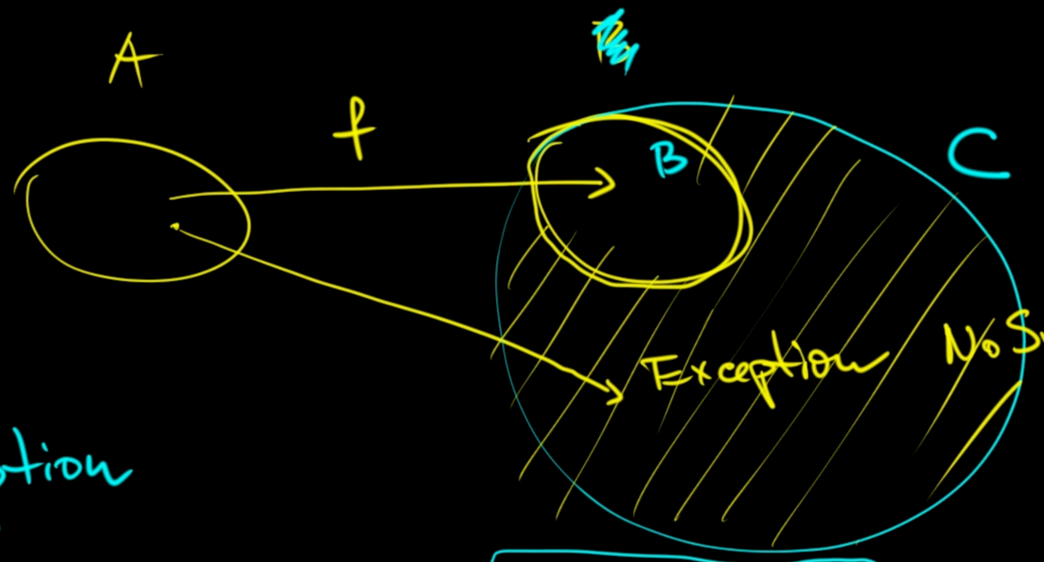
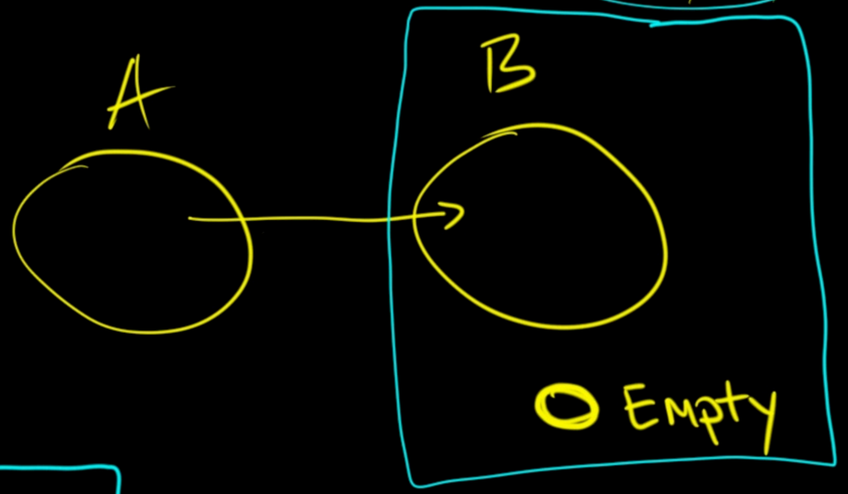
$f : A \rightarrow B$



$f : A \rightarrow B \mid Exception$

$f : A \rightarrow C$



Optional → A
↘ Empty

Option → Some <A>
↘ None

Maybe → Some <A>
↘ None

B | empty

int max(int[])
int first(int[])

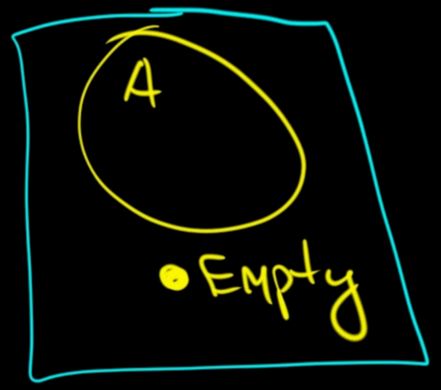
→ Int
↘ "empty"

```
record NonEmptyArray<A>(A head, A[] tail){}
```

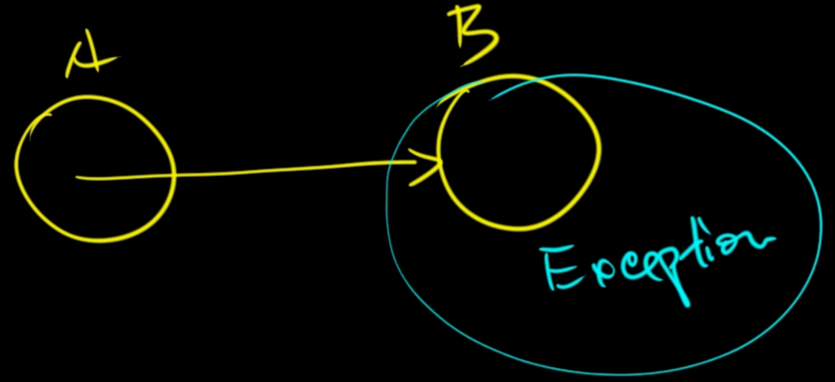
```java
public record NonEmptyArray<A>(A head, A[] tail) implements Iterable<A> {
  @Override
  public Iterator<A> iterator() {
    return new Iterator<>() {
      int index = -1;

      @Override
      public boolean hasNext() {
        return index < tail.length;
      }

      @Override
      public A next() {
        if (index == -1) {
          index++;
          return head;
        } else {
          return tail[index++];
        }
      }
    };
  }
}
```

```java
<A extends Comparable<A>> A min(NonEmptyArray<A> as) {
  A min = as.head();
  for (A a: as) {
    if (a.compareTo(min) < 0) {
      min = a;
    }
  }
  return min;
}
```

Optional<A>



f: A → B | Exception ?

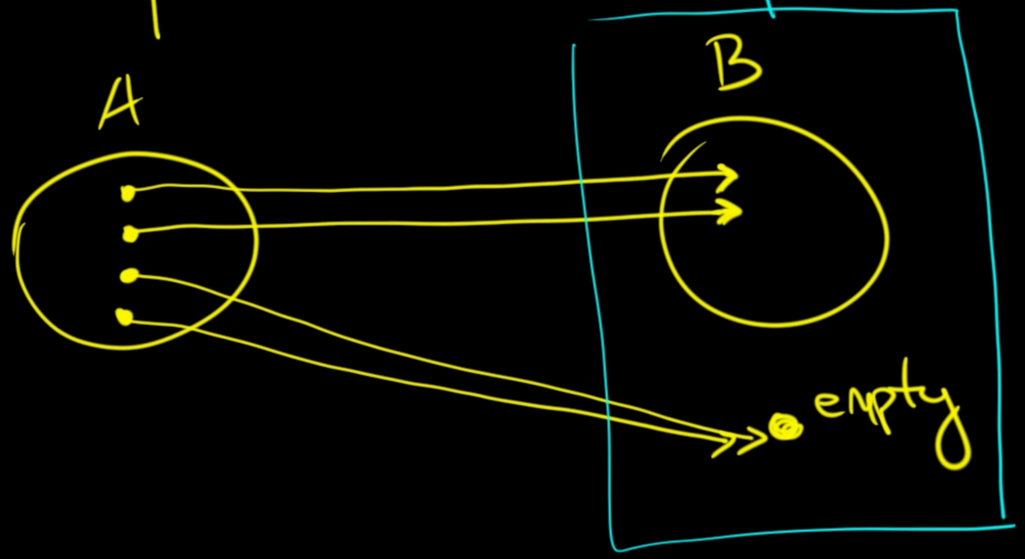


f: + (int, int) → int

f: / (int a, int b) → { a/b
not defined if b=0

A → Option<B>

f: A -> Optional<B> Optional<B> 2014



x : Optional<B>
last
map
first
as.get(idx) → A
→ Exception

Integer.parseInt("abc") → Int
→ Exception

toInt: String -> Int|Except
toIntOpt: String -> Optional<Int>

```
Optional<Integer> first = Stream.of( ...values: 1, 2, 3, 4, 5, 6, 7, 8, 9)
    .filter(x -> x % 13 == 0)
    .findFirst();
```

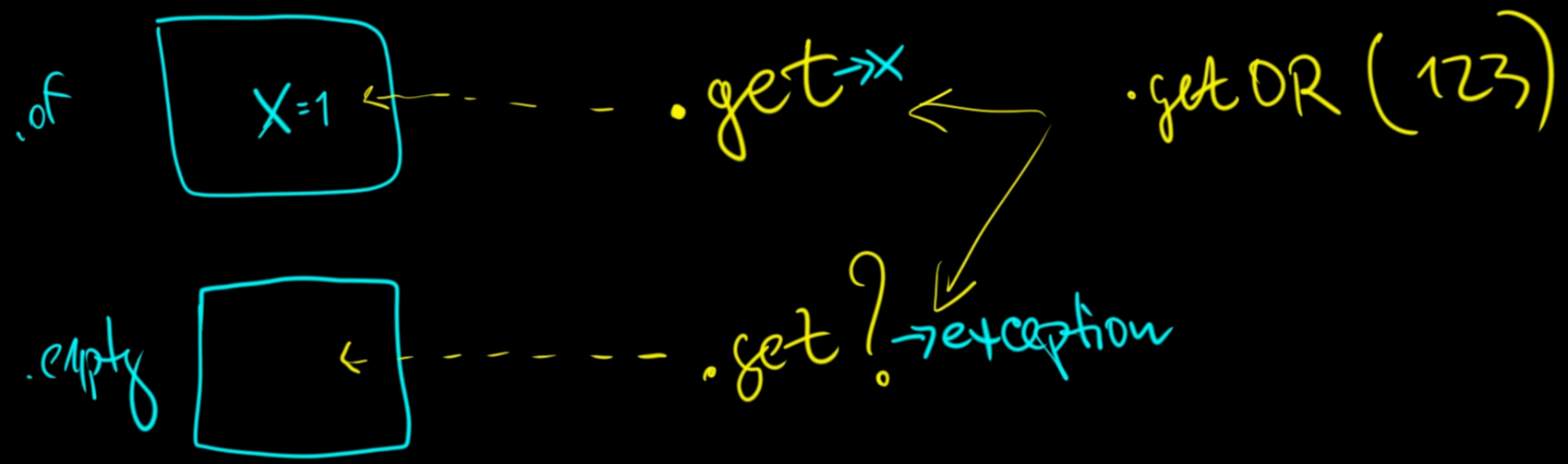
Stream<A>

Stream.of(1)
Stream.empty

≡

Optional.of(1) ≡ [1]
Optional.empty ≡ []

`Optional<Integer>`



```
Optional<Integer> maybeInt = toInt( s: "111");
Integer x = maybeInt.orElse( other: 123);
```

111

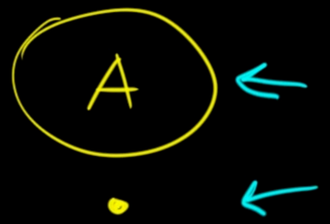
x = 111

```
Optional<Integer> maybeInt = toInt( s: "abc");
Integer x = maybeInt.orElse( other: 123);
```

empty

x = 123

```java
public static Optional<Integer> toInt(String s) {
  try {
    int x = Integer.parseInt(s);
    return Optional.of(x);
  } catch (NumberFormatException e) {
    return Optional.empty();
  }
}
```



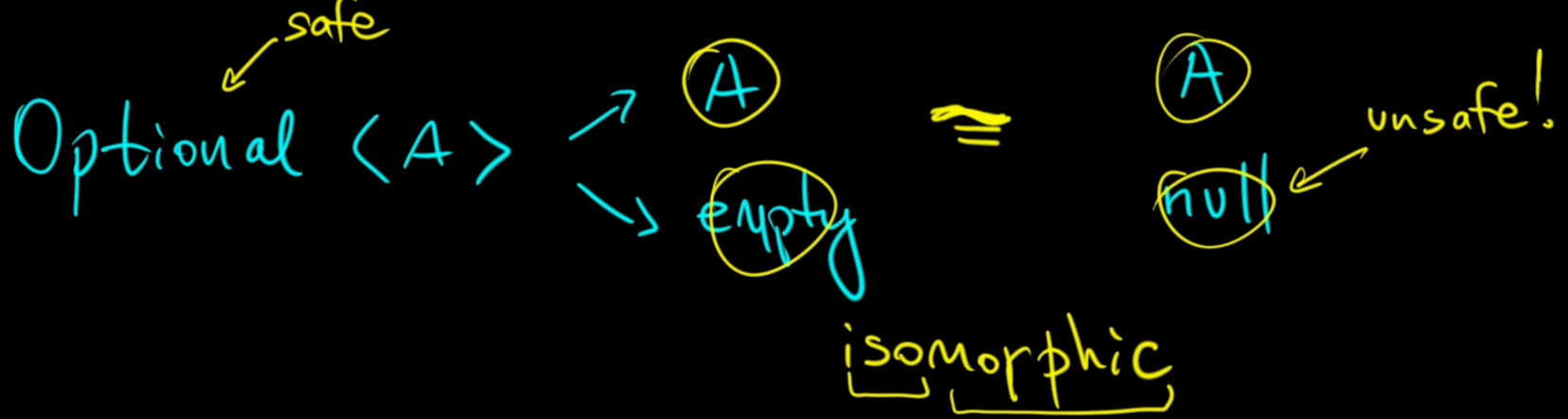
Optional.of(null) → Optional<Object>.of(null)

Object

```java
Optional<String> o = Optional.ofNullable( value: null);
```

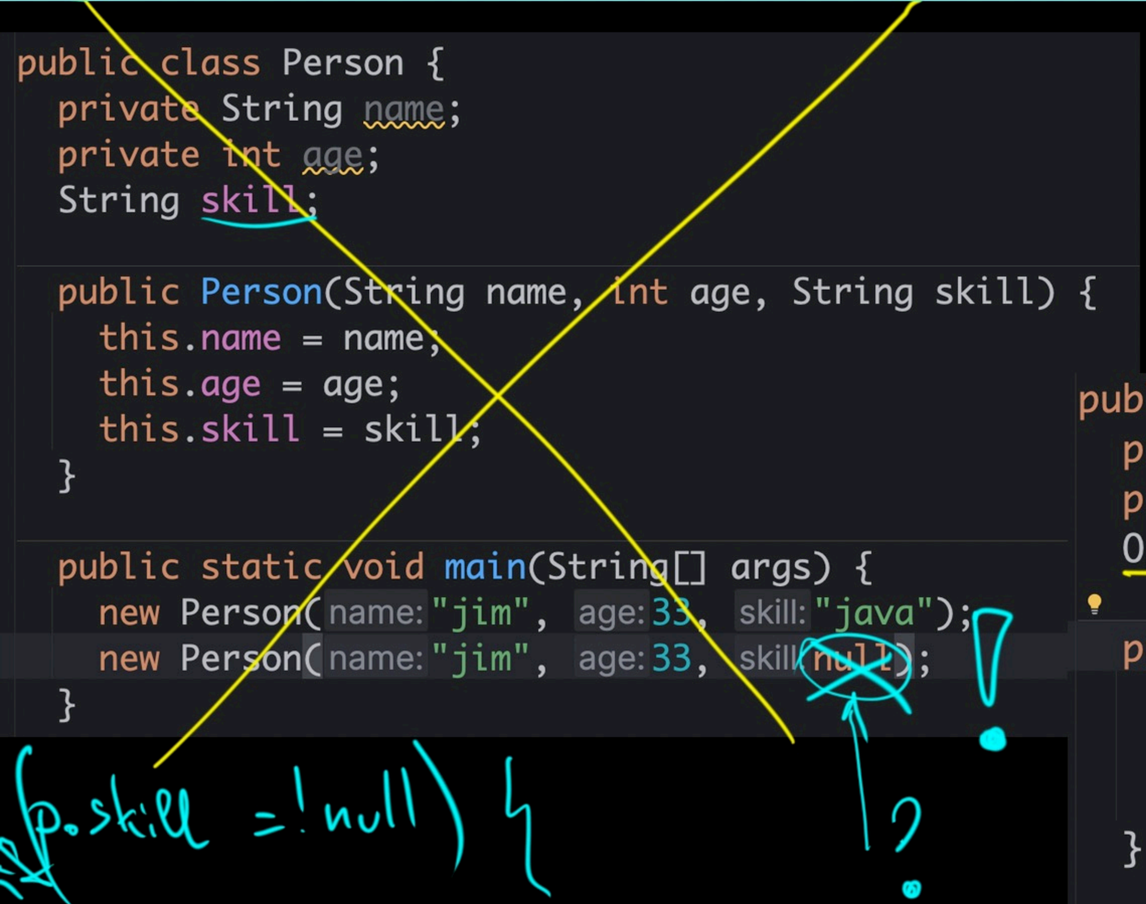
Optional.empty

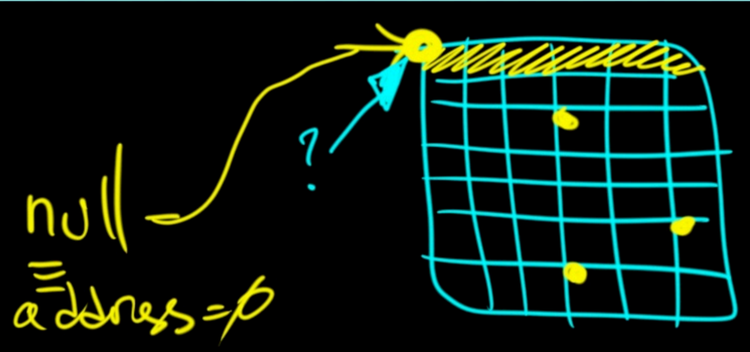
```java
public static <T> Optional<T> ofNullable( @Flow(targetIsContainer = true) T value)
  return value == null ? (Optional<T>) EMPTY
    : new Optional<>(value);
}
```




```
A x = maco A (... ..)
A x = null
```

```java
public class Person {
  private String name;
  private int age;
  String skill;

  public Person(String name, int age, String skill) {
    this.name = name;
    this.age = age;
    this.skill = skill;
  }

  public static void main(String[] args) {
    new Person( name: "jim",  age: 33,  skill: "java");
    new Person( name: "jim",  age: 33,  skill: null);
  }
```



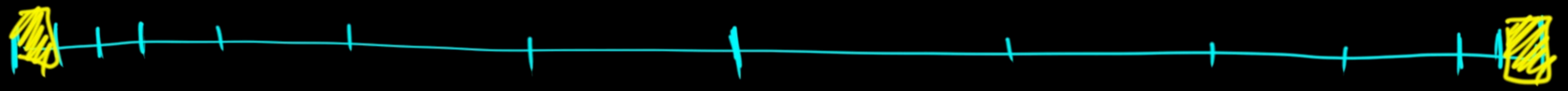



```java
public class Person {
  private String name;
  private int age;
  Optional<String> skill;

  public Person(String name, int age, Optional<String> skill) {
    this.name = name;
    this.age = age;
    this.skill = skill;
  }

  public static void main(String[] args) {
    new Person( name: "jim",  age: 33, Optional.of( value: "java"));
    new Person( name: "jim",  age: 33,  skill: Optional.empty());
  }
```

```java
public class Person {
  private String name;
  private int age;
  Optional<String> skill;

  public Person(String name, int age, Optional<String> skill) {
    this.name = name;
    this.age = age;
    this.skill = skill;
  }

  public static Person of(String name, int age, String skill) {
    return new Person(name, age, Optional.of(skill));
  }

  public static Person noSkill(String name, int age) {
    return new Person(name, age, skill: Optional.empty());
  }

  public static void main(String[] args) {
    new Person(name: "jim", age: 33, Optional.of(value: "java"));
    new Person(name: "jim", age: 33, skill: Optional.empty());
    Person.of(name: "jim", age: 33, skill: "java");
    Person.noSkill(name: "jim", age: 33);
  }
```

bad null

null good



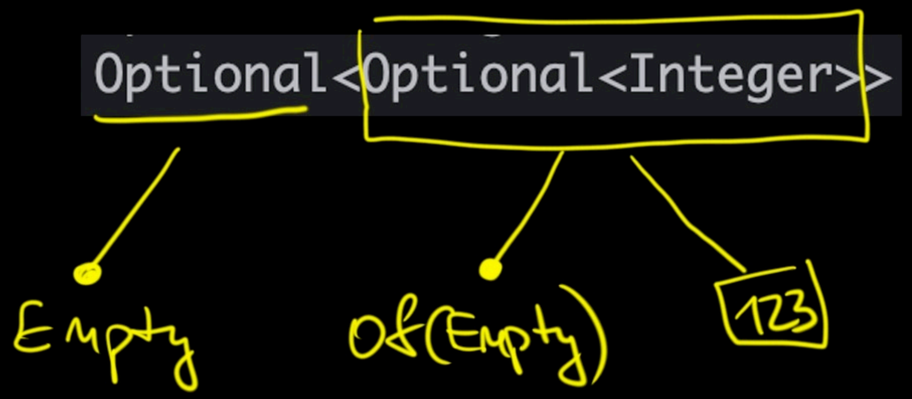
Person x = null;

A → Option<A> 10% A | null

Option<A> → A

Option<A>

```
if (a!=null){
} else {
}
```



1. Empty
2. Of(Empty)
3. Of(Of(x))

```java
Scanner sc = new Scanner(System.in);

System.out.print("Enter x:");
String xs = sc.next();
System.out.print("Enter y:");
String ys = sc.next();

try {
  int x = Integer.parseInt(xs);
  int y = Integer.parseInt(ys);

  int z = x + y;

  String s = String.format("%d + %d = %d", x, y, z);
  System.out.println(s);
} catch (NumberFormatException e) {
  System.out.println("something went wrong");
}
```

```java
Scanner sc = new Scanner(System.in);

System.out.print("Enter x:");
String xs = sc.next();
System.out.print("Enter y:");
String ys = sc.next();

Optional<Integer> maybeX = toInt(xs);
Optional<Integer> maybeY = toInt(ys);

Optional<Integer> maybeSum =
  maybeX
    .flatMap(x ->
      maybeY.map(y -> x + y)
    );

String s = maybeSum.map(z -> String.format("The sum is: %d", z))
  .orElse( other: "something went wrong");

System.out.println(s);
```

Optional<Int>

A | x

1. A → Optional<A>
2. Optional<A> + f → Optional<B>
3. Optional<A> ⇄ void / A / B

Optional<Optional<Int>>

↓

Optional<Int>

A → B | Exception ≅

A → B | null ≅

A ⇒ Optional<B>