





```
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

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

Les patterns matching exhaustifs sont faciles à lire...  
mais vite fastidieux à écrire

**MODÉLISER UN ÉVÉNEMENT EN VALEUR**

TRAINING VALUES OUR PARTNERS

MAGNET 2-DUALS - TRANSLES

```
let (let*) = Option.bind  
let (let+) x f = Option.map f x
```

```
let hit_monster_let_star w t =  
  let* _used = w in  
  let+ _targeted = t in  
  Impacted
```

Les opérateurs  $\text{let}^*$  /  $\text{let}+$  rendent plus lisible l'extraction de valeur de l'Option



```
let hit_monster_bind w t =  
  Option.bind w @@ fun _ -> Option.map (fun _ -> Impacted) t
```

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

**flatMap (aka bind) et map facilitent la manipulation des valeurs optionnelles et**

**garantissent un bon traitement des cas**

```
let (>>=) = Option.bind (* Infix operator for bind *)
let (>>|) x f = Option.map f x (* Infix operator for map with reverse
parameters *)

let hit_monster_point_free w t =
  w >>= fun _ -> t >>| (fun _ -> Impacted)
```

Les opérateurs infix rendent l'écriture et la lecture plus aisée  
... pour peu qu'on prenne le temps d'apprendre ces  
nouveaux opérateurs

# 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 (let+) x f = Option.map f x  
  
let hit_monster_let_star w t =  
    let* _used = w in  
    let+ _targeted = t in  
    Impacted
```

Les opérateur let\* / let+ rendent 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  
Optional (Java) ou Maybe (Scala, Haskell)

