Optional

Carsten Gips (FH Bielefeld)

Motivation

```
public class LSF {
    private Set<Studi> sl;

    public Studi getBestStudi() {
        if (sl == null) return null;  // Fehler: Es gibt noch keine Sammlung

        Studi best = null;
        for (Studi s : sl) {
            if (best == null) best = s;
            if (best.credits() < s.credits()) best = s;
        }
        return best;
    }
}

public static void main(String... args) {
    LSF lsf = new LSF();

    Studi best = lsf.getBestStudi();
    if (best != null) {
        String name = best.name();
        if (name != null) {
            // mach was mit dem Namen ...
        }
    }
}
```

Erzeugen von *Optional*-Objekten

Konstruktor ist `private` ...

- "Kein Wert": `Optional.empty()`
- Verpacken eines non-`null` Elements: `Optional.of()`
  (`NullPointerException` wenn Argument `null`!)

- Verpacken eines "unsicheren"/beliebigen Elements: `Optional.ofNullable()`
  - Liefert verpacktes Element, oder
  - `Optional.empty()`, falls Element `null` war

`null` kann nicht nicht in `Optional<T>` verpackt werden!

LSF liefert jetzt *Optional* zurück

```java
public class LSF {
    private Set<Studi> sl;

    public Optional<Studi> getBestStudi() throws NullPointerException {
        // Fehler: Es gibt noch keine Sammlung
        if (sl == null) throw new NullPointerException("There ain't any collection");

        Studi best = null;
        for (Studi s : sl) {
            if (best == null) best = s;
            if (best.credits() < s.credits()) best = s;
        }

        // Entweder Optional.empty() (wenn best==null) oder Optional.of(best) sonst
        return Optional.ofNullable(best);
    }
}
```

Zugriff auf *Optional*-Objekte

```
Studi best;

// Testen und dann verwenden
if (!lsf.getBestStudi().isEmpty()) {
    best = lsf.getBestStudi().get();
    // mach was mit dem Studi ...
}

// Arbeite mit Consumer
lsf.getBestStudi().ifPresent(studi -> {
    // mach was mit dem Studi ...
});

// Studi oder Alternative (wenn Optional.empty())
best = lsf.getBestStudi().orElse(anne);

// Studi oder NoSuchElementException (wenn Optional.empty())
best = lsf.getBestStudi().orElseThrow();
```

Einsatz mit Stream-API

```
public class LSF {
    ...
    public Optional<Studi> getBestStudi() throws NullPointerException {
        if (sl == null) throw new NullPointerException("There ain't any collection");
        return sl.stream()
                 .sorted((s1, s2) -> s2.credits() - s1.credits())
                 .findFirst();
    }
}


public static void main(String... args) {
    ...
    String name = lsf.getBestStudi()
                     .map(Studi::name)
                     .orElseThrow();
}
```

Regeln für *Optional*

1. Nutze `Optional` nur als Rückgabe für "kein Wert vorhanden"
2. Nutze nie `null` für eine `Optional`-Variable oder einen `Optional`-Rückgabewert
3. Nutze `Optional.ofNullable()` zum Erzeugen eines `Optional`
4. Erzeuge keine `Optional` als Ersatz für die Prüfung auf `null`
5. Nutze `Optional` nicht in Attributen, Methoden-Parametern und Sammlungen
6. Vermeide den direkten Zugriff (`ifPresent()`, `orElseThrow()` ...)

Wrap-Up

`Optional` als Rückgabe für "kein Wert vorhanden"

- `Optional.ofNullable()`: Erzeugen eines `Optional`
  - Entweder Objekt "verpackt" (Argument != `null`)
  - Oder `Optional.empty()` (Argument == `null`)
- Prüfen mit `isEmpty()` und `ifPresent()`
- Direkter Zugriff mit `ifPresent()`, `orElse()` und `orElseThrow()`
- Stream-API: `map()`, `filter()`, `flatMap()`, ...

- Attribute, Parameter und Sammlungen: nicht `Optional` nutzen
- Kein Ersatz für `null`-Prüfung!

LICENSE