Testen von Software
Semesterprojekt - SS 2023

Hinweise

- **Abgabe erfolgt über Moodle und es gibt ein Abnahme im Labor. Termine und Links in Moodle.**
- Ein Grundprojekt ist im Moodle unter `https://gitlab.iue.fh-kiel.de/tsw/semesterprojekt-rumpf` vorhanden.

Aufgabe
Erstellen Sie in einem **2-3 Personen Team** Software mit Behaviour Driven Development(BDD) in Java die folgende Prozess- und Produktanforderungen erfüllt.

1. Prozessanforderungen:
   (a) Erstellen Sie die Software mit BDD in Java.
   (b) Alle Testfälle müssen sinnvoll dokumentiert sein.
   (c) Ihre Software muss sinnvoll vollständig durch (automatisierte) Testfälle abgedeckt sein.
      - Jede Unit muss sinnvoll vollständig getestet werden.
      - Das Zusammenspiel von Units muss getestet werden.
   (d) Der Sourcecode muss alle sinnvollen Prüfungen der statischen Analyse durch Sonarlint erfüllen.
   (e) Der Sourcecode muss muss alle sinnvollen Prüfungen des Google-Java-Checkstyle bestehen.
   (f) Alle Produkte müssen durch ein dokumentiertes Review abgenommen werden.
   (g) Alle nicht-Code Produkte müssen in einem abschließenden Bericht dokumentiert und diskutiert werden.
2. Produktanforderungen
   Das Produkt ist ein Plugin-Modul für Software zur Nutzung einer Bibliotheksdatenbank gedacht. Es soll grundlegende Suchen im Buchbestand der Bibliothek ermöglichen. Dazu müssen folgende Anforderungen erfüllt sein:
   (a) Die Software muss als Softwarebibliothek umgesetzt werden.
   (b) Alle gegebenen Schnittstellen(`Book, Search, SearchParameter, Searchparameter.Builder`) müssen von dieser Softwarebibliothek unverändert implementiert werden.
   (c) Primäre Nutzung der Softwarebibliothek erfolgt über die Schnittstelle `Search`.
   (d) Die Suche erfolgt über eine gegebene Liste von Büchern.
   (e) Die Suche erlaubt aus der gegebenen Liste von Büchern ein Buch über die Identifikationsnummer anzusehen. Falls es kein Buch mit der Identifikationsnummer gibt, wird kein Buch zurückgegeben.
   (f) Die Suche sucht nach Büchern mit einer Reihe von Suchparametern.
   (g) Die Suche stellt eine Hilfsfunktion bereit, um Suchparameter zu erstellen.
   (h) Die Suche speichert alle auf ihr getätigten Suchen mit den Ergebnissen und ermöglicht den Zugriff auf diese.

(i) Die Suche sollte maximal 2 Sekunden laufen und sonst mit einer sinnvollen Fehlermeldung abgebrochen werden.

(j) Bücher besitzen eine eindeutige Identifikationsnummer.

(k) Bücher haben einen Namen und mindestens einen Autor.

(l) Bücher können einer beliebige Menge an Schlüsselworten zugeordnet sein.

(m) Bücher haben ein Rückgabedatum, falls sie ausgeliehen sind.

(n) Bücher haben einen Zähler, wie oft sie ausgeliehen wurden.

(o) Bücher haben Datum an dem sie gekauft wurden.

(p) Bücher haben einen Zustand, der ihre Abnutzung beschreibt (`Brocken, Bad, Good, New`).

(q) Suchparameter können Namen sein.

(r) Suchparameter können Autoren sein.

(s) Suchparameter können Schlüsselwörter sein.

(t) Suchparameter kann der Ausleihstatus sein.

(u) Suchparameter kann die Abfrage, ob ein Buch nach dem gegeben Datum ausgeliehen ist, sein.

(v) Suchparameter kann ein Kaufzeitfenster sein.

(w) Suchparameter kann wie oft ein Buch ausgeliehen wurde sein.

(x) Suchparameter kann der Zustand sein.

(y) Suchparameter können beliebig genutzt und weggelassen werden.

(z) Suchparameter können mit einem Builder(Link) erstellt werden.

Gegebener Code:

Search.java:

```
package de.fhkiel.library.search;

import java.util.List;
import java.util.Map;
import javax.naming.TimeLimitExceededException;

/**
 * The interface Search.
 * Acts as the primary point of use.
 */
public interface Search {
    /**
     * Add {@link Book}s that should be searched.
     * Can be called numerous times, adding the given {@link Book}s.
     *
     * @param books the {@link Book}s to add
     */
    void addBooks(List<Book> books);
```

```
    /**
     * Gets a book.
     *
     * @param id the {@link Book} id
     * @return the {@link Book} or null if no such {@link Book} exists
     */
    Book getBook(int id);

    /**
     * Search for books.
     *
     * @param search the parameters for the search
     * @return the found {@link Book}s
     *
     * @exception TimeLimitExceededException when search takes to long
     */
    List<Book> getBooks(SearchParameter search) throws TimeLimitExceededException;

    /**
     * Gets a {@link SearchParameter.Builder} for the {@link SearchParameter}.
     *
     * @return the {@link SearchParameter.Builder}
     */
    SearchParameter.Builder createSearchParameter();

    /**
     * Returns the unordered history of this search object.
     *
     * @return the {@link Map} of {@link SearchParameter}s to Results
     */
    Map<SearchParameter, List<Book>> history();
}
```

Book.java:

```
package de.fhkiel.library.search;

import java.time.LocalDate;
import java.util.List;
import java.util.Optional;

/**
 * The interface Book.
 */
public interface Book{
  /**
   * The book id.
   * Must be Unique.
   *
   * @return the id
   */
  int id();
```

```
  /**
   * The name of the book.
   *
   * @return the name
   */
  String name();

  /**
   * The {@link List} of authors.
   *
   * @return the authers
   */
  List<String> authors();

  /**
   * A list of keywords.
   *
   * @return the keywords
   */
  List<String> keywords();

  /**
   * An {@link Optional} containing the date a book is due to be returned.
   * Can be empty if a book is not borrowed.
   *
   * @return the optional date
   */
  Optional<LocalDate> borrowedTill();

  /**
   * The {@link LocalDate} a book was bought.
   *
   * @return the date
   */
  LocalDate bought();

  /**
   * A counter how many times a book has been borrowed.
   *
   * @return the counter
   */
  int timesBorrowed();

  /**
   * The {@link Condition} of the book.
   *
   * @return the {@link Condition}
   */
  Condition condition();
}
```

Condition.java:

```
package de.fhkiel.library.search;

/**
 * The enum Condition.
 * Denominates classes of possible conditions a book can be in.
 */
public enum Condition {
    BROCKEN, BAD, GOOD, NEW;
}
```

SearchParameter.java:

```
package de.fhkiel.library.search;

import java.time.LocalDate;
import java.util.List;

/**
 * The interface  for search parameters.
 */
public interface SearchParameter{
  /**
   * A list of names to search for.
   *
   * @return the list of names
   */
  List<String> names();

  /**
   * A list of authors to search for.
   *
   * @return the list of authors
   */
  List<String> authors();

  /**
   * A list of keywords to search for.
   *
   * @return the list of keywords
   */
  List<String> keywords();

  /**
   * If a book is borrowed.
   *
   * @return the borrowed-state
   */
  boolean borrowed();

  /**
   * If a book is borrowed after this date.
   *
   * @return the date
```

```
 */
LocalDate borrowedAfter();

/**
 * Bought after this date.
 *
 * @return the date
 */
LocalDate boughtAfter();

/**
 * Bought before this date.
 *
 * @return the date
 */
LocalDate boughtBefore();

/**
 * Min times borrowed.
 *
 * @return the min number of times
 */
int minTimesBorrowed();

/**
 * Max times borrowed.
 *
 * @return the max number of times
 */
int maxTimesBorrowed();

/**
 * A list of acceptable conditions.
 *
 * @return the acceptable conditions
 */
List<Condition> acceptableConditions();

/**
 * The Builder interface to create a search parameter.
 * <a href="https://refactoring.guru/design-patterns/builder">Pattern</a>
 */
interface Builder{
  /**
   * Add names to search for.
   *
   * @param name the names as a vararg
   * @return the builder
   */
  Builder addNamesToSearch(String... name);
```

```
/**
 * Add authors to search for.
 *
 * @param author the authors as a vararg
 * @return the builder
 */
Builder addAuthorsToSearch(String... author);

/**
 * Add keywords to search for.
 *
 * @param keyword the keywords as a vararg
 * @return the builder
 */
Builder addKeywordsToSearch(String... keyword);

/**
 * Is the book borrowed now.
 *
 * @param borrowed the borrowed state
 * @return the builder
 */
Builder bookIsBorrowedNow(boolean borrowed);

/**
 * Book is borrowed after.
 *
 * @param date the date
 * @return the builder
 */
Builder bookIsBorrowedAfter(LocalDate date);

/**
 * Book was bought after.
 *
 * @param date the date
 * @return the builder
 */
Builder bookWasBoughtAfter(LocalDate date);

/**
 * Book was bought before.
 *
 * @param date the date
 * @return the builder
 */
Builder bookWasBoughtBefore(LocalDate date);

/**
 * Book was borrowed at least this many times.
 *
```

```
     * @param min the min times borrowed
     * @return the builder
     */
    Builder bookWasBorrowedAtLeastTimes(int min);

    /**
     * Book was borrowed at most this many times.
     *
     * @param max the max times borrowed
     * @return the builder
     */
    Builder bookWasBorrowedAtMostTimes(int max);

    /**
     * Acceptable conditions the books can be in.
     *
     * @param condition the conditions as a vararg
     * @return the builder
     */
    Builder acceptableConditions(Condition... condition);

    /**
     * Create the {@link SearchParameter} for the search.
     *
     * @return the search parameter
     */
    SearchParameter createParameterForSearch();
  }
}
```