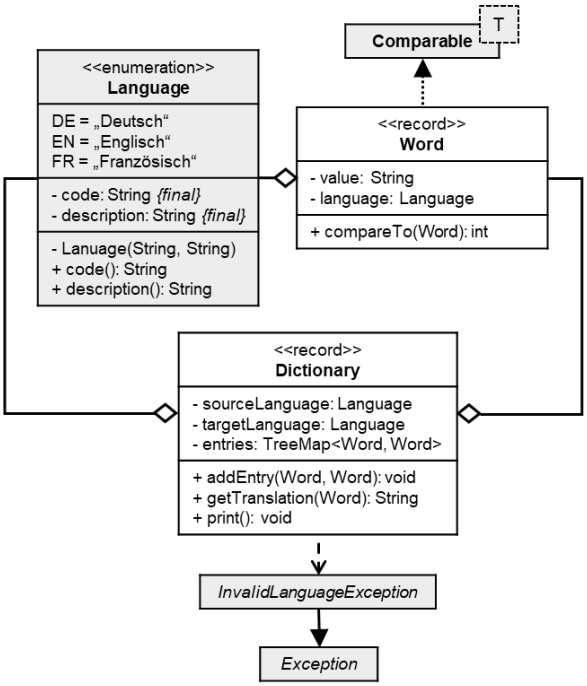
Aufgabe 1 (12 Punkte)

a) Erläutere kurz, was man unter einer **Schnittstelle (Interface)** versteht (2 Punkte)
b) Benenne die **4 Varianztypen** bei der generischen Programmierung (2 Punkte)
c) Erläutere kurz den wesentlichen Zweck von **inneren Klassen** (2 Punkte)
d) Erläutere kurz den wesentlichen Unterschied zwischen **intermediären und terminalen Operationen** (2 Punkte)
e) Erläutere kurz, was man unter der **Red-Green-Refactor-Methode** versteht (2 Punkte)
f) Skizziere die **Testpyramide** (2 Punkte)

Aufgabe 2 (13 Punkte)

Erstelle die Klassen `Word` (3 Punkte) und `Dictionary` (10 Punkte) anhand des abgebildeten Klassendiagrams

Klassendiagramm



Hinweis zur Klasse Word

Die Methode `compareTo(Word)` soll so implementiert werden, dass damit Wörter aufsteigend nach ihrem Wert (`value`) sortiert werden können.

Hinweise zur Klasse Dictionary

- Die Methode `addEntry(Word, Word)` soll dem Wörterbuch die beiden eingehenden Wörter hinzufügen. Für den Fall, dass die Sprache des ersten eingehenden Wortes (`language`) nicht der Quellsprache des Wörterbuchs (`sourceLanguage`) entspricht, oder dass die Sprache des zweiten eingehenden Wortes (`language`) nicht der Zielsprache des Wörterbuchs (`targetLanguage`) entspricht, soll die Ausnahme `InvalidLanguageException` ausgelöst werden
- Die Methode `getTranslation(Word)` soll zum eingehenden Wort die dazugehörige Übersetzung als Zeichenkette zurückgeben. Für den Fall, dass zum eingehenden Wort keine Übersetzung gefunden werden kann, soll eine leere Zeichenkette zurückgegeben werden
- Die Methode `print()` soll alle Attribute wie abgebildet auf der Konsole ausgeben

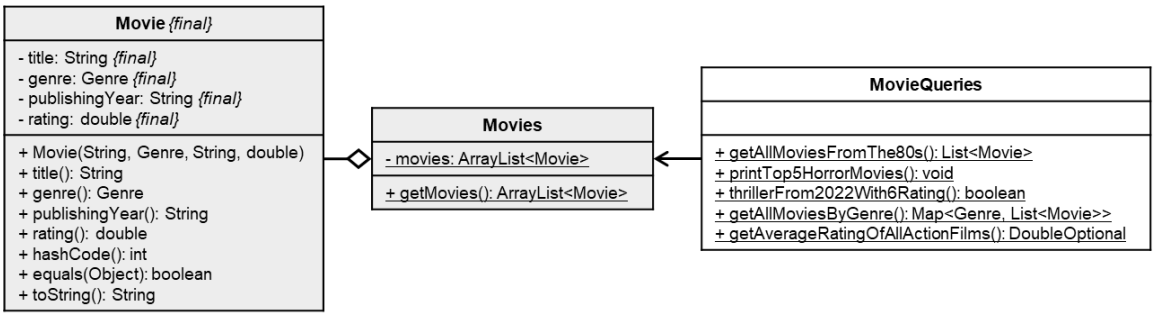
Konsolenausgabe

```
Wörterbuch Deutsch – Englisch
Haus = house
Maus = mouse
Weihnachtsmann = santa claus
…
```

Aufgabe 3 (15 Punkte)

Erstelle die Klasse `MovieQueries` anhand des abgebildeten Klassendiagramms.

Klassendiagramm



Hinweise zur Klasse MovieQueries

- Die statische Methode `getAllMoviesFromThe80s()` soll alle Filme der Klasse `Movies` zwischen 1980 und 1990 als Liste zurückgeben
- Die statische Methode `printTop5HorrorMovies()` soll die ersten 5 Horror-Filme der Klasse `Movies` absteigend sortiert nach der Bewertung in der Form *FILMTITEL (Erscheinungsjahr)* ausgeben
- Die statische Methode `thrillerFrom2022With6Rating()` soll zurückgeben, ob es in der Klasse `Movies` einen Thriller aus dem Jahr 2022 mit einer Bewertung von 5,5 bis 6,4 gibt
- Die statische Methode `getAllMoviesByGenre()` soll alle Filme je Genre als Assoziativspeicher zurückgeben
- Die statische Methode `getAverageRatingOfAllActionFilms()` soll die durchschnittliche Bewertung aller Action-Filme als Gleitpunktzahl zurückgeben

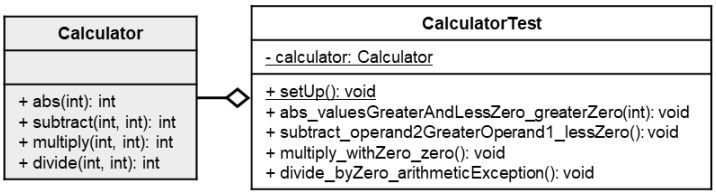
Methoden

Klasse	Methode
Stream<T>	`anyMatch(Predicate<T>): boolean`
Stream<T>	`collect(Collector<T, A, R>): R`
Stream<T>	`filter(Predicate<T>): Stream<T>`
Stream<T>	`forEach(Consumer<T>): void`
Stream<T>	`limit(): Stream<T>`
Stream<T>	`map(Function<T, R>): Stream<R>`
Stream<T>	`mapToDouble(ToDoubleFunction<T, R>): DoubleStream`
Stream<T>	`sorted(Comparator<T>): Stream<T>`
DoubleStream	`average(): OptionalDouble`
Collectors	`toList(): Collector<T, ?, List<T>>`
Collectors	`groupingBy(Function<T, K>): Collector<T, ?, Map<K, List<T>>>`
String	`toUpperCase(): String`
Predicate<T>	`test(T): boolean`
Consumer<T>	`accept(T): void`
Function<T, R>	`apply(T) R`
ToDoubleFunction<T, R>	`applyAsDouble(T): double`
Comparator<T>	`compare(T, T): int`

Aufgabe 4 (10 Punkte)

Erstelle die Testklasse `CalculatorTest` anhand des abgebildeten Klassendiagramms.

Klassendiagramm



Hinweise zur Testklasse CalculatorTest

- Die statische Lebenszyklus-Methode `setUp()` soll sicherstellen, dass vor dem Test alle Attribute der Klasse initialisiert werden
- Die parametrisierte Testmethode `abs_valuesGreaterAndLessZero_greaterZero(int)` soll den nachfolgenden Testfall abdecken:
    - Zu testende Methode: `abs(int)`
    - Eingaben: Zahl kleiner Null, Zahl gleich Null, Zahl größer Null
    - Erwartetes Ergebnis: Zahl größer Null
- Die Testmethode `subtract_operand2GreaterOperand1_lessZero()` soll den nachfolgenden Testfall abdecken:
    - Zu testende Methode: `subtract(int, int)`
    - Eingabe: die zweite eingehende Zahl ist größer als die erste eingehende Zahl
    - Erwartetes Ergebnis: Zahl kleiner Null
- Die Testmethode `multiply_withZero_zero()` soll den nachfolgenden Testfall abdecken:
    - Zu testende Methode: `multiply(int, int)`
    - Eingabe: mindestens eine der beiden eingehenden Zahlen ist Null
    - Erwartetes Ergebnis: Null
- Die Testmethode `divide_byZero_arithmeticException()` soll den nachfolgenden Testfall abdecken:
    - Zu testende Methode: `divide(int, int)`
    - Eingabe: die zweite eingehende Zahl ist Null
    - Erwartetes Ergebnis: `ArithmeticException`