

MODÉLER UNE POTENTIELLE

ENVA = 17

```
sealed interface Result<T,E> {
    record Ok<T,E>(T ok) implements Result<T,E> {
        public Ok {
            java.util.Objects.requireNonNull(ok);
        }
    }
    record Err<T,E>(E error) implements Result<T,E> {
        public Err {
            java.util.Objects.requireNonNull(error);
        }
    }
    public static<T> Ok ok(T ok){
        return new Ok(ok);
    }
    public static<E> Err err(E error){
        return new Err(error);
    }
}

record Weapon(String name){
    public Weapon {
        java.util.Objects.requireNonNull(name);
    }
    public static Weapon weapon(String name){
        return new Weapon(name);
    }
}

public class Main
{
    public static Result<Weapon, Exception> mustCarryABow(Weapon w){
        return w.name().equals("bow") ? Result.ok(w) : Result.err(new Exception("bow not carried"));
    }
    public static void main(String[] args) {
        var myWeapon = Weapon.weapon("bow");
        switch (mustCarryABow(myWeapon)){
            case Result.Ok<Weapon, Exception> o -> System.out.println("Cool you carry a ".concat(o.ok().name()));
            case Result.Err<Weapon, Exception> e -> System.out.println("Error:".concat(e.error().getMessage()));
        }
    }
}
```

MODÉLISER UNE ERREUR POTENTIELLE

```
sealed interface Result<T,E> {
    record Ok<T,E>(T ok) implements Result<T,E> {
        public Ok {
            java.util.Objects.requireNonNull(ok);
        }
    }
    record Err<T,E>(E error) implements Result<T,E> {
        public Err {
            java.util.Objects.requireNonNull(error);
        }
    }
    public static<T> Ok ok(T ok){
        return new Ok(ok);
    }
    public static<E> Err err(E error){
        return new Err(error);
    }
}

record Weapon(String name){
    public Weapon {
        java.util.Objects.requireNonNull(name);
    }
    public static Weapon weapon(String name){
        return new Weapon(name);
    }
}

public class Main
{
    public static Result<Weapon, Exception> mustCarryABow(Weapon w){
        return w.name().equals("bow") ? Result.ok(w) : Result.err(new Exception("bow not carried"));
    }
    public static void main(String[] args) {
        var myWeapon = Weapon.weapon("bow");
        switch (mustCarryABow(myWeapon)){
            case Result.Ok<Weapon, Exception> o -> System.out.println("Cool you carry a ".concat(o.ok().name()));
            case Result.Err<Weapon, Exception> e -> System.out.println("Error:".concat(e.error().getMessage()));
        }
    }
}
```

MODÉLISER UNE ERREUR POTENTIELLE

EN JAVA

N'existe pas dans la lib standard

Peut être encodé avec les génériques (vous avez vu comment) ... c'est pas compliqué mais il manque quelques fonctions pour être réellement utilisable (pure, map, flatmap, fold)

Si vous voulez faire du Java « PRO » vous pouvez utiliser VAVR

https://www.vavr.io/vavr-docs/#_either ...

Ou passez à Scala ou Kotlin + Arrow-Kt