MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

TRAITER LES VALEURS OPTIONNELLES

MIAGE M2 - QUALITÉ DU SI - THOMAS HAESSLÉ

```
let hit_monster_pattern_match w t  =
  match w with
   | None -> None
   | Some w -> match t with
     | None -> None
     | Some t -> Some Impacted
```

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

```
let (let*) = Option.bind

let hit_monster_let_star w t =
  let* used = w in
  let* targeted = t in
  Some Impacted
```

L’operateur let* rend plus lisible l’extraction de valeur de l’Option

```
let (>>=) = Option.bind (* Infix operator for bind *)

let hit_monster_dot_free w t =
  w >>= fun _ -> t >>= fun _ -> Some Impacted
```

```
 let hit_monster_bind w t =
  Option.bind w @@ fun _ -> Option.bind t @@ fun _ -> Some
Impacted
```

```
Optional<Impacted> hitMonsterFlatMap(Optional<Weapon> w,
Optional<Target> t) {
        return w.flatMap( sw -> t.map(st -> new Impacted()));
    }
```

MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

TRAITER LES VALEURS OPTIONNELLES

```
Optional<Impacted> hitMonsterFlatMap(Optional<Weapon> w,
Optional<Target> t) {
       return w.flatMap( sw -> t.map(st -> new Impacted()));
    }
```

```
let (let*) = Option.bind

let hit_monster_let_star w t =
  let* used = w in
  let* targeted = t in
  Some Impacted
```

L’operateur let* **rend plus lisible l’extraction de valeur de l’Option**

TAKE AWAY

LES OPTIONS

A utiliser pour modéliser l’absence de valeur

Sécurisant, surtout quand on dispose d’ADT

Facile à manipuler avec syntaxe spécifique
(let -> OCaml, let! -> F#, do notation -> Haskell, for comprehension -> Scala)*
… ou à défaut flatMap / bind

Dans certains langages Option s’appelle Maybe (Scala, Haskell)