Java interfaces og lambda udtryk

Java interfaces kan (fra og med Java 8) indeholde 3 slags metoder:

- abstrakte metode (metoder uden krop)
- default metoder (erklæret med nøgleordet *default)*
- statiske metoder (erklæret med nøgleordet *static*).

Abstrakte metoder (metoder uden krop) *skal* overskrives i klasser, som implementerer interfacet. Default metoder er normale ikke-statiske metoder (med krop), som nedarves og *kan* overskrives i klasser, som implementerer interfacet. Statiske metoder er normale statiske metoder (med krop).

I de følgende afsnit er vist eksempler på brug af default metoder, defineret i interfaces fra Java Collections FrameWork (JCF), brugt sammen med lambda udtryk.

Iterable<E>.forEach()

Metodehovedet for metoden *forEach()*:

```
default void forEach(Consumer<E> action)
```

Typen Consumer<E> er et funktionelt interface med metoden

void action(E e)

Metoden *forEach()* kan derfor bruges sammen med et lambdaudtryk, som tager en parameter af type E og returnerer void:

Consumer<E> lambda `(E) -> void`

Da alle collections i JCF implementerer Iterable<E>, så har alle collections en forEach() metode.

Eksempel med *forEach()*:

```
List<Person> persons = ... // liste med person objekter
persons.forEach(p -> System.out.println(p.getName());
```

Kodeforklaring: For alle personer i listen udskrives navnet.

Endnu et eksempel med *forEach()*:

```
List<Person> persons = ... // liste med person objekter
persons.forEach(p -> {
      if (p.getAge() > 18
            System.out.println(p.getName());
});
```

Kodeforklaring: For alle personer i listen, ældre end 18, udskrives navnet.

Bemærk paranteserne i koden herover.

List<E>.sort()

Metodehovedet for metoden *sort()*:

```
default void sort(Comparator<E> c)
```

Typen Comparator<E> er et funktionelt interface med metoden

```
int compare(E e1, E, e2)
```

Metoden *sort()* kan derfor bruges sammen med et lambdaudtryk, som tager 2 parametre af typen E og returnerer en int:

Comparator<E> lambda: `(E e1, E e2) -> int`

Eksempel med *sort()*:

```
List<Person> persons = ... // liste med person objekter
persons.sort((p1, p2) -> p1.getAge() - p2.getAge());
```

Kodeforklaring: Personerne i listen sorteres stigende efter alder.

Eksempel med *sort()* og *thenComparing()*:

```
Comparator<Person> compareAge =
      (p1, p2) -> Integer.compare(p1.getAge(), p2.getAge());
Comparator<Person> compareName =
      (p1, p2) -> p1.getName().compareTo(p2.getName());
persons.sort(compareAge.thenComparing(compareName));
```

Kodeforklaring: Personerne i listen sorteres stigende, først efter alder og dernæst efter navn.

Metoden *thenComparing()* er en default metode i Comparator<E> interfacet med følgende metodehovede:

```
default Comparator<E> thenComparing(Comparator<E> other)
```

List<E>.removeIf()

Metodehovedet for metoden *removeIf()*:

```
default void removeIf(Predicate<E> filter)
```

Typen Predicate<E> er et funktionelt interface med metoden

```
boolean test(E e)
```

Metoden *removeIf()* kan derfor bruges sammen med et lambdaudtryk, som tager en parameter af typen E og returnerer en boolean:

Predicate<E> lambda: `(E e) -> boolean`

Eksempel med *removeIf()*:

```
List<Person> persons = ... // liste med person objekter
persons.removeIf(p -> p.getAge() < 18 || p.getAge() > 60);
```

Kodeforklaring: Personer i listen yngre end 18 eller ældre end 60 fjernes.