

Resväska som *Composite* i Java

Lisa Dahl och Mostafa Shihadeh

November 7, 2025

Contents

1	Introduktion	1
1.1	Designval	1
1.2	Bygga och kompilera	1
2	Kod	3
2.1	Klassen Component	3
3	Konstruktör	3
4	Metoder	4
4.1	Leaf	4
4.1.1	Konstruktör	4
4.1.2	Metoder	5
4.2	Composite	5
4.2.1	Fält	5
4.2.2	Konstruktör	6
4.2.3	Barnhantering (add/remove/getChildren)	6
4.2.4	Totalvikt (getWeight)	6
4.2.5	toString	7
4.3	Client	7
4.3.1	Bygg upp strukturen (suitcase)	8
4.3.2	Skriv ut totalvikt och innehåll (före borttagning)	9
4.3.3	Ta bort några saker/behållare	9
4.3.4	Skriv ut totalvikt och innehåll (efter borttagning)	9
5	Kommentarer om korrekthet och mönstertrohet	9
6	Bygginstruktion	9

1 Introduktion

Vi vill implementera uppgift 1 (resväska enligt *Composite*-mönstret) med literär programmering. Vi bygger tre klasser: **Component** (abstrakt), **Leaf** (pryl) och **Composite** (behållare), samt ett testprogram **Client**.

1.1 Designval

- **Component** bär gemensamma attribut: **name** och **weight** (egen vikt).
- **Leaf** representerar en enskild pryl. **getWeight()** returnerar bara dess egen vikt.
- **Composite** representerar en behållare med barn. **getWeight()** summerar behållarens egen vikt och alla barns vikter. **toString()** traverserar rekursivt.
- Metoderna **add** och **remove** finns bara i **Composite** (inte i **Component**).

1.2 Bygga och kompilera

Vi vill skriva en byggfil för GNU Make. Själva, maskin-genererade reglerna läggs i **Suitcase.mk** som tanglas ur denna **.nw**-fil och sedan *inkluderas* från toppens **Makefile**.

Först lägger vi grundmålen och PDF-reglerna:

```
<Suitcase.mk>≡
TARGETS= Suitcase.pdf Suitcase.mk
all: classes Suitcase.pdf

Suitcase.pdf: Suitcase.tex
    pdflatex -interaction=nonstopmode -halt-on-error Suitcase.tex
    pdflatex -interaction=nonstopmode -halt-on-error Suitcase.tex

Suitcase.tex: Suitcase.nw
    noweave -latex Suitcase.nw > Suitcase.tex
```

Därefter tanglar vi ut varje `.java`-fil ur den litterära källan (samt bäddar in radmarkörer så att `noerr.pl` kan mappa fel till `.nw`):

$\langle \textit{Suitcase.mk} \rangle + \equiv$

```
Component.java: Suitcase.nw
    notangle -L'//line %L "%F"%N' -RComponent.java Suitcase.nw > Component.java
```

```
Leaf.java: Suitcase.nw
    notangle -L'//line %L "%F"%N' -RLeaf.java Suitcase.nw > Leaf.java
```

```
Composite.java: Suitcase.nw
    notangle -L'//line %L "%F"%N' -RComposite.java Suitcase.nw > Composite.java
```

```
Client.java: Suitcase.nw
    notangle -L'//line %L "%F"%N' -RClient.java Suitcase.nw > Client.java
```

Nu lägger vi sektionen för att kompilera och köra Java:

$\langle \textit{Suitcase.mk} \rangle + \equiv$

```
.PHONY: classes run clean-Suitcase
classes: Component.java Leaf.java Composite.java Client.java
    @if [ -x ./noerr.pl ]; then ./noerr.pl javac *.java; else javac *.java; fi
```

```
run: classes
    java Client
```

Slutligen städreglerna:

$\langle \textit{Suitcase.mk} \rangle + \equiv$

```
clean: clean-Suitcase
clean-Suitcase:
    rm -f Suitcase.tex Suitcase.aux Suitcase.log Suitcase.toc
    rm -f Suitcase.mk Suitcase.pdf
    rm -f *.class *.java
```

2 Kod

I det här avsnittet definierar vi de fyra Javafilerna som tänglas ut från denna .nw-fil.

2.1 Klassen Component

Ansvar: bas-klass med namn och egenvikt samt abstrakta metoder.

Filen `Component.java` innehåller definitionen av klassen `Component`. Översiktligt ser den ut så här:

```
<Component.java>≡
public abstract class Component {
    <Component attributes>
    <Component constructor>
    <Component methods>
}
```

3 Konstruktör

Vi skapar en konstruktör för klassen `Konstruktör` som initierar alla attribut.

Vi vill kunna skapa en behållare med ett namn och en vikt.

Vi vill att klassen `Component` ska ha följande attribut:

- **name:** namn på komponenten (sträng)
- **weight:** egen vikt i kg (flyttal)

Dessa attribut ska vara skyddade och slutgiltiga (`protected final`) för att förhindra ändring efter konstruktion.

```
<Component attributes>≡
protected final String name;
protected final double weight;
```

Vi lägger även till kontroller i konstruktorn för att säkerställa att namnet inte är null eller tomt, och att vikten är icke-negativ.

```
<Component constructor>≡
protected Component(String name, double weight) {
    if (name == null || name.isBlank()) throw new IllegalArgumentException("name");
    if (weight < 0) throw new IllegalArgumentException("weight");
    this.name = name;
    this.weight = weight;
}
```

4 Metoder

Vi definierar metoderna `getName`, `getOwnWeight` i klassen `Component`, samt de abstrakta metoderna `getWeight` och `toString`:

- `getName`: returnerar komponentens namn.
- `getOwnWeight`: returnerar komponentens egen vikt (utan barn).
- `getWeight`: abstrakt metod som ska returnera totalvikten (inklusive barn).
- `toString`: abstrakt metod som ska returnera en strängrepresentation av komponenten.

```
<Component methods>≡  
    public String getName() { return name; }  
    public double getOwnWeight() { return weight; }  
    public abstract double getWeight();  
    public abstract String toString();
```

4.1 Leaf

Ansvar: en enskild pryl att packa.

Filen `Leaf.java` innehåller definitionen av klassen `Leaf`. Översiktligt ser den ut så här:

```
<Leaf.java>≡  
    public class Leaf extends Component {  
        <Leaf constructor>  
        <Leaf methods>  
    }
```

4.1.1 Konstruktör

Konstruktorn tar namn och egen vikt och skickar vidare till basklassen `Component`.

```
<Leaf constructor>≡  
    public Leaf(String name, double weight) {  
        super(name, weight);  
    }
```

4.1.2 Metoder

Totalvikten för ett löv är samma som dess egen vikt. `toString` beskriver prylen.

```
<Leaf methods>≡
    @Override
    public double getWeight() {
        return this.weight;
    }

    @Override
    public String toString() {
        return this.name + " (" + this.weight + " kg)";
    }
```

4.2 Composite

Ansvar: en behållare som kan innehålla andra `Component`. Egen vikt *plus* alla barns vikter utgör totalvikten.

Filen `Composite.java` innehåller definitionen av klassen `Composite`. Översiktligt ser den ut så här:

```
<Composite.java>≡
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Composite extends Component {
    <Composite fields>
    <Composite constructor>
    <Composite child-management>
    <Composite weight>
    <Composite toString>
}
```

4.2.1 Fält

Vi lagrar barn i en muterbar lista, men exponerar en oföränderlig vy utåt.

```
<Composite fields>≡
    private final List<Component> children = new ArrayList<>();
```

4.2.2 Konstruktor

Konstruktorn tar behållarens namn och egen vikt.

```
<Composite constructor>≡  
    public Composite(String name, double ownWeight) {  
        super(name, ownWeight);  
    }
```

4.2.3 Barnhantering (add/remove/getChildren)

Vi kan lägga till och ta bort barn, och ge tillbaka en oföränderlig vy av listan.

```
<Composite child-management>≡  
    public void add(Component c) {  
        if (c == null) throw new IllegalArgumentException("child");  
        children.add(c);  
    }  
  
    public void remove(Component c) {  
        children.remove(c);  
    }  
  
    public List<Component> getChildren() {  
        return Collections.unmodifiableList(children);  
    }
```

4.2.4 Totalvikt (getWeight)

Totalvikt = egen vikt + summan av alla barns totalvikter (rekursivt).

```
<Composite weight>≡  
    @Override  
    public double getWeight() {  
        double sum = this.weight;  
        for (Component c : children) {  
            sum += c.getWeight();  
        }  
        return sum;  
    }
```

4.2.5 toString

Vi bygger en rekursiv beskrivning där barn listas inom hakparenteser.

```

<Composite toString>≡
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(this.name).append(" ").append(this.weight).append(" kg) [");
        for (int i = 0; i < children.size(); i++) {
            sb.append(children.get(i).toString());
            if (i < children.size() - 1) sb.append(", ");
        }
        sb.append("]");
        return sb.toString();
    }

```

4.3 Client

Ansvar: bygga en resväska med minst tre nivåer och minst tio prylar, skriva ut totalvikt och innehåll, ta bort några objekt och skriva ut igen.

Filen `Client.java` innehåller testprogrammet. Översiktligt ser det ut så här:

```

<Client.java>≡
    public class Client {
        public static void main(String[] args) {
            <Client build-structure>
            <Client print-before>
            <Client removals>
            <Client print-after>
        }
    }

```


4.3.1 Bygg upp strukturen (suitcase)

Vi skapar roten (resväskan), underbehållare och löv.

```
<Client build-structure>≡
// Roten: själva resväskan (egen vikt 2.3 kg)
Composite suitcase = new Composite("Resväska", 2.3);

// Större plagg (löven)
Leaf tshirt1 = new Leaf("T-shirt vit", 0.18);
Leaf tshirt2 = new Leaf("T-shirt svart", 0.19);
Leaf jeans   = new Leaf("Jeans", 0.75);
Leaf chinos  = new Leaf("Chinos", 0.55);
Leaf bok     = new Leaf("Pocketbok", 0.28);

// Necessär (behållare) med egen vikt 0.12 kg och innehåll
Composite necessar = new Composite("Necessär", 0.12);
Leaf tvål          = new Leaf("Tvål", 0.09);
Leaf schampo       = new Leaf("Schampo", 0.22);
Leaf borste        = new Leaf("Tandborste", 0.03);
Leaf tandkräm      = new Leaf("Tandkräm", 0.11);
necessar.add(tvål);
necessar.add(schampo);
necessar.add(borste);
necessar.add(tandkräm);

// Påse i necessären (tredje nivån)
Composite påse = new Composite("Påse", 0.01);
Leaf hårspännen = new Leaf("Hårspännen (10 st)", 0.02);
påse.add(hårspännen);
necessar.add(påse);

// Mindre väska för elektronik (behållare) | nivå två
Composite techbag = new Composite("Tech-väska", 0.20);
Leaf laddare      = new Leaf("Laddare", 0.15);
Leaf hörlurar     = new Leaf("Hörlurar", 0.08);
Leaf powerbank    = new Leaf("Powerbank", 0.18);
techbag.add(laddare);
techbag.add(hörlurar);
techbag.add(powerbank);

// Packa allt i resväskan
suitcase.add(tshirt1);
suitcase.add(tshirt2);
suitcase.add(jeans);
suitcase.add(chinos);
suitcase.add(bok);
```

```
suitcase.add(necessar);  
suitcase.add(techbag);
```

4.3.2 Skriv ut totalvikt och innehåll (före borttagning)

```
<Client print-before>≡  
System.out.printf("Totalvikt före borttagning: %.2f kg%n", suitcase.getWeight());  
System.out.println("Innehåll före borttagning:");  
System.out.println(suitcase.toString());
```

4.3.3 Ta bort några saker/behållare

Vi tar bort en pryl i en behållare, hela teknikväskan och påsen i necessären.

```
<Client removals>≡  
techbag.remove(powerbank);           // ta bort en pryl i behållare  
suitcase.remove(techbag);             // ta bort hela behållaren  
necessar.remove(påse);               // ta bort tredje nivåns behållare
```

4.3.4 Skriv ut totalvikt och innehåll (efter borttagning)

```
<Client print-after>≡  
System.out.printf("Totalvikt efter borttagning: %.2f kg%n", suitcase.getWeight());  
System.out.println("Innehåll efter borttagning:");  
System.out.println(suitcase.toString());
```

5 Kommentarer om korrekthet och mönstertrohet

- `getWeight()` anropat på ett `Composite`-objekt itererar rekursivt över alla barn (och deras barn), vilket följer *Composite*-mönstrets idé.
- `toString()` på `Composite` bygger en rekursiv presentation av hela trädet.
- `Leaf` saknar `add/remove` och är därmed ett *löv*.
- Egen vikt för behållare ingår alltid i totalvikten, som efterfrågats i labb-texten.

6 Bygginstruktion

Kör `make` för att generera `Suitcase.pdf` och `Suitcase.mk`. Kör `make classes` för att kompilera Javafilerna och `make run` för att köra demon av `Client`.