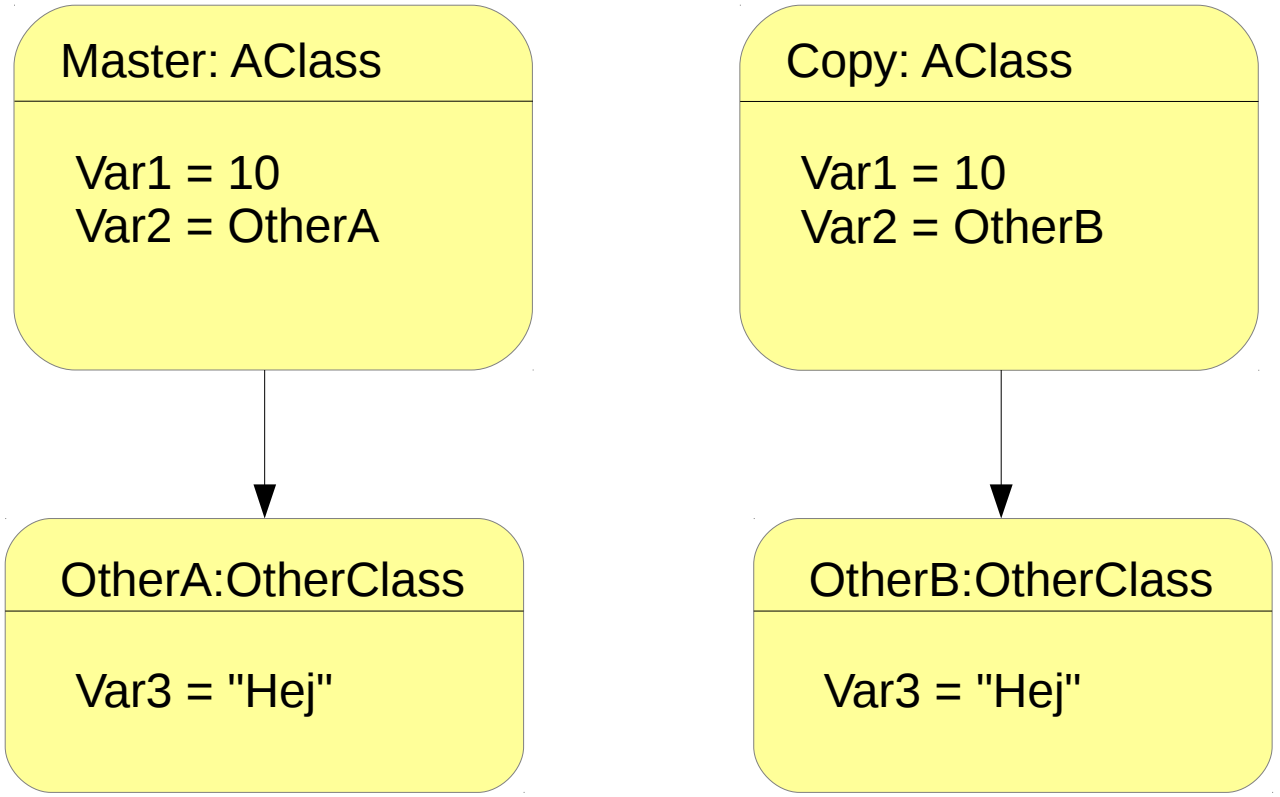
Grund kopia


Master: AClass
Var1 = 10
Var2 = OtherA
OtherA:OtherClass
Fält3 = "Hej"

Grund kopia


Master: AClass
Var1 = 10
Var2 = OtherA
Copy: AClass
Var1 = 10
Var2 = OtherA
OtherA:OtherClass
Var3 = "Hej"

Djup kopia


Master: AClass
Var1 = 10
Var2 = OtherA
Copy: AClass
Var1 = 10
Var2 = OtherB
OtherA:OtherClass
Var3 = "Hej"
OtherB:OtherClass
Var3 = "Hej"

Djupa kopior

- När man gör djupa kopior lägger man extra kod efter anropet till `super.clone()`

```
public Object clone() throws ...{
    MyObject copy = (MyObject) super.clone();
    copy.someField = someField.clone();
    return copy;
}
```

- **OBS:** Vissa fält bör kanske klonas, medan andra fortfarande bör vara referenser till samma objekt som i originalet. Det senare gäller t.ex för referenser till singletons.

Prototype (GoF)

- **Creational Pattern**
- Man börjar med att skapa ett objekt som får vara prototyp, och sedan klonar man detta när man vill skapa nya objekt.
- Kan användas för att förse fält med startvärden som kan ändras under programmets körning
- Om det är beräkningsmässigt tungt att räkna ut startvärdena, eller initiera de objekt som skall innehållas, kan man göra detta en gång för alla i prototypen, för att sedan kopiera resultaten till de nyskapade objekten
- Kan med fördel kombineras med mönstret **Factory**

Prototypexempel

- Skapa nya textobjekt (t.ex textrutor i ett grafikprogram, meddelande-delen i ett e-mail, eller inmatningsrutor till en ordbehandlare), som ska förses med värden för font, teckenkodning, stavningskontrollsspråk. Dessa värden skall kunna ändras medan programmet körs
- Skapa nya objekt i en spelmiljö utifrån en design som spelaren kan välja själv
- Genetisk programmering eller partikelfilter, där man vill skapa många objekt med de egenskaper som man just nu tror är de bästa

Flyweight (GoF)

- **Structural Pattern**
- Ger ett sätt att minska mängden resurser (minne) som behövs av varje instans av en klass
- Så mycket som möjligt av informationen i objektet lagras utanför objektet, på en plats som delas med andra objekt
- Exempel:
  - ett tecken i en ordbehandlartext innehåller ett fält som säger vilket tecken det ska vara, några fält med information om fontstorlek, huruvuda det är kursivt, fetstil eller dyl. Själva den grafiska informationen av hur tecknet skall se ut kan dock delas med alla andra objekt som representerar samma tecken.