

MODÉLISER UN ÉVÉNEMENT EN VALEUR

OPTIONAL

MAGNUM2-QUALITÉ D'SI-THO MAS HAE SLÉ & QUENTIN BURG

```
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();  
}
```

```
import java.util.Optional;
```

```
class Weapon {}
```

```
class Target {}
```

```
class Impacted{}
```

```
Optional<Weapon> armYouBow = Optional.empty();
```

```
Optional<Target> targetMonster = Optional.empty();
```

```
Optional<Impacted> hitMonster(Optional<Weapon> w,
```

```
Optional<Target> t) {
```

```
    return Optional.empty();
```

```
}
```

Optional en Java sert à décrire l'existence ou non d'une valeur!

Utiliser null est une faute.

Vous pouvez vous en prémunir en utilisant `java.util.Objects.requireNonNull` dans vos constructeurs

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**

MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

OPTION EN JAVA

```
import java.util.Optional;
class Weapon {}
class Target {}
class Impacted{}

Optional<Weapon> armYouBow = Optional.empty();
Optional<Target> targetMonster = Optional.empty();
Optional<Impacted> hitMonster(Optional<Weapon> w,
Optional<Target> t) {
    return Optional.empty();
}
```

Optional en Java sert à décrire l'existence ou non d'une valeur!

Utiliser null est une faute.

Vous pouvez vous en prémunir en utilisant `java.util.Objects.requireNonNull` dans vos constructeurs

MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

TRAITER LES VALEURS OPTIONNELLES EN JAVA

```
Optional<Impacted> hitMonsterIf(Optional<Weapon> w, Optional<Target> t) {  
    if (w.isPresent() && t.isPresent()) {  
        return Optional.of(new Impacted());  
    } else {  
        return Optional.empty();  
    }  
}
```

Optional n'a pas évolué en sealed interface/record en Java17

La manipulation de Optional est fastidieuse et demande de la vigilance (pas de pattern matching)