Lambda-uttryck

Från Java version 8.

```
parameterlista -> funktionskropp
```

Parameterlista:

```
()
(typ1 param1, typ2 param2, ...)
(param1, param2, ...)
param
```

Funktionskropp:

```
{ deklarationer och satser }
uttryck
metodanrop(...)
```

Lambda-uttryck kan ges när typen är ett funktionsgränssnitt.
(Gränssnitt med endast en funktion.)

Exempel:

```
Arrays.sort(a, (a1, a2) -> (int) (a1.getValue()- a2. getValue()));
```

```
myButton.addActionListener
        ( e -> { String name = myInput.getText();
                 myLabel.setText("Welcome " + name + "!");} );
```

gränssnittets namn	funktionens namn	funktionens profil
`Runnable`	`run`	`() -> void`
`Callable<T>`	`call`	`() -> T`
`Comparator<T>`	`compare`	`(T, T) -> boolean`
`Consumer<T>`	`accept`	`(T) -> void`
`BiConsumer<K, V>`	`accept`	`(K, V) -> void`
`UnaryOperator<T>`	`apply`	`(T) -> T`
`BinaryOperator<T>`	`apply`	`(T, T) -> T`
`Predicate<T>`	`test`	`(T) -> boolean`
`BiPredicate<K, V>`	`test`	`(K, V) -> boolean`
`Function<T, V>`	`apply`	`(T) -> V`
`BiFunction<K, V, W>`	`apply`	`(K, V) -> W`
`Supplier<T>`	`get`	`() -> T`
X`UnaryOperator`	`applyAs`*X*	(*x*) -> *x*
X`BinaryOperator`	`applyAs`*X*	(*x*, *x*) -> *x*
X`Function<T>`	`apply`	(*x*) -> *T*

X står för `Int`, `Long` eller `Double` och *x* för **`int`**, **`long`** resp. **`double`**

`Runnable` i `java.lang`
`Callable` i `java.util.concurrent`
`Comparator` i `java.util`
övriga i `java.util.function`

Klassen `Stream`

- Generisk
- Har metoder med lambda-uttryck som parametrar
- Hämtar data från listor och mängder via metoden `stream`
- Hämtar data från en `BufferedReader` via metoden `lines`
- Kan samla ihop data en `Container` eller en `String` med en `Collector`
- Använder s.k. lat evaluering
- Kan exekveras seriellt eller parallellt

Exempel:

```
words.stream()
      .filter(s -> s.length() > 10)
      .map(s -> s.toUpperCase())
      .forEach(s -> System.out.println(s));
```

```
List<String> longWords = words.stream()
                              .filter(s -> s.length() > 8)
                              .map(s -> s.toUpperCase())
                              .sorted(String::compareTo)
                              .collect(Collectors.toList());
```