Human factory i Java (paket `human`)

Lisa Dahl och Mostafa Shihadeh

November 7, 2025

Contents

1 Introduktion

I denna del implementerar vi *Human factory*. Vi skriver en abstrakt `Human` och tre konkreta subklasser: `NonBinary`, `Woman` och `Man`. En statisk fabriksmetod i `Human` väljer subklass baserat på personnumrets näst sista tecken: `'0'` ⇒ `NonBinary`; udda siffra ⇒ `Man`; jämn (men ej `'0'`) ⇒ `Woman`.

Vi placerar `Human`, `NonBinary`, `Woman`, `Man` i paketet `human`. Testprogrammet `TestHuman` ligger på nivån ovan (defaultpaketet) och får endast skapa instanser via fabriken. Det ska inte gå att kompilera `new NonBinary(...)` eller `new Human(){}`.

1.1 Bygga och kompilera

Precis som i uppgift 1 låter vi denna `.nw` tängla en maskingenererad `HumanFactory.mk` som toppens `Makefile` *inkluderar*.

Först lägger vi grundmålen, inkl. PDF:

⟨*HumanFactory.mk*⟩≡
```
TARGETS= HumanFactory.pdf HumanFactory.mk
all: classes-human HumanFactory.pdf

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

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

Därefter regler för att tängla ut Javakällorna (`human/` skapas vid behov) med radmarkörer (bra med `noerr.pl`):

⟨*HumanFactory.mk*⟩+≡
```
human/Human.java: HumanFactory.nw
	mkdir -p human
	notangle -L'//line %L "%F"%N' -Rhuman/Human.java HumanFactory.nw > human/Human

human/NonBinary.java: HumanFactory.nw
	mkdir -p human
	notangle -L'//line %L "%F"%N' -Rhuman/NonBinary.java HumanFactory.nw > human/No

human/Woman.java: HumanFactory.nw
	mkdir -p human
	notangle -L'//line %L "%F"%N' -Rhuman/Woman.java HumanFactory.nw > human/Woman

human/Man.java: HumanFactory.nw
	mkdir -p human
	notangle -L'//line %L "%F"%N' -Rhuman/Man.java HumanFactory.nw > human/Man.java

TestHuman.java: HumanFactory.nw
	notangle -L'//line %L "%F"%N' -RTestHuman.java HumanFactory.nw > TestHuman.java
```

Kompilera och kör:

⟨*HumanFactory.mk*⟩+≡
```
.PHONY: classes-human run-human clean-HumanFactory
classes-human: human/Human.java human/NonBinary.java human/Woman.java human/Man.java Te
	@if [ -x ./noerr.pl ]; then ./noerr.pl javac human/*.java TestHuman.java; else

run-human: classes-human
	java TestHuman
```

Städregler:

⟨*HumanFactory.mk*⟩+≡
```
clean: clean-HumanFactory
clean-HumanFactory:
        rm -f HumanFactory.tex HumanFactory.aux HumanFactory.log HumanFactory.toc
        rm -f HumanFactory.mk HumanFactory.pdf
        rm -f human/*.class *.class
        rm -f TestHuman.java human/*.java
        rmdir human 2>/dev/null || true
```

2 Kod

Här följer klasserna i paketet `human` och testprogrammet. Varje fil presenteras med en översikt och därefter delsteg (chunks) i samma stil som i uppgift 1.

2.1 `Human` (abstrakt)

Ansvar: bära gemensamma fält (`name`, `pnr`) och erbjuda fabriksmetoden.

Filen `human/Human.java` ser översiktligt ut så här:

⟨*human/Human.java*⟩≡
```
package human;

public abstract class Human {
        ⟨Human fields⟩
        ⟨Human ctor (package-private)⟩
        ⟨Human factory⟩
        ⟨Human methods⟩
}
```

2.1.1 Fält

Vi lagrar namn och personnummer (oföränderliga efter konstruktion).

⟨*Human fields*⟩≡
```
        protected final String name;
        protected final String pnr;
```

2.1.2 Konstruktor (paket-synlig)

Konstruktorn är *paketsynlig* (ingen modifierare) så att kod utanför `human` inte kan `new`a `Human` eller anonym subklass.

⟨*Human ctor (package-private)*⟩≡
```
Human(String name, String pnr) {
        if (name == null || name.isBlank()) throw new IllegalArgumentException
        if (pnr == null || pnr.length() < 2) throw new IllegalArgumentExceptio
        this.name = name;
        this.pnr = pnr;
}
```

2.1.3 Fabriksmetod

Vi följer uppgiftens tumregel på *näst sista tecknet*:

- `'0'` ⇒ `NonBinary`
- udda siffra ⇒ `Man`
- jämn siffra (ej `'0'`) ⇒ `Woman`

Vi erbjuder den variant som används i exemplen (`create(name, pnr)`). Vill man, kan man lägga till en `create(pnr)` som ger ett standardnamn.

⟨*Human factory*⟩≡
```
public static Human create(String name, String pnr) {
        if (pnr == null || pnr.length() < 2) throw new IllegalArgumentExceptio
        char c = pnr.charAt(pnr.length() - 2); // näst sista tecknet
        if (c == '0') {
                return new NonBinary(name, pnr);
        }
        if (!Character.isDigit(c)) {
                throw new IllegalArgumentException("pnr: näst sista tecknet mås
        }
        int d = c - '0';
        if ((d % 2) == 1) {
                return new Man(name, pnr);
        } else {
                return new Woman(name, pnr);
        }
}
```

2.1.4 Metoder

Vi exponerar namn och pnr. `toString()` implementeras i subklasserna.

⟨*Human methods*⟩≡
```
public String getName() { return name; }
public String getPnr() { return pnr; }
@Override public abstract String toString();
```

2.2 NonBinary

Ansvar: konkret `Human` för ickebinär. Klassen är *paketsynlig* (ingen `public`) och *final* så klienten varken kan referera till den eller ärva utanför paketet.

Filen `human/NonBinary.java` ser ut så här:

⟨*human/NonBinary.java*⟩≡
```
package human;

final class NonBinary extends Human {
    ⟨NonBinary ctor⟩
    ⟨NonBinary toString⟩
}
```

2.2.1 Konstruktor

Också paketsynlig (ingen modifierare) — kan inte anropas utanför paketet.

⟨*NonBinary ctor*⟩≡
```
NonBinary(String name, String pnr) {
    super(name, pnr);
}
```

2.2.2 toString

⟨*NonBinary toString*⟩≡
```
@Override
public String toString() {
    return "Jag är icke-binär och heter " + name;
}
```

2.3 Woman

Ansvar: konkret `Human` för kvinna. Paketsynlig och `final`.
Filen `human/Woman.java`:

⟨*human/Woman.java*⟩≡
```
package human;

final class Woman extends Human {
        ⟨Woman ctor⟩
        ⟨Woman toString⟩
}
```

2.3.1 Konstruktor

⟨*Woman ctor*⟩≡
```
        Woman(String name, String pnr) {
                super(name, pnr);
        }
```

2.3.2 `toString`

⟨*Woman toString*⟩≡
```
        @Override
        public String toString() {
                return "Jag är kvinna och heter " + name;
        }
```

2.4 Man

Ansvar: konkret `Human` för man. Paketsynlig och `final`.
Filen `human/Man.java`:

⟨*human/Man.java*⟩≡
```
package human;

final class Man extends Human {
        ⟨Man ctor⟩
        ⟨Man toString⟩
}
```

2.4.1 Konstruktor

⟨*Man ctor*⟩≡
```
        Man(String name, String pnr) {
                super(name, pnr);
        }
```

2.4.2 toString

⟨*Man toString*⟩≡
```
@Override
public String toString() {
        return "Jag är man och heter " + name;
}
```

2.5 TestHuman

Ansvar: visa fabriksanvändning och utskrift. Testprogrammet ligger i default-paketet och kan endast skapa objekt via `Human.create`.

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

⟨*TestHuman.java*⟩≡
```
import human.Human;

public class TestHuman {
        ⟨TestHuman main⟩
}
```

2.5.1 main

Vi skapar ett objekt av varje typ via fabriken och skriver ut.

⟨*TestHuman main*⟩≡
```
public static void main(String[] args) {
        Human billie = Human.create("Billie", "xxxxxx-560x"); // näst sista = (
        Human anna   = Human.create("Anna",   "xxxxxx-642x"); // näst sista = 2
        Human magnus = Human.create("Magnus", "xxxxxx-011x"); // näst sista = 1

        System.out.println(billie);
        System.out.println(anna);
        System.out.println(magnus);

        // Följande rader ska INTE kompilera (demonstreras separat, inte i dett
        // NonBinary nb = new NonBinary("X", "000000-5600");    // ej synlig ut
        // Human h = new Human("X", "000000-5600") {};          // Human() ej s
}
```