Lambda funktioner

Funktionelt interface

Et funktionelt interface er et interface med præcis én abstrakt metode. Et interface indeholder beskrivelser af abstrakte metoder i form af metodens signatur og returtype (en metode signatur i Java består af metodens navn og typerne på metodens parametre).

Her er et eksempel på et funktionelt interface:

```
@FunctionalInterface
public interface Filter {
    boolean accept(String s);
}
```

Interface ALTID public

Interfacet *Filter* herover beskriver én metode:

- metoden hedder *accept*
- metoden tager en parameter af String type
- metoden returnerer en værdi af boolean type.

Lambda funktion

En lambda funktion er en metode med navn, parametre og returtype som enhver anden metode. Det specielle ved en lambda funktion er syntaksen for metoden[1].

En lambda funktion er uløseligt knyttet til et funktionelt interface, da variable af funktionel interface type er de eneste variable, som kan referere til en lambda funktion.

Koden herunder erklærer en variabel af funktionel interface type, som refererer til en lambda funktion:

```
Filter f5 = (String str) -> {
    return str.length() > 5;
};
```

Lambda funktionen står på højre side i assignment sætningen. Højresiden definerer en metode, som tager en parameter af String type og returnerer en værdi af boolean type. Parametre til en lambda funktion står i parentesen til venstre for pilen (->), og metodens krop er udtrykket i krøllede paranteser til højre for pilen. Metodens navn er *accept()*. Det kan ikke ses på lambdafunktionen, men da det funktionelle interface *Filter* beskriver præcis én metode med navnet *accept*, så kan metodens navn udledes heraf. Metodens returtype er boolean. Det kan ses på definitionen af *accept()* metoden i *Filter*, men det kan også ses indirekte på lambda funktionen ved at se på return-sætningen.

1 Faktisk er en lambda funktion ikke en metode, da en metode per definition er en del af klasse. En lambda funktion er ikke en del af en klasse, men har ligesom en metode navn, parametre og returværdi.

Variablen *f5* kan nu bruges til at kalde metoden *accept()*:

```
boolean accepted = f5.accept("Lambda funktion");
System.out.println(accepted);
```

Ovenstående kode vil printe *true*, da argumentet til *accept()* metoden har en længde større end 5.

Den mest almindelige anvendelse af en lambda funktion er som parameter til en metode. Her er et eksempel:

```
public static int countAccepted(ArrayList<String> list, Filter f) {
    int count = 0;
    for (String s : list) {
        if (f.accept(s)) {
            count++;
        }
    }
    return count;
}
```

Metoden tæller hvor mange af strengene i ArrayList'en, som accepteres af *accept()* metoden. Metoden *countAccepted()* er en generel tælle metode, hvor *accept()* metoden bestemmer, hvad der tælles.

Her er kode, som tæller strenge længere end 5 (*strings* er en ArrayList som indeholder at antal objekter af type String):

```
Filter f5 = (String str) -> {
    return str.length() > 5;
};
int countLongerThan5 = countAccepted(strings, f5);
System.out.println(countLongerThan5);
```

Herunder er kode, som tæller strenge som starter med bogstavet *m*:

```
Filter fm = (String str) -> {
    return str.startsWith("m");
};
int countStartsWithM = countAccepted(strings, fm);
System.out.println(countStartsWithM);
```

Lambda funktion skrevet med forkortet syntaks

Ovenfor skrev vi lambda funktionen med navnet *accept()* således:

```
Filter f5 = (String str) -> {
    return str.length() > 5;
};
```

Metodens navn og metodens returtype fremgår ikke umiddelbart af koden herover, men det blev udledt fra det funktionelle interface *Filter*:

```
@FunctionalInterface
public interface Filter {
    boolean accept(String s);
}
```

Parametre til en lambda funktion kan også udledes fra det funktionelle interface. Definitionen af *accept()* metoden i *Filter* viser, at metoden tager én parameter af type String. Det samme må derfor gælde for lambda funktionen. Derfor kan man udelade typen på parametre til en lambda funktion. Udelad typen på parametre
Eksemplet med variablen f5 kan derfor forkortes til (bemærk, at typen String på parameteren er fjernet):

```
Filter f5 = (str) -> {
    return str.length() > 5;
};
```

I specielle tilfælde kan syntaksen for en lambda funktion forkortes yderligere.

Hvis der kun er én parameter til lambda funktionen, kan parentesen omkring parameteren fjernes:

Kun én parameter == udelad parantes

```
Filter f5 = str -> {
    return str.length() > 5;
};
```

Hvis der kun er én kommando i en lambda funktions krop, kan de krøllede paranteser og en evt. return fjernes:

Kun én kommando lambdas krop == krøllede parenteser og return kan fjernes

```
Filter f5 = str ->
    str.length() > 5;
```

Det er så almindeligt at skrive lambda funktionen på én linie:

```
Filter f5 = str -> str.length() > 5;
```

Når man har denne helt korte syntaks for en lambda funktion, så undlader man ofte at definere en variabel af funktionelt interface type, når en lambda funktion skal bruges som parameter til en metode.

Koden

```
Filter f5 = (String str) -> {
    return str.length() > 5;
};
int countLongerThan5 = countAccepted(strings, f5);
System.out.println(countLongerThan5);
```

bliver så til (variablen *f5* er udeladt)

```
int countLongerThan5 = countAccepted(strings, str -> str.length() > 5);
System.out.println(countLongerThan5);
```

eller endnu kortere til

```
System.out.println(countAccepted(strings, str -> str.length() > 5));
```

GUI eksempel med lambda funktion

I JavaFX har Button klassen følgende metode:

```
public class Button {
    public final void setOnAction(EventHandler<ActionEvent> handler)
    { ... }
}
```

Parameteren til metoden er det funktionelle interface EventHandler:

```
@FunctionalInterface
public interface EventHandler<ActionEvent> {
    public void handle(ActionEvent event);
}
```

Metoden setOnAction() kan derfor kaldes med en lambda funktion. Her er et simpelt eksempel:

setOnAction() kan kaldes med lambda funktion

```
Button btnPrint = new Button("Print");
btnPrint.setOnAction(event -> System.out.println("Hello World!")
```

Bemærk, at den korte form for en lambda funktion er anvendt (dvs. ingen type på parameteren og ingen krøllede paranteser).

Her er den korte lambda funktion brugt

Ofte skal lambda funktionen i tilknytning til en button udføre mange linier kode. I dette tilfælde er det bedst at definere en metode, som indeholder de mange linier kode. Lambda funktionen skal så blot indeholde et kald til metoden:

```
Button btnPrint = new Button("Print Greeting");
btnPrint.setOnAction(event -> this.printAction());
```

Metoden printAction() herover skal returnere void.

Comparator eksempel med lambda funktion

Klassen ArrayList<E> har en sort() metode, som sorterer objekterne i listen ved at bruge en comparator() metode. Metoden sort() ser således ud:

```
public class ArrayList<E> {
    public void sort(Comparator<E> comparator)  { … }
}
```

Her er Comparator<E> et funktionelt interface:

```
@FunctionalInterface
public interface Comparator<E> {
    int compare(E e1, E e2);
}
```

Metoden sort() kan derfor kaldes med en lambda funktion.

Lad os som eksempel antage, at vi har en Person klasse, hvor alle personer har et *name* felt med tilhørende *getName()* metode. Vi kan nu sortere en ArrayList<Person> ved at bruge sort() metoden med en lambda funktion:

```
persons.sort((p1, p2) -> p1.getName().compareTo(p2.getName());
```

Her benyttes, at String klassen har en compareTo() metode, som kan sammenligne strings.

Hvis du synes, at udtrykket herover ikke er let at læse, så kan lambda funktionen gemmes i en variabel som vist herunder:

```
Comparator<Person> pc = (p1, p2) → p1.getName().compareTo(p2.getName());
persons.sort(pc);
```