Resväska som *Composite* i Java

Lisa Dahl och Mostafa Shihadeh

November 7, 2025

Contents

1 Introduktion

Vi implementerar uppgift 1 (resväska enligt *Composite*-mönstret) med litterär programmering. Vi bygger tre klasser: `Component` (abstrakt), `Leaf` (pryl) och `Composite` (behållare), samt ett testprogram `Client`. I X4 lägger vi till två iteratorer (bredden-först och preorder) och gör `Composite` itererbar.

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`/`remove` finns bara i `Composite` (inte i `Component`).

2 Bygga och kompilera

Precis som i kursens exempel låter vi denna `.nw` tängla en maskingenererad `Suitcase.mk` som toppens `Makefile` *inkluderar*. Vi använder `noweave` för PDF och `notangle` för Java, med radmarkörer (`-L`) så `noerr.pl` kan mappa fel till `.nw`.

2.1 Grundmål och PDF-regler

Vi definierar `TARGETS`, ett `all`-mål, samt 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
```

2.2 Tangle: Java-källor med radmarkörer

Vi tanglar ut varje `.java`-fil ur den litterära källan och bäddar in radmarkörer för bättre felrapporter.

⟨*Suitcase.mk*⟩+≡

```
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
```

2.3 (X4) Tangle: iteratorerna

Vi tanglar även ut de två iteratorerna.

⟨*Suitcase.mk*⟩+≡

```
BreadthFirstIterator.java: Suitcase.nw
  notangle -L'//line %L "%F"%N' -RBreadthFirstIterator.java Suitcase.nw > BreadthFirstI

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

2.4 Kompilera och köra

Vi samlar Java-byggmålen. Om `noerr.pl` finns används det; annars kör vi `javac` direkt.

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

run: classes
  java Client
```

2.5 Städ

Vi tar bort genererade filer.

⟨*Suitcase.mk*⟩+≡
```
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
```

3 Kod

I detta avsnitt definierar vi Javafilerna som tänglas ut.

3.1 `Component` (abstrakt)

Ansvar: bas-klass med namn och egenvikt samt abstrakta metoder.
Filen `Component.java` ser översiktligt ut så här:

⟨*Component.java*⟩≡
```
public abstract class Component {
  ⟨Component attributes⟩
  ⟨Component constructor⟩
  ⟨Component methods⟩
}
```

3.1.1 Attribut

Vi vill att `Component` ska ha följande attribut:

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

Dessa är *protected final*.

⟨*Component attributes*⟩≡
```
protected final String name;
protected final double weight;
```

3.1.2 Konstruktor

Vi kontrollerar att namnet inte är 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;
}
```

3.1.3 Metoder

Vi exponerar `getName` och `getOwnWeight`. De abstrakta metoderna `getWeight` och `toString` implementeras i subklasser.

⟨*Component methods*⟩≡
```
public String getName() { return name; }
public double getOwnWeight() { return weight; }
public abstract double getWeight();
@Override public abstract String toString();
```

3.2 `Leaf`

Ansvar: en enskild pryl att packa.

Filen `Leaf.java` ser översiktligt ut så här:

⟨*Leaf.java*⟩≡
```
public class Leaf extends Component {
  ⟨Leaf constructor⟩
  ⟨Leaf methods⟩
}
```

3.2.1 Konstruktor

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

⟨*Leaf constructor*⟩≡
```
public Leaf(String name, double weight) {
  super(name, weight);
}
```

3.2.2 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)";
}
```

3.3 `Composite`

Ansvar: en behållare som kan innehålla andra `Component`. Egen vikt *plus* alla barns vikter utgör totalvikten. I X4 gör vi den även *itererbar* så att for-each fungerar (default: BFS).

Filen `Composite.java` ser översiktligt ut så här:

⟨*Composite.java*⟩≡
```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

public class Composite extends Component implements Iterable<Component> {
  ⟨Composite fields⟩
  ⟨Composite constructor⟩
  ⟨Composite child-management⟩
  ⟨Composite weight⟩
  ⟨Composite toString⟩
  ⟨Composite iterability⟩
}
```

3.3.1 Fält

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

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

3.3.2 Konstruktor

Konstruktorn tar behållarens namn och egen vikt.

⟨*Composite constructor*⟩≡
```
public Composite(String name, double ownWeight) {
  super(name, ownWeight);
}
```

3.3.3 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);
}
```

3.3.4 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;
}
```

3.3.5 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();
}
```

3.3.6 Itererbarhet (X4)

Vi gör `Composite Iterable<Component>`. Defaultiteratorn är BFS. Vi lägger även metoder för explicit BFS respektive preorder.

⟨*Composite iterability*⟩≡
```
@Override
public Iterator<Component> iterator() {
  return new BreadthFirstIterator(this);
}

public Iterator<Component> breadthFirstIterator() {
  return new BreadthFirstIterator(this);
}

public Iterator<Component> preorderIterator() {
  return new PreorderIterator(this);
}
```

3.4 `BreadthFirstIterator` (X4)

Ansvar: leverera noder i bredden-först-ordning (rot, alla barn, barnbarn, ...).
Filen `BreadthFirstIterator.java` ser översiktligt ut så här:

⟨*BreadthFirstIterator.java*⟩≡
```
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Iterator;
import java.util.NoSuchElementException;

public class BreadthFirstIterator implements Iterator<Component> {
  ⟨BFS fields⟩
  ⟨BFS ctor⟩
  ⟨BFS hasNext⟩
  ⟨BFS next⟩
  ⟨BFS remove⟩
}
```

3.4.1 Fält

Vi använder en kö (`ArrayDeque`) av `Component`.

⟨*BFS fields*⟩≡
```
  private final Deque<Component> queue = new ArrayDeque<>();
```

3.4.2 Konstruktor

Vi startar med roten.

⟨*BFS ctor*⟩≡
```
  public BreadthFirstIterator(Component root) {
    if (root == null) throw new IllegalArgumentException("root");
    queue.add(root);
  }
```

3.4.3 `hasNext`

⟨*BFS hasNext*⟩≡
```
  @Override
  public boolean hasNext() {
    return !queue.isEmpty();
  }
```

3.4.4 `next`

Om elementet är `Composite`, läggs dess barn sist i kön.

⟨*BFS next*⟩≡
```
@Override
public Component next() {
  if (queue.isEmpty()) throw new NoSuchElementException();
  Component current = queue.removeFirst();
  if (current instanceof Composite) {
    for (Component child : ((Composite) current).getChildren()) {
      queue.addLast(child);
    }
  }
  return current;
}
```

3.4.5 `remove`

Inte del av labbkraven.

⟨*BFS remove*⟩≡
```
@Override
public void remove() {
  throw new UnsupportedOperationException();
}
```

3.5 `PreorderIterator` (X4)

Ansvar: leverera noder i preorder (djupet-först): rot, sedan rekursivt barn vänster→höger.

Filen `PreorderIterator.java` ser översiktligt ut så här:

⟨*PreorderIterator.java*⟩≡
```
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Iterator;
import java.util.List;
import java.util.NoSuchElementException;

public class PreorderIterator implements Iterator<Component> {
  ⟨PRE fields⟩
  ⟨PRE ctor⟩
  ⟨PRE hasNext⟩
  ⟨PRE next⟩
  ⟨PRE remove⟩
}
```

3.5.1 Fält

Stack över `Component` som ska besökas.

⟨*PRE fields*⟩≡
```
private final Deque<Component> stack = new ArrayDeque<>();
```

3.5.2 Konstruktor

Börja med roten överst på stacken.

⟨*PRE ctor*⟩≡
```
public PreorderIterator(Component root) {
  if (root == null) throw new IllegalArgumentException("root");
  stack.push(root);
}
```

3.5.3 `hasNext`

⟨*PRE hasNext*⟩≡
```
@Override
public boolean hasNext() {
  return !stack.isEmpty();
}
```

3.5.4 `next`

Poppa ett element; om `Composite`, lägg dess barn på stacken i *omvänd* ordning.

⟨*PRE next*⟩≡
```
@Override
public Component next() {
  if (stack.isEmpty()) throw new NoSuchElementException();
  Component current = stack.pop();
  if (current instanceof Composite) {
    List<Component> children = ((Composite) current).getChildren();
    for (int i = children.size() - 1; i >= 0; i--) {
      stack.push(children.get(i));
    }
  }
  return current;
}
```

3.5.5 `remove`

Inte del av labbkraven.

⟨*PRE remove*⟩≡
```
@Override
public void remove() {
  throw new UnsupportedOperationException();
}
```

3.6 `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. I X4 demonstrerar vi även traversal i BFS och preorder.

Filen `Client.java` ser översiktligt 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⟩
    ⟨Client iterate-bfs⟩
    ⟨Client iterate-preorder⟩
  }
}
```

3.6.1 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);
```

3.6.2 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());
```

3.6.3 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
```

3.6.4 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());
```

3.6.5 (X4) Traversal: Bredden-först (for-each) och preorder

Vi skriver ut *endast nodernas namn* med `getName()` så att ordningen syns tydligt.

⟨*Client iterate-bfs*⟩≡

```
System.out.println("Bredden-först (default for-each):");
for (Component co : suitcase) {              // Composite.iterator() = BFS
  System.out.print(co.getName() + " ");
}
System.out.println();
```

⟨*Client iterate-preorder*⟩≡

```
System.out.println("Preorder (djupet-först):");
java.util.Iterator<Component> it = new PreorderIterator(suitcase);
while (it.hasNext()) {
  System.out.print(it.next().getName() + " ");
}
System.out.println();
```

4 Bygginstruktion

Kör `make` (eller `make classes`) för att kompilera Java. Kör `make run` för att köra demon. Kör `make Suitcase.pdf` för att generera PDF:en.