Java 8 API Design Prinzipien Nachschlagewerk

Optional <T>

- Die Klasse Optional <T> stellt Methoden bereit, um den oft lästigen Umgang mit null-Werten deutlich zu vereinfachen
- Ein Optional kann man sich als Datenbehälter vorstellen, der entweder einen Wert enthält oder leer (Optional.empty) ist. Das ist nicht gleichbedeutend mit null!
- An Schnittstellen wird Optional <T> z.B an Stelle von Referenzen verwendet, die null sein können
- Fazit: Optional <T> kann sowohl helfen, Code zu vereinfachen, als auch NullPointerExceptions zu vermeiden. Doch in der Praxis funktioniert dieser Ansatz nur, wenn Optional konsequent im eigenen Code umgesetzt wird.

Streams anstatt Arrays

- Ein Stream repräsentiert eine Folge von Elementen und unterstützt verschiedene Arten von Operationen, um Berechnungen an diesen Elementen durchzuführen.
- Mit Java 8 verfügt die Collection Schnittstelle über zwei Methoden zum Generieren eines Stream: stream() und parallelStream().
- Streams stellen Ströme von Referenzen dar, die es erlauben, verkettete Operationen auf diesen Referenzen nacheinander oder parallel auszuführen. Die Daten, die durch die Referenzen repräsentiert werden, werden durch den Stream selbst nicht verändert.
- Fazit: Ein Stream ist only Readable, im Gegensatz dazu lässt sich das Array modifizieren.

Das Interface und die von ihm abgeleiteten Interfaces stellen lediglich eine Vielzahl von Methoden bereit, die in zwei Hauptkategorien eingeteilt werden und meist Lambda Ausdrücke als Argumente übergeben bekommen:

- **intermediäre Operationen** (intermediate operations) liefern einen Stream, der weiterverarbeitet werden kann
    - (z.B. filter(), map(), distinct(), sorted(), etc.).

- **terminale Operationen** (terminal operations) führen ihrerseits Operationen auf den Referenzen des Streams aus
    - (forEach(), reduce(), toArray(), etc.).
    - Sie können einen Wert liefern und beenden den Strom. Ist ein Strom einmal geschlossen, so können keine weiteren Operationen auf ihm ausgeführt werden.

Wie kann man Streams erzeugen?
Streams können aus Arrays, Listen, anderen Collections und aus Einzelobjekten, sowie mittels sog. StreamBuilder erzeugt werden. Je nach verwendeter Methode kann das Ergebnis jedoch unterschiedlich ausfallen.

Funktionale Interfaces und Lamdas

- Lambda-Ausdrücke sind Implementierungen von funktionalen Schnittstellen, also Schnittstellen mit genau einer abstrakten Methode, und eine Alternative und Abkürzung zu Klassen, die Schnittstellen implementieren. So lässt sich sehr einfach Programmcode ausdrücken und an anderen Methoden übergeben.
- Lambda-Ausdrücke und funktionale Schnittstellen haben eine ganz besondere Beziehung, denn ein Lambda-Ausdruck ist ein Exemplar einer solchen funktionalen Schnittstelle. Natürlich müssen Typen und Ausnahmen passen. Dass funktionale Schnittstellen genau eine abstrakte Methode vorschreiben, ist eine naheliegende Einschränkung, denn gäbe es mehrere, müsste ein Lambda-Ausdruck ja auch mehrere Implementierungen anbieten oder irgendwie eine Methode bevorzugen und andere ausblenden.
- Wenn wir ein Objekt vom Typ einer funktionalen Schnittstelle aufbauen möchten, können wir folglich zwei Wege einschlagen: Wir können die traditionelle Konstruktion über die Bildung von Klassen wählen, die funktionale Schnittstellen implementieren, und dann mit new ein Exemplar bilden, oder wir können mit kompakten Lambda-Ausdrücken arbeiten.
- In Java Version 8 wurde der Lambda-Ausdruck eingeführt, um die funktionale Programmierung mit Java zu ermöglichen. Funktionen können also in Form eines Lambda-Ausdrucks an Methoden als Parameter übergeben werden. In Java erkennt man das Lambda am Pfeiloperator: ->

- Fazit: Lambda-Ausdrücke werden mittlerweile in Java Programmen sehr häufig eingesetzt, da man mit ihnen deutlich kompakteren Code schreiben kann. Insbesondere in Kombination mit Streams sind sie ein tolles Mittel, um lange Schleifen zu ersetzen. Funktionale Interfaces sind eine notwendige Voraussetzung für den Einsatz von Lambdas.

@FunctionalInterface Annotation

Im vorherigen Abschnitt haben wir erfahren, dass jedes Funktionale Interface mit nur einer abstrakten Methode mit einem Lambda-Ausdruck implementiert werden kann.
Diese Eigenschaft eines Interfaces kann man auch durch die Annotation @FunctionalInterface festschreiben. Zum Beispiel wird man bei der Deklaration des Interfaces Function die Annotation voranstellen:

```
@FunctionalInterface
public interface Function<T, R> { ... }
```

Die Annotation ist zwar nicht zwingend vorgeschrieben, der Compiler garantiert aber nun, dass das Interface wirklich nur eine abstrakte Methode hat.

Vermeide Overloading bei Methoden in Interfaces

- Der Begriff Überladen beschreibt eine Technik in der objektorientierten Programmierung, die es einer abgeleiteten Klasse erlaubt, eine eigene Implementierung einer von der Basisklasse geerbten Methode zu definieren.
- Wie nennt man das erneute Definieren einer Methode mit dem gleichen Namen?
    - Dieses Konzept, das mehrfache Verwenden des gleichen Methodennamens, wird als Überladen von Methoden bezeichnet.
- Beim Überladen von Methoden in einem Interface, müssen sich die Methodennamen unterscheiden, da ansonsten der Lamda-Ausdruck die Methoden nicht unterscheiden kann.

Parametercheck

Bei einem Parametercheck geht es darum ein Objekt auf Richtigkeit zu prüfen, bevor man damit arbeitet. In der Präsentation z.B wurde der Inhalt auf null geprüft und da dieser null war, wurde dieser nicht hinzugefügt. Ohne einen Parametercheck, wäre der null Wert hinzugefügt worden, was man oft nicht möchte.

Default Methoden in Interfaces

- Eine Default Methode ist im Gegensatz zu einer abstrakten Methode, eine Methode mit einem body Inhalt. Dieser wird so an eine implementierende Klasse übergeben, ohne überschrieben werden zu müssen.
- Bemerkenswert ist, dass Default-Methoden dazu führen, dass Java mit der Version 8 Mehrfachvererbung von Funktionalität ermöglicht. Zu Mehrfachvererbung kommt es zum Beispiel, wenn eine Klasse mehrere Interfaces implementiert, die alle Default-Methoden enthalten. Die Funktionalität aller dieser Default-Methoden ist dann natürlich in der abgeleiteten Klasse verfügbar
- Zusätzlich kann unsere Klasse natürlich zusätzlich von einer anderen Klasse abgeleitet sein und deren Funktionalität auch noch erben. Damit ist es dann in Java 8 möglich, dass eine Klasse die Funktionalität von einer Super-Klasse und beliebig vielen Super-Interfaces (mit Default-Methoden) erbt.
- Bisher war es in Java so, dass Methoden in Interfaces abstrakt sein mussten. Das heißt, eine Methode in einem Interface legte allein ihre Signatur und ihre allgemeine Semantik fest. Eine Implementierung konnten sie nicht haben. Die Implementierung wurde erst von den (nicht-abstrakten) Klassen, die von dem Interface abgeleitet waren, zur Verfügung gestellt.

Was ist eine API und wofür wird es verwendet?

Die Abkürzung API ist ziemlich verbreitet. Vor allem unter denjenigen, die aktiv mit verschiedenen Programmen auf dem PC interagieren.

Viele Benutzer verwenden heute die Synchronisierung von Programmen zwischen mehreren Geräten sowie die Synchronisierung zwischen zwei verschiedenen Programmen. Bei der Synchronisation wird eine Verbindung zwischen zwei Softwareprogrammen hergestellt, die einen unterschiedlichen Quellcode haben.

Um die zusätzliche Arbeit der Kontaktaufnahme zwischen den verschiedenen Programmen zu vermeiden, wurde eine API (Application Programming Interface) geschaffen. Eine Anwendungsprogrammierschnittstelle (API) ist ein Satz von Regeln und Definitionen, die es verschiedenen Technologien ermöglichen, miteinander zu kommunizieren. Eine gut konstruierte, sichere und gut dokumentierte API ist die Grundlage für die Softwareentwicklung. Der Benutzer kann nur auf bestimmte Daten zugreifen, und alle sensiblen Informationen bleiben privat.

Eine API verbirgt den Codeteil und stellt die Kommunikation zwischen Programmen her. Die API kann man sich als eine Art Brücke vorstellen.

Fazit
Eine API ist ein Satz von Befehlen, Funktionen, Protokollen und Objekten, die Programmierer verwenden können, um eine Software zu erstellen oder mit einem externen System zu interagieren. Sie stellt Entwicklern Standardbefehle für die Ausführung allgemeiner Operationen zur Verfügung, so dass Codes nicht von Grund auf neu geschrieben werden müssen.

Die API – auch Programmierstelle genannt- ermöglicht es demnach Anwendungen miteinander zu kommunizieren. Die API ist nicht die Datenbank oder gar der Server, sondern der Code, der die Zugangspunkte für den Server regelt und die Kommunikation ermöglicht.

Somit wird der Datenaustausch zwischen verschiedenen Systemen um ein Vielfaches beschleunigt und vereinfacht.