Université Paris Cité

Compléments en Programmation Orientée Objet
TP n° 8 - Généricité, interfaces scellées
(Correction)

Exercice 1 : Implémentation de `Result<T, E>`

La gestion explicite des erreurs améliore la clarté du code en précisant de manière explicite les points où des erreurs peuvent se produire. Elle facilite la compréhension du flux d'exécution du programme en permettant aux développeurs d'identifier aisément les opérations pouvant échouer et de traiter ces échecs de manière ciblée. En imposant aux développeurs d'adresser activement les erreurs, ce mécanisme réduit le risque d'erreurs silencieuses susceptibles de passer inaperçues. Ainsi, chaque éventualité d'erreur est clairement définie et prise en compte, renforçant la robustesse globale du code. De nombreux langages encouragent ce style de gestion des erreurs, à l'instar de Rust avec son type `Result<T, E>` et de Scala avec son type `Either[T, E]`.

Objectif : Le but de cet exercice est d'implémenter un type `Result<T, E>` en Java, permettant une gestion explicite des erreurs. Bien qu'il soit possible de le réaliser avec l'héritage et le polymorphisme, notre objectif est d'obtenir une implémentation plus propre en utilisant les fonctionnalités récentes de Java, telles que les **`record`** et les interfaces scellées (`sealed`).

Exemple d'utilisation : Pour montrer où nous voulons en venir voici un exemple portant sur la division entière et son cas problématique : la division par zéro. Le type doit être utilisable avec cet exemple :

```java
class Main {
    static Result<Integer, ArithmeticException> deviser(Integer a, Integer b) {
        try {
            return Result.<Integer, ArithmeticException>ok(a / b);
        } catch (ArithmeticException e) {
            return Result.<Integer, ArithmeticException>err(new ArithmeticException("Division par 0"));
        }
    }

    public static void main(String[] args) {
        Result<Integer, ArithmeticException> r1 = deviser(4, 2);
        Result<Integer, ArithmeticException> r2 = deviser(1, 0);

        if(r1.isErr()) {
            System.out.println("r1 a échoué avec " + r1.getErr());
        } else if(r1.isOk()) {
            System.out.println("r1 a réussi avec " + r1.getValue());
        }
        if(r2.isErr()) {
            System.out.println("r2 a échoué avec " + r2.getErr());
        } else if(r1.isOk()) {
            System.out.println("r2 a réussi avec " + r2.getValue());
        }
    }
}
```

L'éxecution de ce code doit produire l'output suivant :

```
r1 a réussi avec 2
r2 a échoué avec java.lang.ArithmeticException: Division par 0
```

À faire : Écriver l'interface `Result<T, E>` ainsi que les **`record`** : `Ok<T, E>` et `Err<T, E>` tel que :

1. Votre implémentation fonctionne avec le code exemple en haut ;
2. L'interface `Result<T, E>` ne peut pas être implémentée par des classes définies par l'utilisateur ;
3. Assurez-vous que le type paramétrique `E` est toujours un sous-type de `Exception` ;
4. Dans les cas où les appels à `getErr()` et `getValue()` ne sont pas définis, ils jettent une exception `IllegalStateException`.

Correction :

```java
sealed interface Result<T, E extends Exception> permits Ok, Err {
    public boolean isErr();
    public boolean isOk();

    public Exception getErr();
    public T getValue();

    public static <G, H extends Exception> Result<G, H> ok(G value) {
        return new Ok<G, H>(value);
    }

    public static <G, H extends Exception> Result<G, H> err(H exception) {
        return new Err<G, H>(exception);
    }
}

record Ok<T, E extends Exception>(T value) implements Result<T, E> {
    public boolean isErr() {
        return false;
    }
    public boolean isOk() {
        return true;
    }

    public Exception getErr() {
        throw new IllegalStateException();
    }
    public T getValue() {
        return this.value();
    }
}

record Err<T, E extends Exception>(E exception) implements Result<T, E> {
    public boolean isErr() {
        return true;
    }
    public boolean isOk() {
        return false;
    }

    public Exception getErr() {
        return this.exception();
    }
    public T getValue() {
        throw new IllegalStateException();
    }
}
```