MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

UNE VALEUR POUR SEMER LA DÉSOLATION : NULL

MIAGE M2 - QUALITÉ DU SI - THOMAS HAESSLÉ & QUENTIN BURG

```
class Weapon {}
class Target {}
class Impacted{}

public class Main {
    static Weapon armYouBow = null;
    static Target targetMonster = null;
    static Impacted hitMonster(Weapon w, Target t) {
        return null;
    }
    public static void main(String[] args) {
     hitMonster(armYouBow, targetMonster).toString();
    }
}
```

OCaml, Rust, Haskell : null n'existe pas

Kotlin, Swift, F# : n'autorisent null que si explicitement autorisé à la déclaration

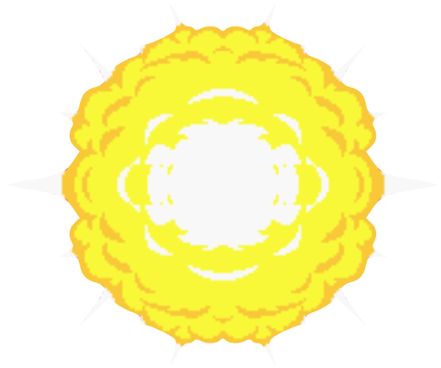
TS (bien configuré) : n'autorise null que si explicitement typé

```
Exception in thread "main" java.lang.NullPointerException
```

🤯 Qui accepterait de travailler avec un langage qui autorise la compilation de ce programme? Sérieusement ?

MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

UNE VALEUR POUR SEMER LA DÉSOLATION : NULL



```
Exception in thread "main" java.lang.NullPointerException
```

```java
class Weapon {}
class Target {}
class Impacted{}

public class Main {
    static Weapon armYouBow = null;
    static Target targetMonster = null;
    static Impacted hitMonster(Weapon w, Target t) {
        return null;
    }
    public static void main(String[] args) {
     hitMonster(armYouBow, targetMonster).toString();
    }
}
```

🤯 Qui accepterait de travailler avec un langage qui autorise la compilation de ce programme? Sérieusement ?

OCaml, Rust, Haskell : null n'existe pas

Kotlin, Swift, F# : n'autorisent null que si explicitement autorisé à la déclaration

TS (bien configuré) : n'autorise null que si explicitement typé

MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

OPTION EN JAVA

```
class Weapon {}
class Target {}
class Impacted{}

sealed interface Option<A> {
   record None<A>() implements Option<A> {}
   record Some<A>(A value) implements Option<A> {
       public Some {
            java.util.Objects.requireNonNull(value);
       }
   }
}
Option<Weapon> armYouBow = new Option.None();
Option<Target> targetMonster = new Option.None();
Option<Impacted> hitMonster(Weapon w, Target t) {
       return new Option.None();
}
```

`Option<A>` est un type « OU »

`None<A>` représente l'absence de valeur pour le type Option<A>. Ce type a exactement 1 valeur pour un ensemble `A` donné.
`Some<A>` représente une valeur de type A, ce type a autant de valeurs que l'ensemble `A`