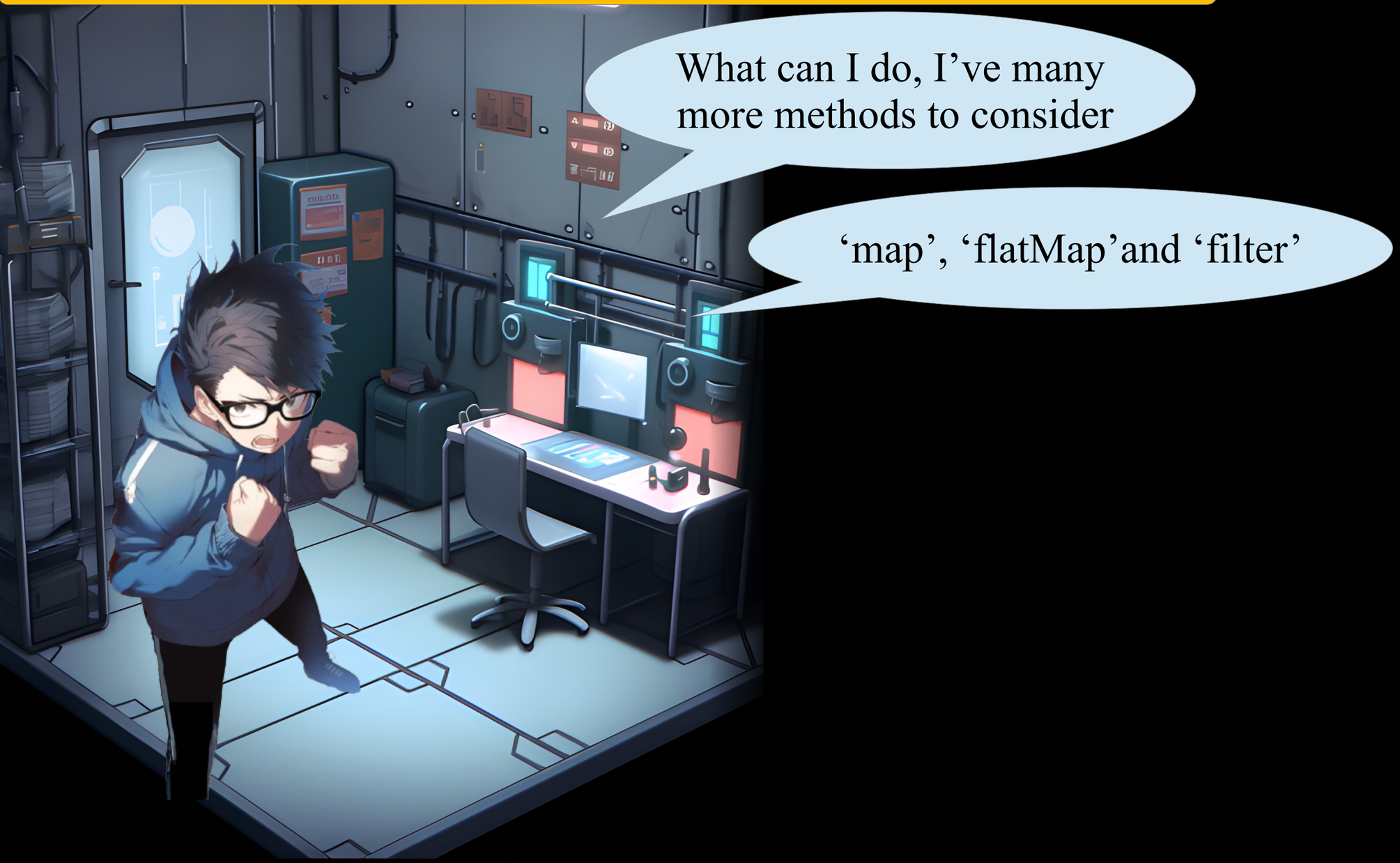

And with this realization, Dany was fully out of power

Now perceiving the surrounding reality again,
Dany is blocked in the room with much more to answer

Now perceiving the surrounding reality again,
Dany is blocked in the room with much more to answer

Now perceiving the surrounding reality again,
Dany is blocked in the room with much more to answer



Now perceiving the surrounding reality again,
Dany is blocked in the room with much more to answer

What can I do, I’ve many more methods to consider

‘map’, ‘flatMap’and ‘filter’

I also have to address the feedback about the singleton

Now perceiving the surrounding reality again,
Dany is blocked in the room with much more to answer

What can I do, I’ve many more methods to consider

‘map’, ‘flatMap’and ‘filter’

I also have to address the feedback about the singleton

Can I do all of this without my power?


Can I do it?
How?


Can I do it?
How?
I can not see
a solution!

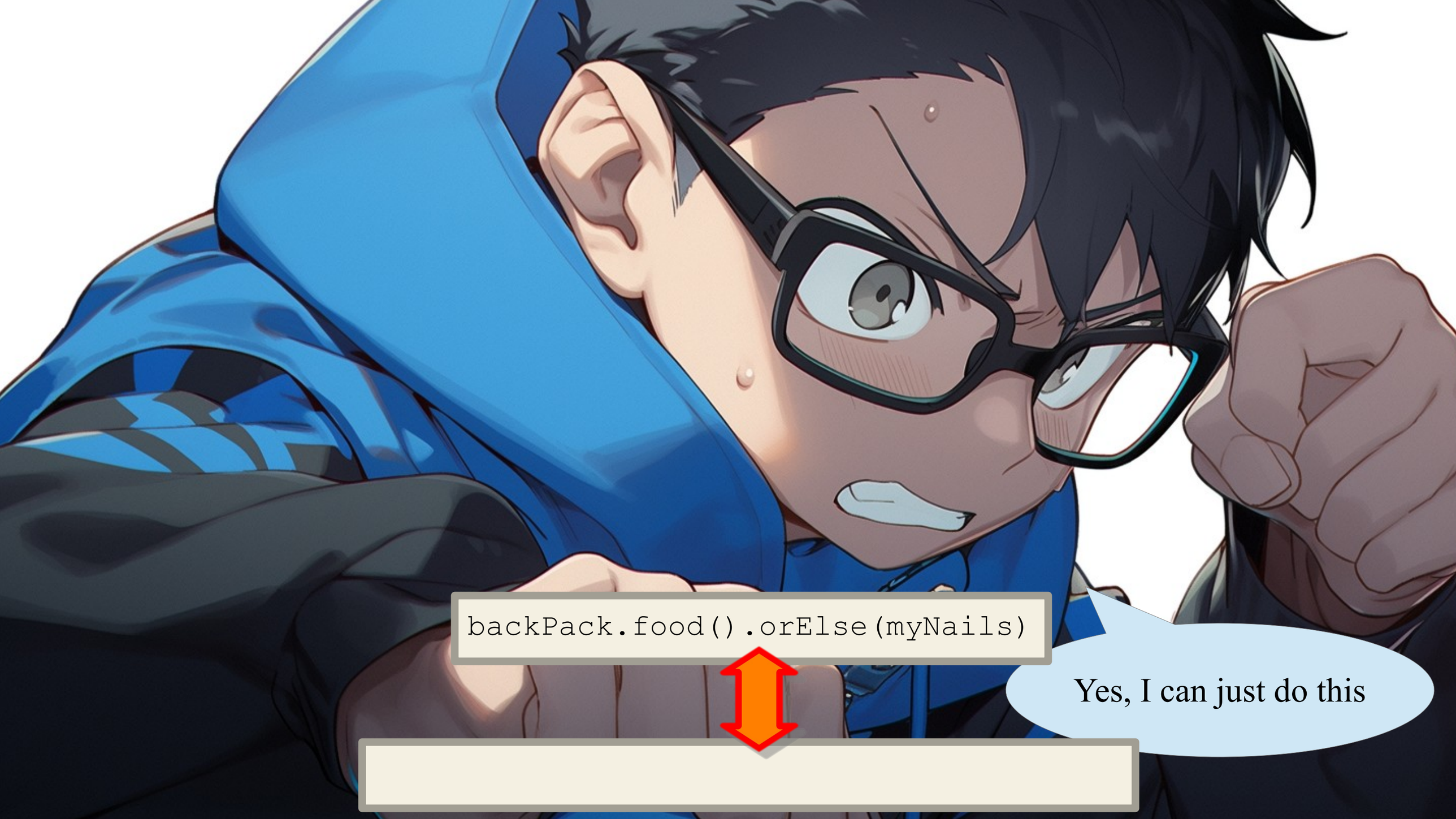

Relax Dany, start simple!

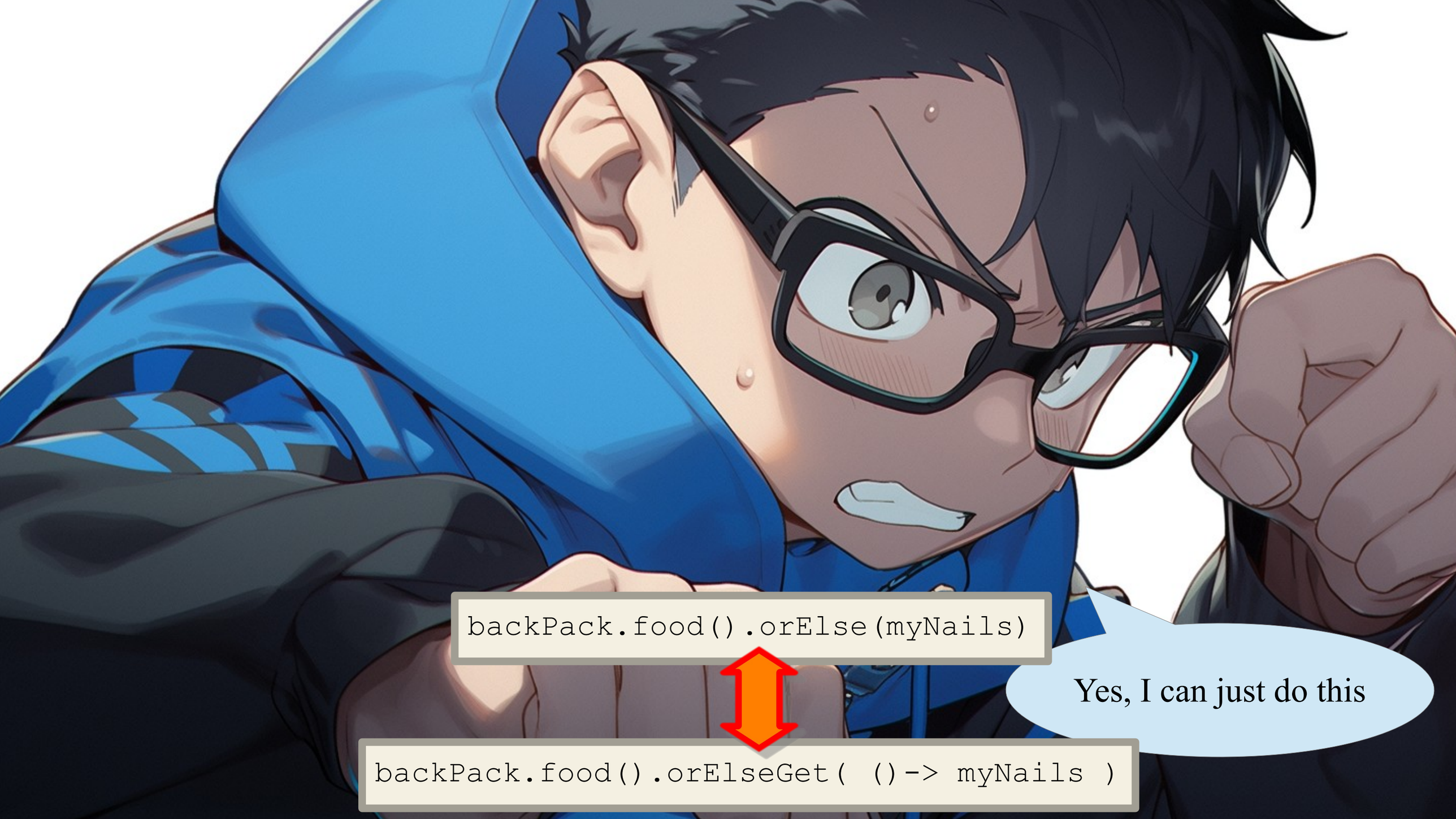

Relax Dany, start simple!
We just envisioned orElseGet and how orElseThrow is sort of derived but not completely

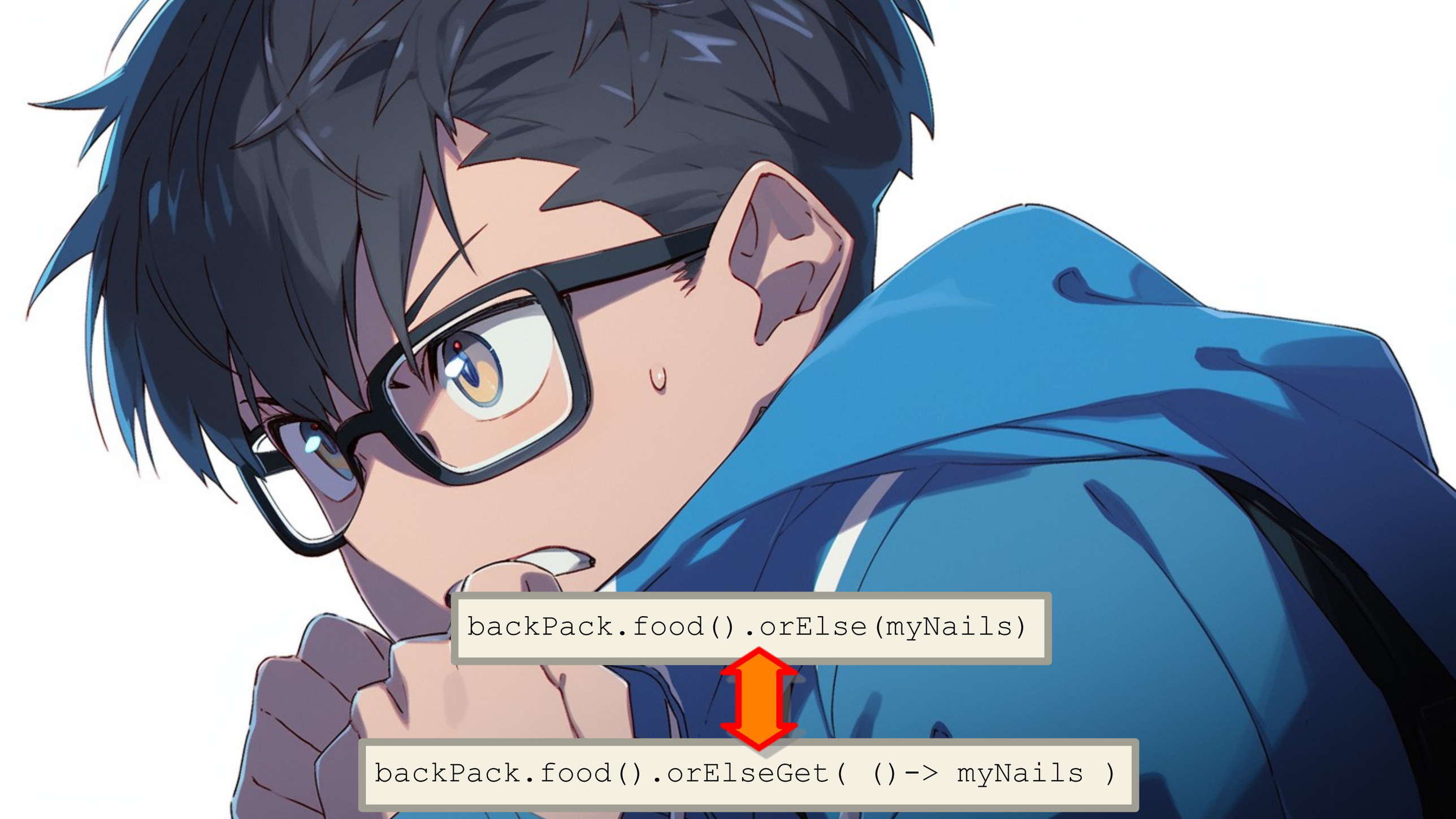

Relax Dany, start simple!
We just envisioned orElseGet and how orElseThrow is sort of derived but not completely
Is the simpler orElse derivable from orElseGet?


Yes, I can just do this


Yes, I can just do this


backPack.food().orElse(myNails)
Yes, I can just do this


backPack.food().orElse(myNails)
backPack.food().orElseGet(()-> myNails)
Yes, I can just do this


backPack.food().orElse(myNails)
backPack.food().orElseGet(()-> myNails)


That is, if I already have the result,
I can turn the result into a supplier
by adding the trebuche '()->'
backPack.food().orElse(myNails)
backPack.food().orElseGet(()-> myNails)


That is, if I already have the result,
I can turn the result into a supplier
by adding the trebuche '()->'
backPack.food().orElse(myNails)
Yes, 'orElse' is
truly redundant
backPack.food().orElseGet(()-> myNails)


Now for the singleton


Now for the singleton
I remember one
of the Pupon chants:

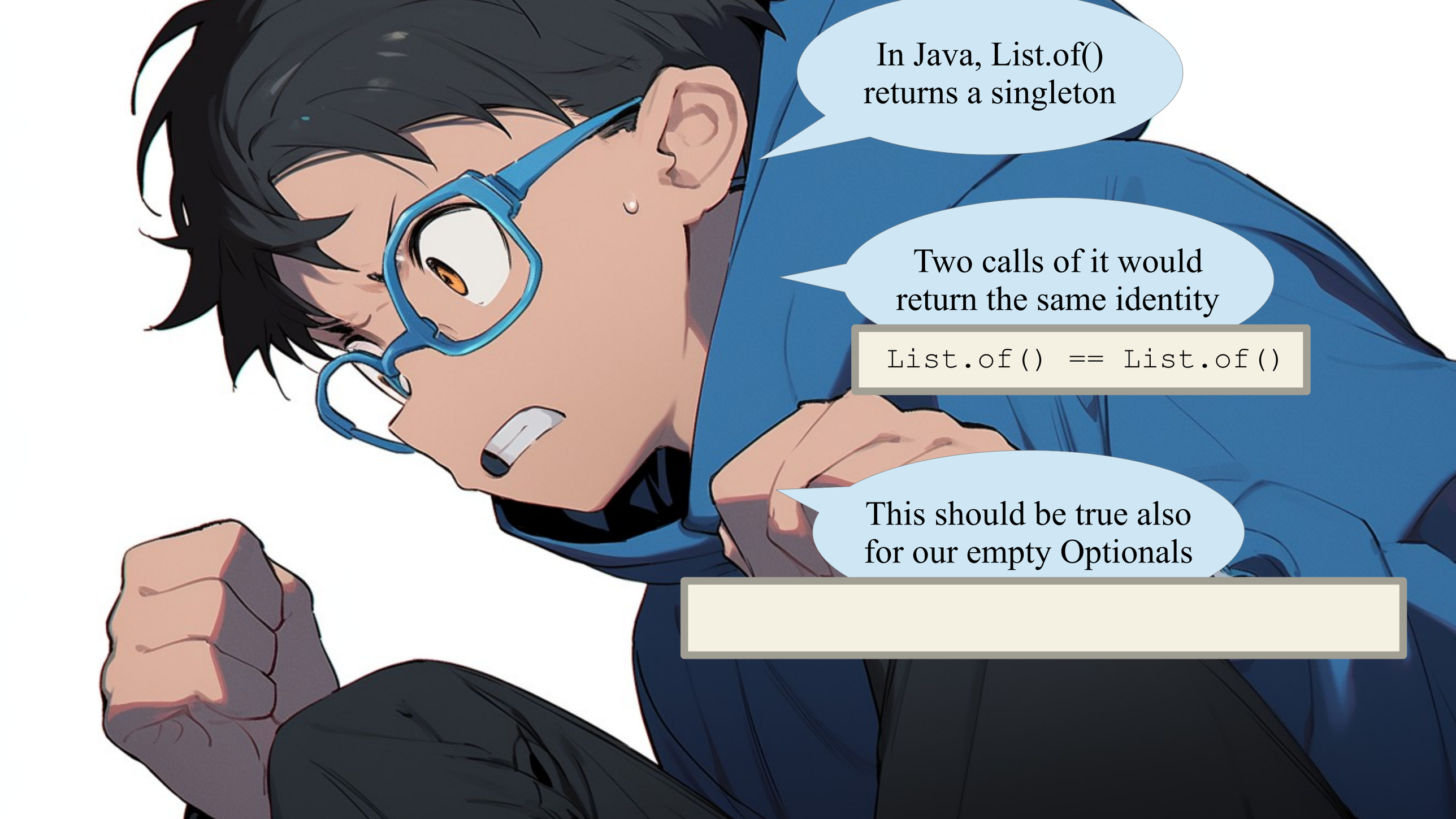

Now for the singleton
I remember one
of the Pupon chants:
“The empty list is a list”

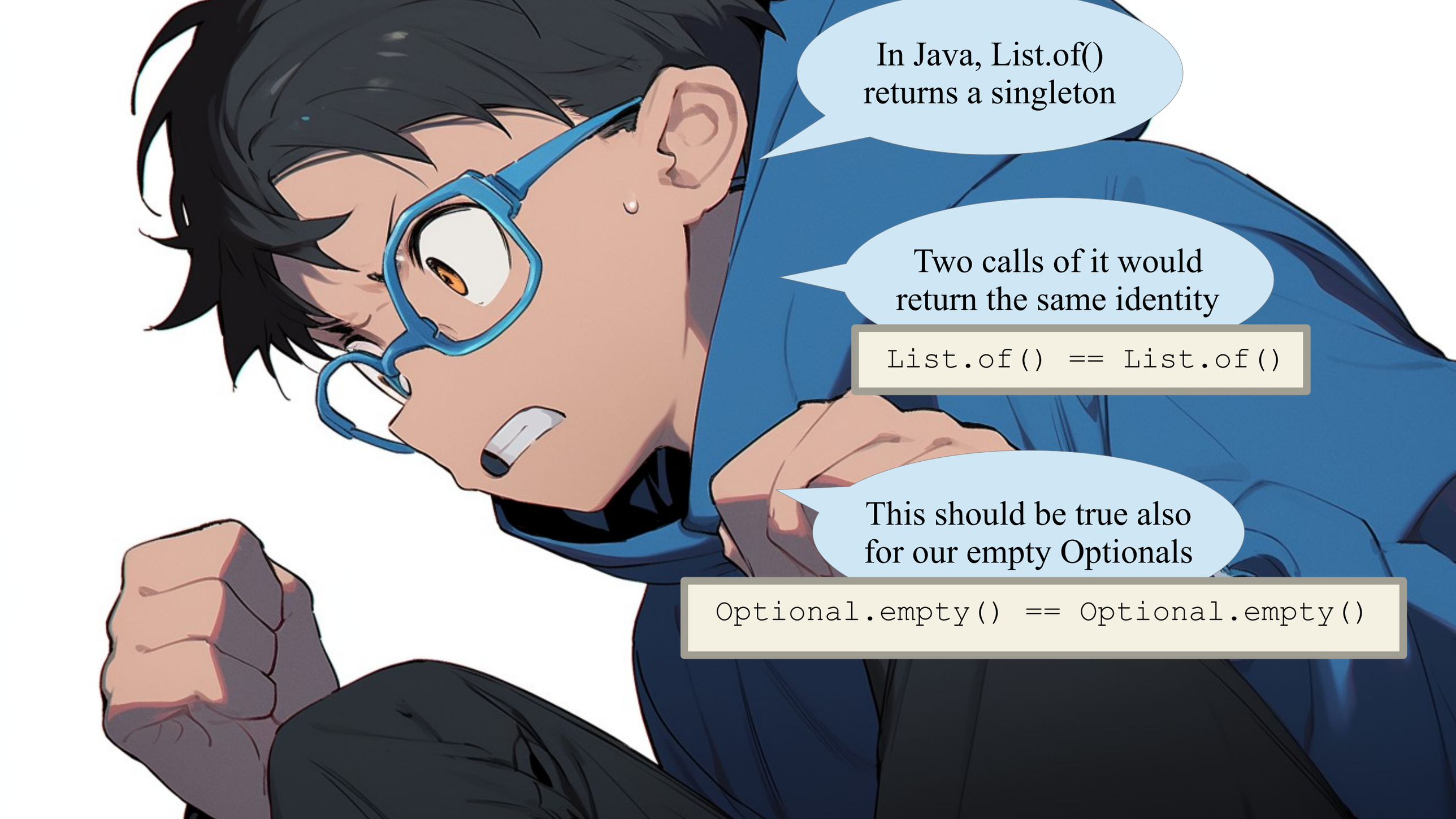

Now for the singleton
I remember one of the Pupon chants:
“The empty list is a list”
“how many empty lists do we need?”


In Java, List.of()
returns a singleton


In Java, List.of()
returns a singleton
Two calls of it would
return the same identity


In Java, List.of()
returns a singleton
Two calls of it would
return the same identity
List.of() == List.of()


In Java, List.of() returns a singleton
Two calls of it would return the same identity
List.of() == List.of()
This should be true also for our empty Optionals


In Java, List.of()
returns a singleton
Two calls of it would
return the same identity
List.of() == List.of()
This should be true also
for our empty Optionals
Optional.empty() == Optional.empty()


But how can I do it?


I can use a static field!

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{

   ...
}
final class Empty<T> implements Optional<T>{


   ...
}
record Some<T>(T get) implements Optional<T>{...}
```


```
public sealed interface Optional<T> permits Empty<T>, Some<T>{

   ...
}
final class Empty<T> implements Optional<T>{


   ...
}
record Some<T>(T get) implements Optional<T>{...}
```


```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
   static <T> Optional<T> empty(){ return Empty.instance; }
   ...
}
final class Empty<T> implements Optional<T>{



   ...
}
record Some<T>(T get) implements Optional<T>{...}
```


```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  static <T> Optional<T> empty(){ return Empty.instance; }
  ...
}
final class Empty<T> implements Optional<T>{
  static final Empty<T> instance= new Empty<T>();

  ...
}
record Some<T>(T get) implements Optional<T>{...}
```


```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
   static <T> Optional<T> empty(){ return Empty.instance; }
   ...
}
final class Empty<T> implements Optional<T>{
   static final Empty<T> instance= new Empty<T>();
   private Empty(){}
   ...
}
record Some<T>(T get) implements Optional<T>{...}
```

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  static <T> Optional<T> empty(){ return Empty.instance; }
  ...
}
final class Empty<T> implements Optional<T>{
  static final Empty<T> instance= new Empty<T>();
  private Empty(){}
  ...
}
record Some<T>(T get) implements Optional<T>{...}
```

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  static <T> Optional<T> empty(){ return Empty.instance;}
  ...
}
final class Empty<T> implements Optional<T>{
  static final Empty<T> instance= new Empty<T>();
  private Empty(){}
  ...
}
record Some<T>(T get) implements Optional<T>{...}
```

empty returns a constant

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
   static <T> Optional<T> empty(){ return Empty.instance;
   ...
}
final class Empty<T> implements Optional<T>{
   static final Empty<T> instance= new Empty<T>();
   private Empty(){}
   ...
}
record Some<T>(T get) implements Optional<T>{...}
```

empty returns a constant

class Empty initializes the constant

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
   static <T> Optional<T> empty(){ return Empty.instance;
   ...
}
final class Empty<T> implements Optional<T>{
   static final Empty<T> instance= new Empty<T>();
   private Empty(){}
   ...
}
re                                ments Optional<T>{...}
```

empty returns a constant

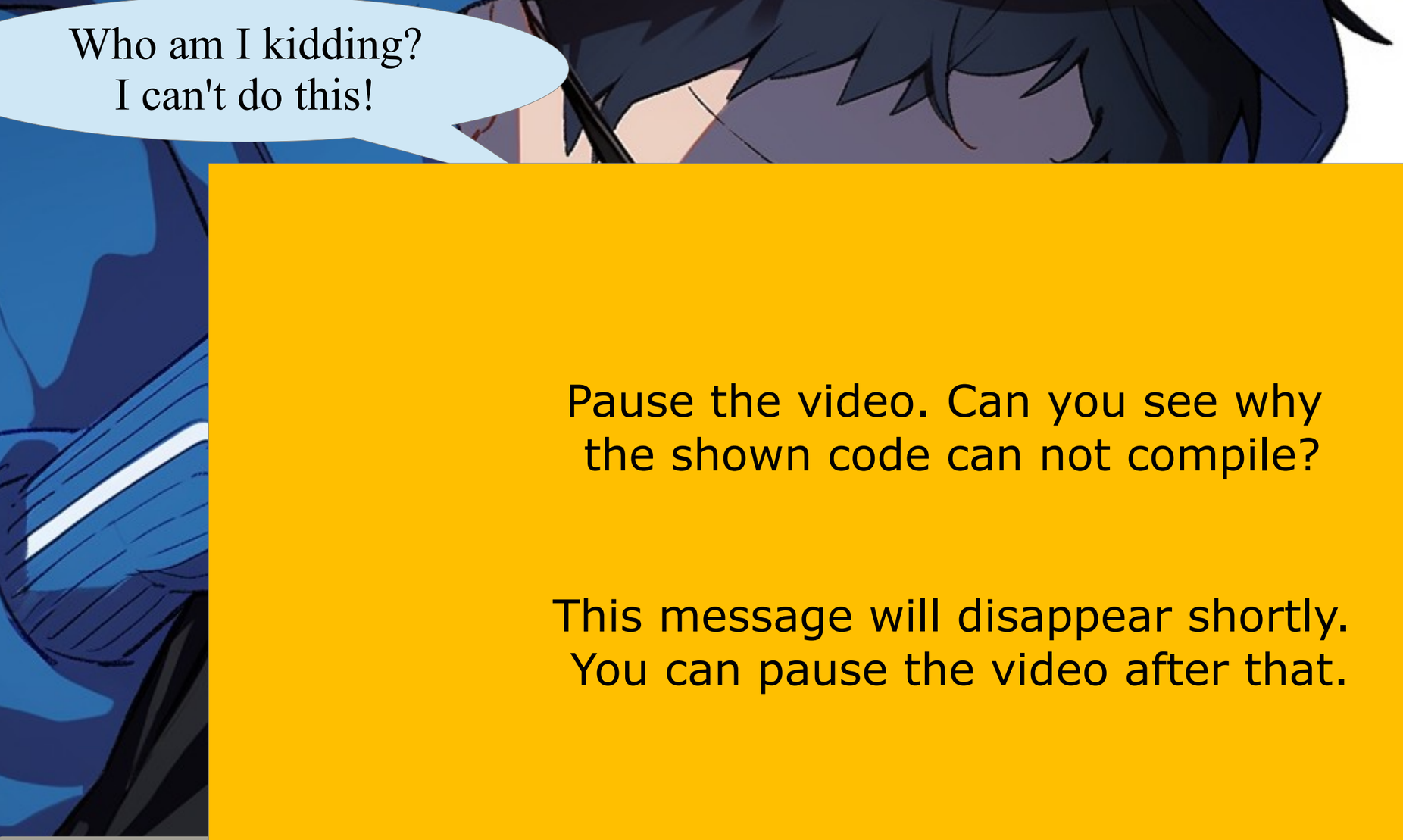
class Empty initializes the constant

And I set the constructor private, so that ‘instance’ is the only instance

```
final class Empty<T> implements Optional<T>{
  static final Empty<T> instance= new Empty<T>();
  private Empty(){}
  ...
}
```


```
final class Empty<T> implements Optional<T>{
  static final Empty<T> instance= new Empty<T>();
  private Empty(){}
  ...
}
```


```
final class Empty<T> implements Optional<T>{
  static final Empty<T> instance= new Empty<T>();
  private Empty(){}
  ...
}
```



Pause the video. Can you see why
the shown code can not compile?

This message will disappear shortly.
You can pause the video after that.

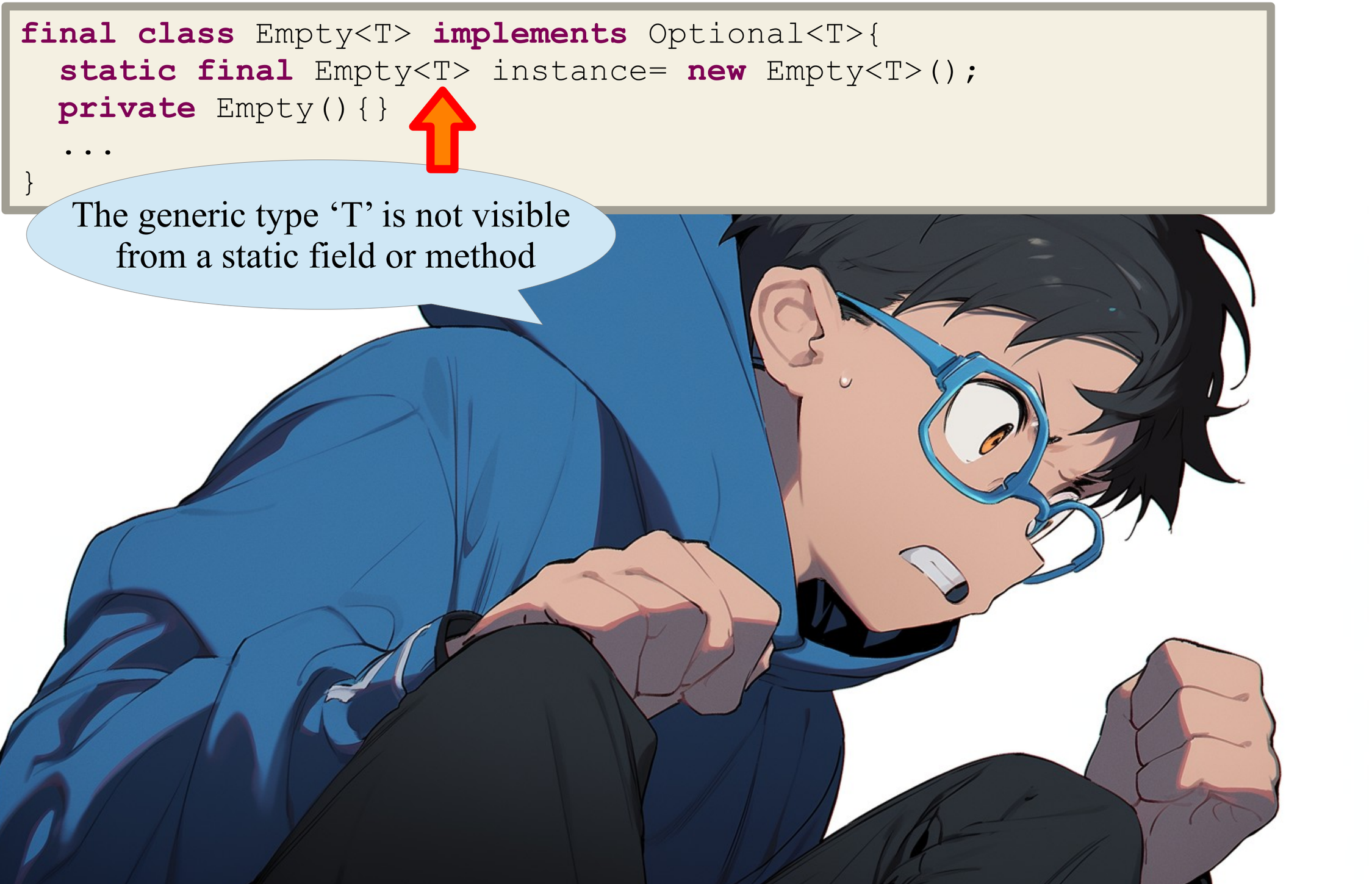
not
ile!

```
final cl
  static final Empty<T> instance= new Empty<T>();
  private Empty(){}
  ...
}
```


```
final class Empty<T> implements Optional<T>{
  static final Empty<T> instance= new Empty<T>();
  private Empty(){}
  ...
}
```

```
final class Empty<T> implements Optional<T>{
   static final Empty<T> instance= new Empty<T>();
   private Empty(){}
   ...
}
```

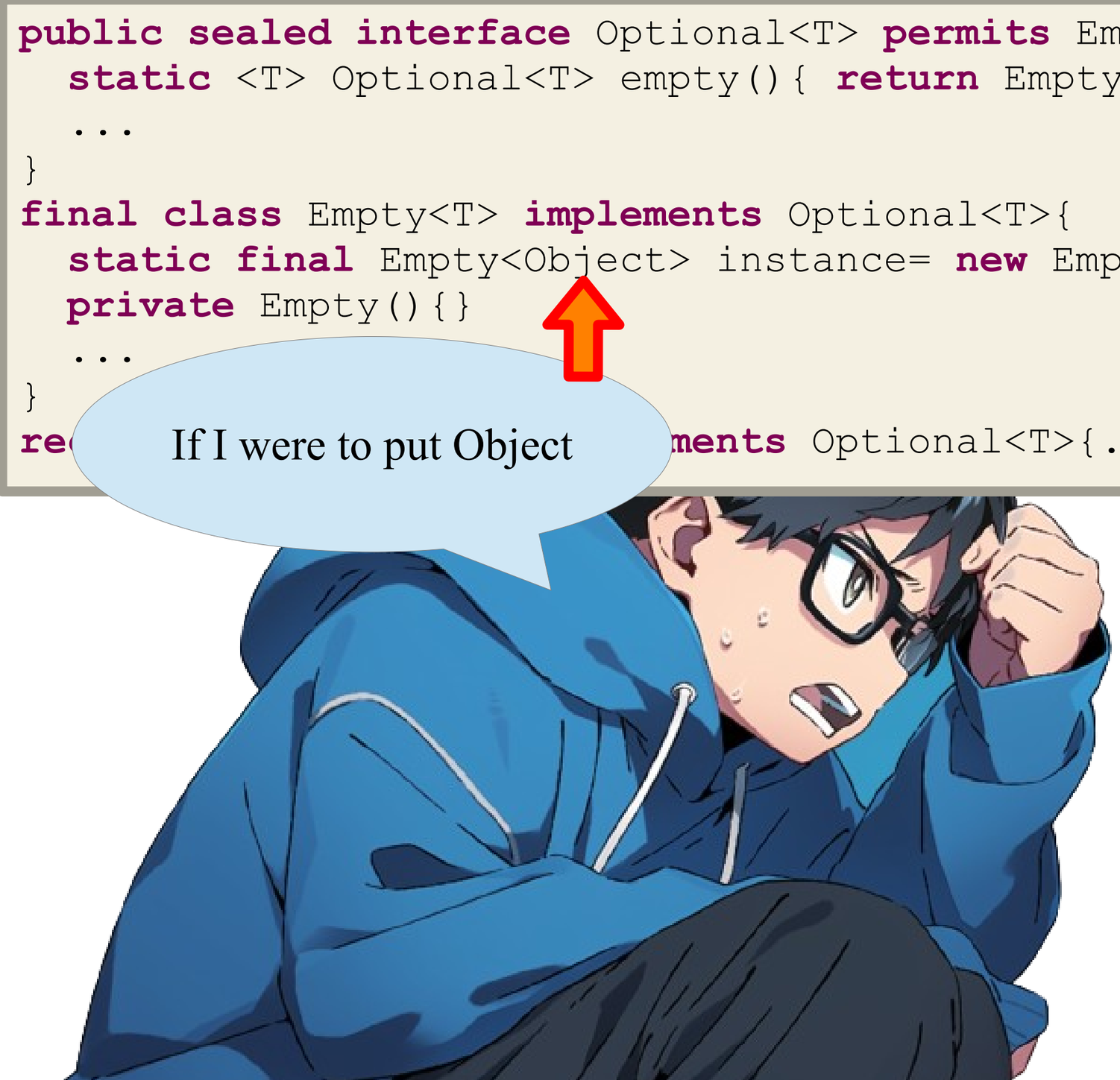
```
final class Empty<T> implements Optional<T>{
   static final Empty<T> instance= new Empty<T>();
   private Empty(){}
   ...
}
```



```
final class Empty<T> implements Optional<T>{
  static final Empty<T> instance= new Empty<T>();
  private Empty(){}
  ...
}
```

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  static <T> Optional<T> empty(){ return Empty.instance; }
  ...
}
final class Empty<T> implements Optional<T>{
  static final Empty<Object> instance= new Empty<>();
  private Empty(){}
  ...
}
record Some<T>(T get) implements Optional<T>{...}
```

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
   static <T> Optional<T> empty(){ return Empty.instance; }
   ...
}
final class Empty<T> implements Optional<T>{
   static final Empty<Object> instance= new Empty<>();
   private Empty(){}
   ...
}
re                                    ments Optional<T>{...}
```



```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
   static <T> Optional<T> empty(){ return Empty.instance;
   ...
}
final class Empty<T> implements Optional<T>{
   static final Empty<Object> instance= new Empty<>();
   private Empty(){}
   ...
}
re                                ments Optional<T>{...}
```

Then this line would not compile

If I were to put Object

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
   static <T> Optional<T> empty(){ return Empty.instance;
   ...
}
final class Empty<T> implements Optional<T>{
   static final Empty<Object> instance= new Empty<>();
   private Empty(){}
   ...
}
re                                ments Optional<T>{...}
```

Then this line would not compile

If I were to put Object

Should I just cast my problem away?


I mean, there is
no ‘T’ instance


I mean, there is no ‘T’ instance
Stored into an empty Optional of T


I mean, there is
no ‘T’ instance
Stored into an empty
Optional of T
So, maybe
the cast is safe?

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  @SuppressWarnings("unchecked")
  static <E> Optional<E> empty(){ return (Optional<E>)Empty.Instance; }
  ...
}
final class Empty<T> implements Optional<T>{
  static final Empty<Object> instance= new Empty<>();
  private Empty(){}
  ...
}
record Some<T>(T get) implements Optional<T>{...}
```

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  @SuppressWarnings("unchecked")
  static <E> Optional<E> empty(){ return (Optional<E>)Empty.Instance; }
  ...
}
final class Empty<T> implements Optional<T>{
  static final Empty<Object> instance= new Empty<>();
  private Empty(){}
  ...
}
record Some<T>(T get) implements Optional<T>{...}
```

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  @SuppressWarnings("unchecked")
  static <E> Optional<E> empty(){ return (Optional<E>)Empty.Instance; }
  ...
}
final class Empty<T> implements Optional<T>{
  static final Empty<Object> instance= new Empty<>();
  private Empty(){}
  ...
}
record Some<T>(T get) implements Optional<T>{...}
```

Here it is

And of course we need the suppress warnings when we cast generics around

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  MAP??
}
final class Empty<T> implements Optional<T>{
  MAP??
}
record Some<T>(T get) implements Optional<T>{
  MAP??
}
```

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  MAP??
}
final class Empty<T> implements Optional<T>{
  MAP??
}
record Some<T>(T get) implements Optional<T>{
  MAP??
}
```

Now I can focus on the other methods.


Optional dot map


Optional dot map
The idea is that we operate on the value without taking it outside of the box

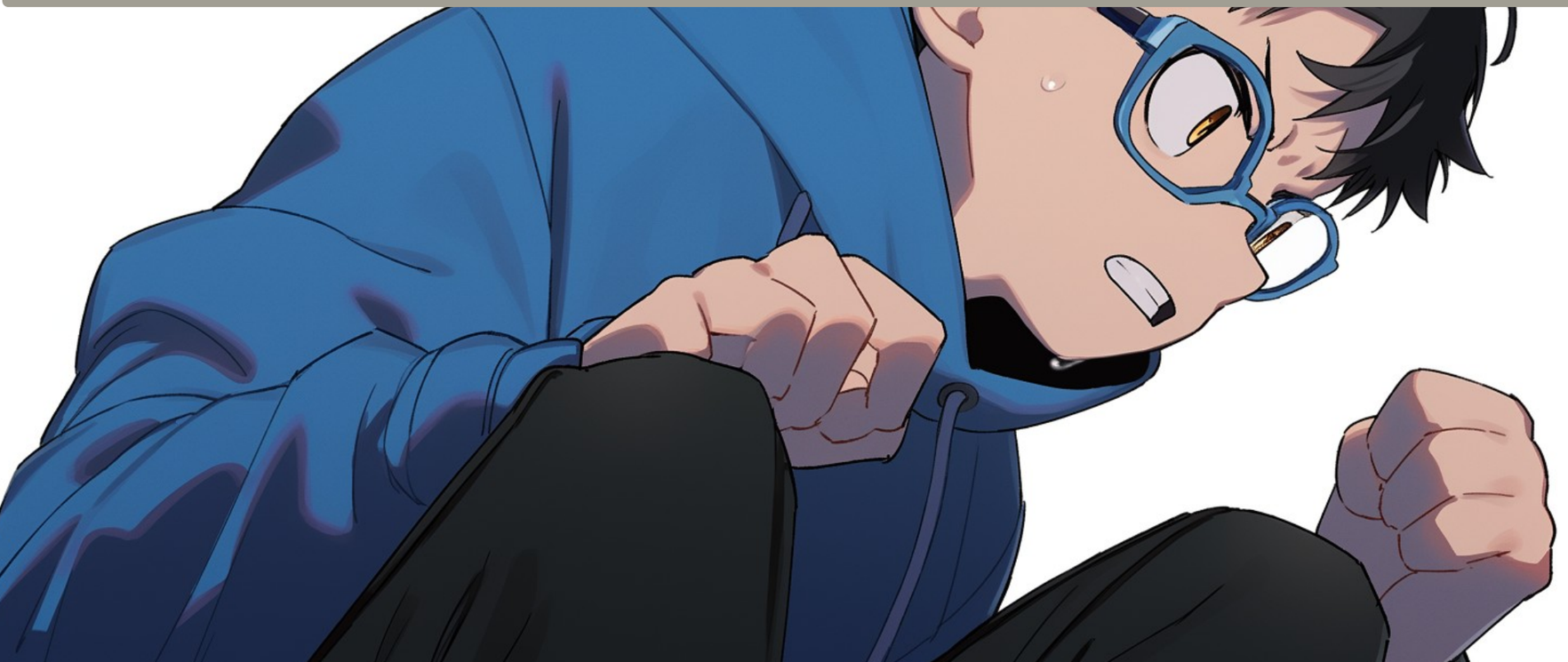

Can I do stuff
about this box

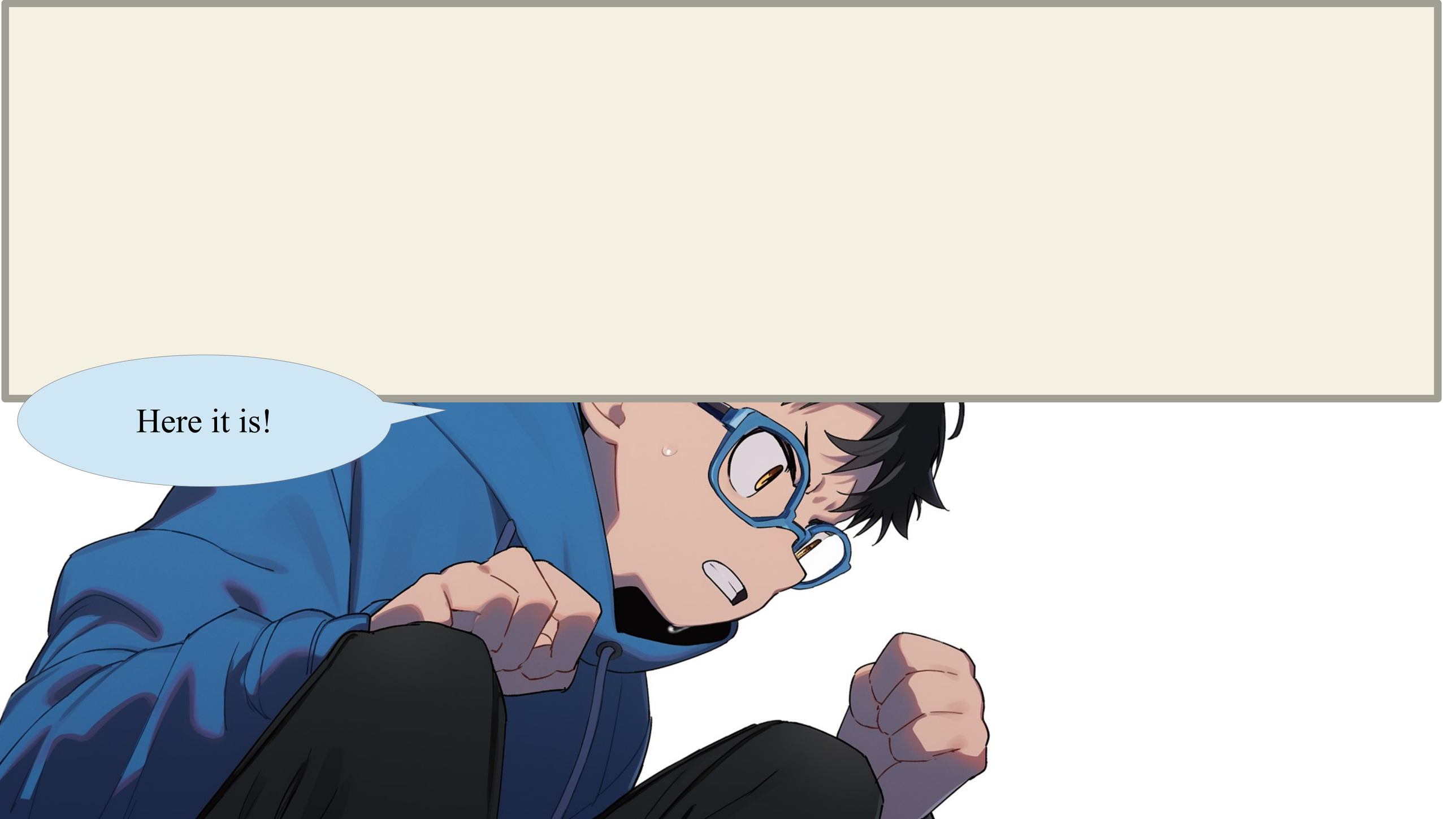

Can I do stuff about this box
Without opening it?


If I can not get the value out...


If I can not get the value out...
can I instead...


Get a lambda in!




Here it is!

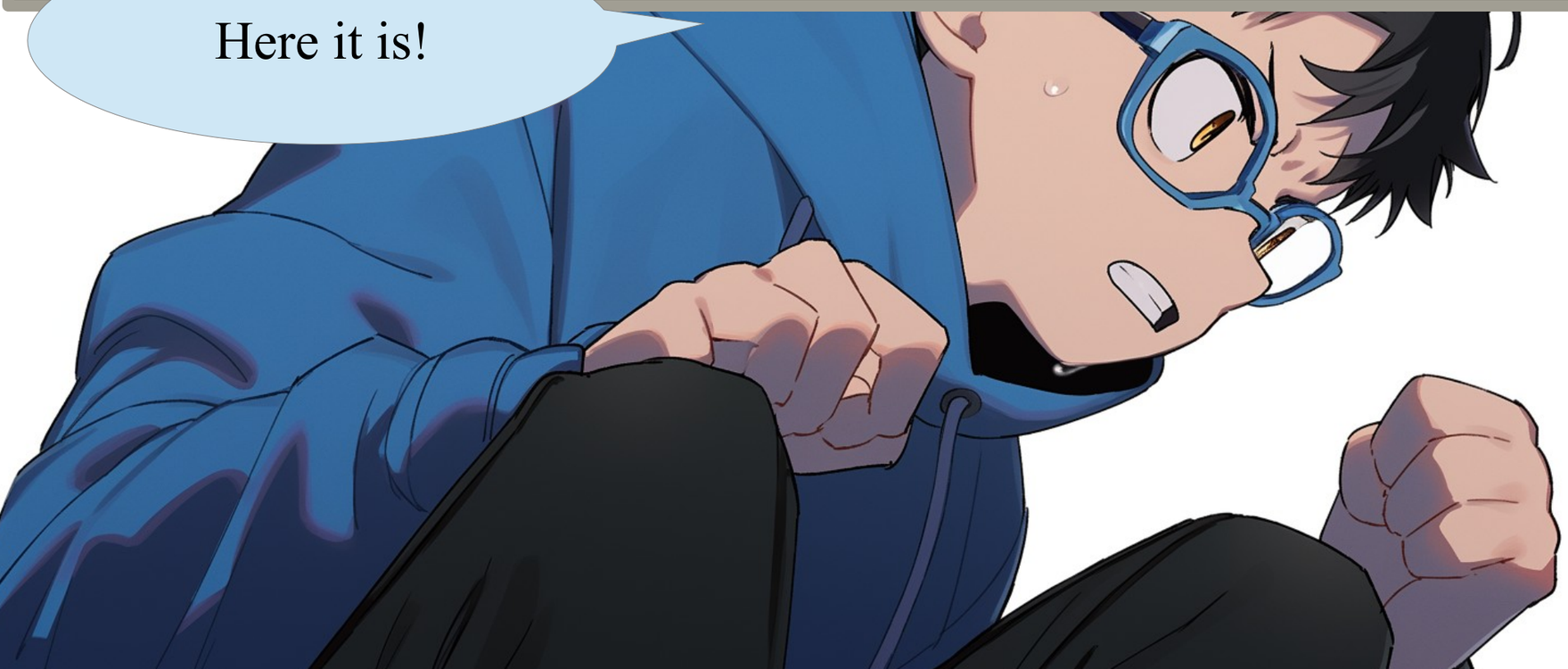
```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
   <U> Optional<U> map(Function<T, U> m);
}
```

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <U> Optional<U> map(Function<T, U> m);
}
final class Empty<T> implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){ return Optional.empty(); }
}
```

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <U> Optional<U> map(Function<T, U> m);
}
final class Empty<T> implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){ return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){
    return Optional.ofNullable(m.apply(get));
  }
}
```

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <U> Optional<U> map(Function<T, U> m);
}
final class Empty<T> implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){ return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){
    return Optional.ofNullable(m.apply(get));
  }
}
```

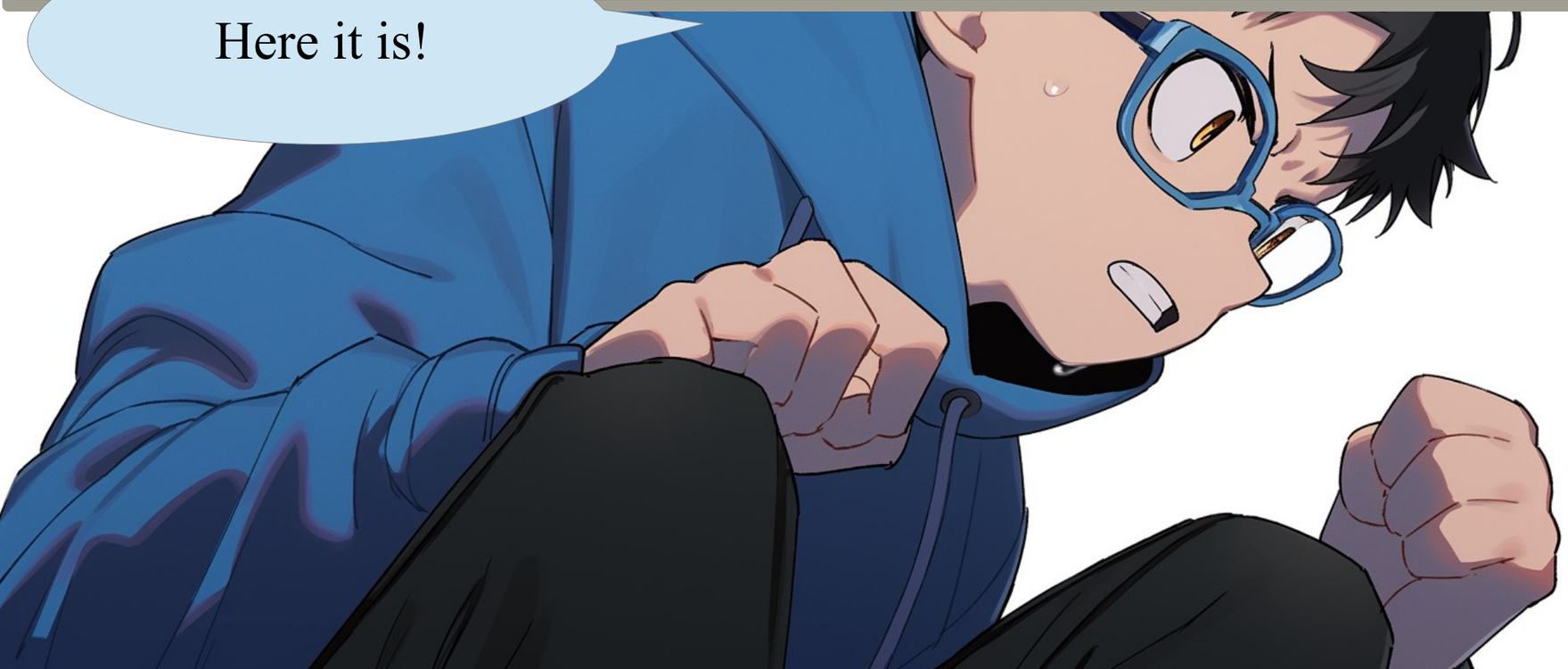
Takes a function from T to U



```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <U> Optional<U> map(Function<T, U> m);
}
final class Empty<T> implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){ return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){
    return Optional.ofNullable(m.apply(get));
  }
}
```

Takes a function from T to U

On empty, returns empty



```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <U> Optional<U> map(Function<T, U> m);
}
final class Empty<T> implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){ return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){
    return Optional.ofNullable(m.apply(get));
  }
}
```

Takes a function from T to U

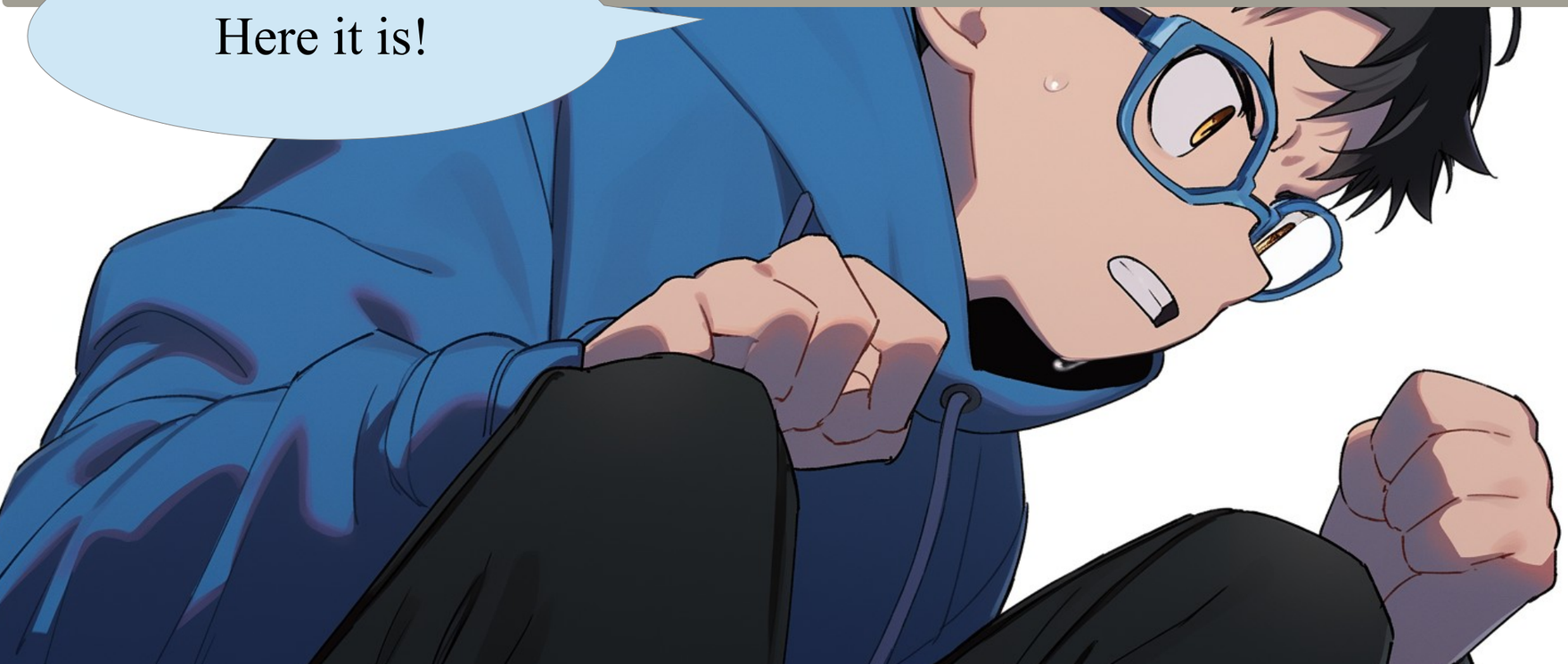
On empty, returns empty

When there is a value, it applies the function and wraps the result in another optional

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <U> Optional<U> map(Function<T, U> m);
}
final class Empty<T> implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){ return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){
    return Optional.ofNullable(m.apply(get));
  }
}
```

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <U> Optional<U> map(Function<T, U> m);
}
final class Empty<T> implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){ return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){
    return Optional.ofNullable(m.apply(get));
  }
}
```

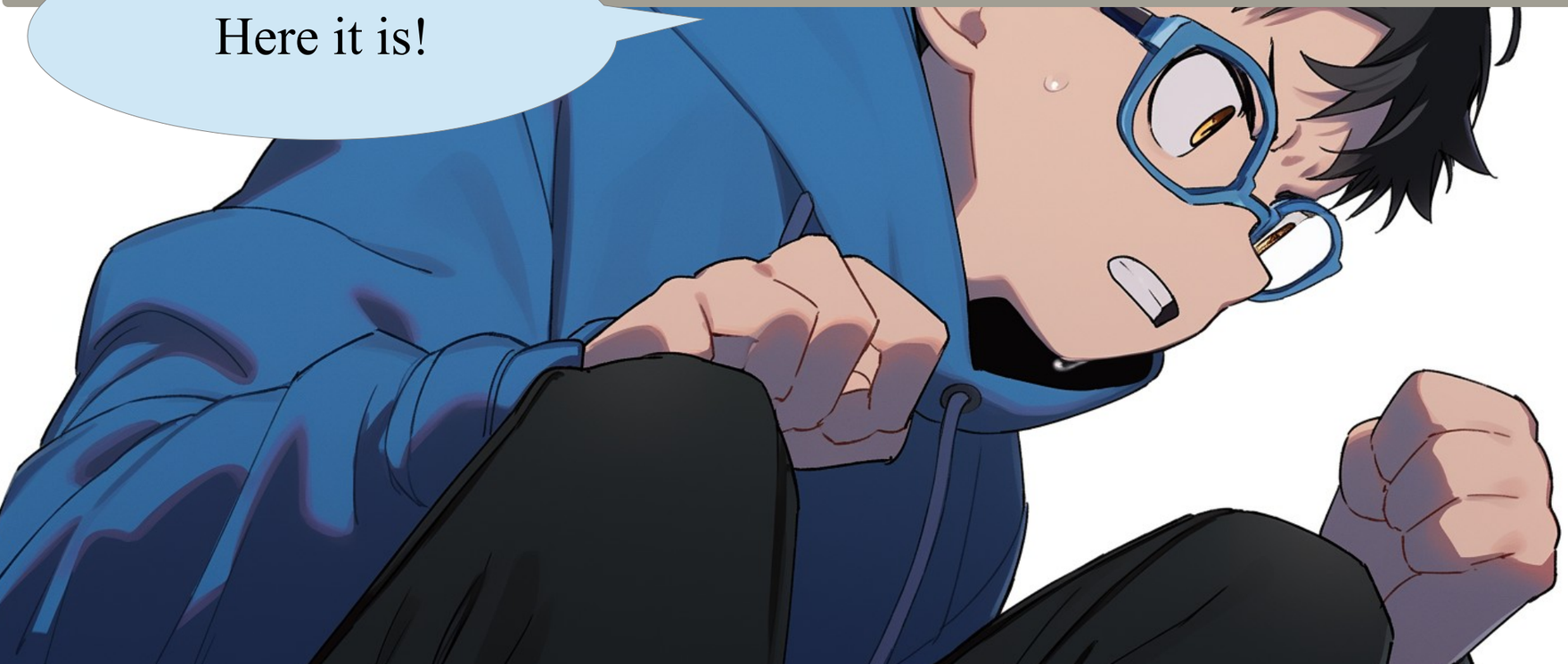
Returns an Optional of U



```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <U> Optional<U> map(Function<T, U> m);
}
final class Empty<T> implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){ return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){
    return Optional.ofNullable(m.apply(get));
  }
}
```

Returns an Optional of U

Empty is just a casted version of ‘this’



```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <U> Optional<U> map(Function<T, U> m);
}
final class Empty<T> implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){ return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <U> Optional<U> map(Function<T, U> m){
    return Optional.ofNullable(m.apply(get));
  }
}
```

Returns an Optional of U

Empty is just a casted version of ‘this’

Here it is!

We use ‘ofNullable’ instead of ‘of’ so that if the map returns null, we have a valid result


I remember this corner case because of Hanton! Personally, I would have just thrown an error in that case.


I remember this corner case because of Hanton! Personally, I would have just thrown an error in that case.


I remember this corner case because of Hanton! Personally, I would have just thrown an error in that case.
Does not need to make sense to you! Just follow the docs


Now filter

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  Optional<T> filter(Predicate<T> p);
}
```

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  Optional<T> filter(Predicate<T> p);
}
final class Empty<T> implements Optional<T>{
  public Optional<T> filter(Predicate<T> p){ return this; }
}
```

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  Optional<T> filter(Predicate<T> p);
}
final class Empty<T> implements Optional<T>{
  public Optional<T> filter(Predicate<T> p){ return this; }
}
record Some<T>(T get) implements Optional<T>{
  public Optional<T> filter(Predicate<T> p){
    if (p.test(get)){ return this; }
    return Optional.empty();
  }
}
```

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  Optional<T> filter(Predicate<T> p);
}
final class Empty<T> implements Optional<T>{
  public Optional<T> filter(Predicate<T> p){ return this; }
}
record Some<T>(T get) implements Optional<T>{
  public Optional<T> filter(Predicate<T> p){
    if (p.test(get)){ return this; }
    return Optional.empty();
  }
}
```

Takes a predicate of T

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  Optional<T> filter(Predicate<T> p);
}
final class Empty<T> implements Optional<T>{
  public Optional<T> filter(Predicate<T> p){ return this; }
}
record Some<T>(T get) implements Optional<T>{
  public Optional<T> filter(Predicate<T> p){
    if (p.test(get)){ return this; }
    return Optional.empty();
  }
}
```

Takes a predicate of T

On an empty ‘this’, returns empty

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  Optional<T> filter(Predicate<T> p);
}
final class Empty<T> implements Optional<T>{
  public Optional<T> filter(Predicate<T> p){ return this; }
}
record Some<T>(T get) implements Optional<T>{
  public Optional<T> filter(Predicate<T> p){
    if (p.test(get)){ return this; }
    return Optional.empty();
  }
}
```

Takes a predicate of T

On an empty ‘this’, returns empty

When there is a value,
return the same optional
if the predicate passes.
Otherwise, empty


Now flatmap

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <R> Optional<R> flatMap(Function<T, Optional<R>> m);
}
```

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <R> Optional<R> flatMap(Function<T, Optional<R>> m);
}
final class Empty<T> implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return Optional.empty(); }
}
```

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <R> Optional<R> flatMap(Function<T, Optional<R>> m);
}
final class Empty<T> implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return m.apply(get);
  }
}
```

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <R> Optional<R> flatMap(Function<T, Optional<R>> m);
}
final class Empty<T> implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return m.apply(get);
  }
}
```

Takes a mapper

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <R> Optional<R> flatMap(Function<T, Optional<R>> m);
}
final class Empty<T> implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return m.apply(get);
  }
}
```

Takes a mapper

On an empty ‘this’, returns empty

```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <R> Optional<R> flatMap(Function<T, Optional<R>> m);
}
final class Empty<T> implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return m.apply(get);
  }
}
```

Takes a mapper

On an empty ‘this’, returns empty

When there is a value,
just apply the mapper

```java
public sealed interface Optional<T> permits Empty<T>, Some<T>{
  <R> Optional<R> flatMap(Function<T, Optional<R>> m);
}
final class Empty<T> implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return Optional.empty(); }
}
record Some<T>(T get) implements Optional<T>{
  public <R> Optional<R> flatMap(Function<T, Optional<R>> m){
    return m.apply(get);
  }
}
```

Takes a mapper

On an empty ‘this’, returns empty

When there is a value,
just apply the mapper

Now flatmap

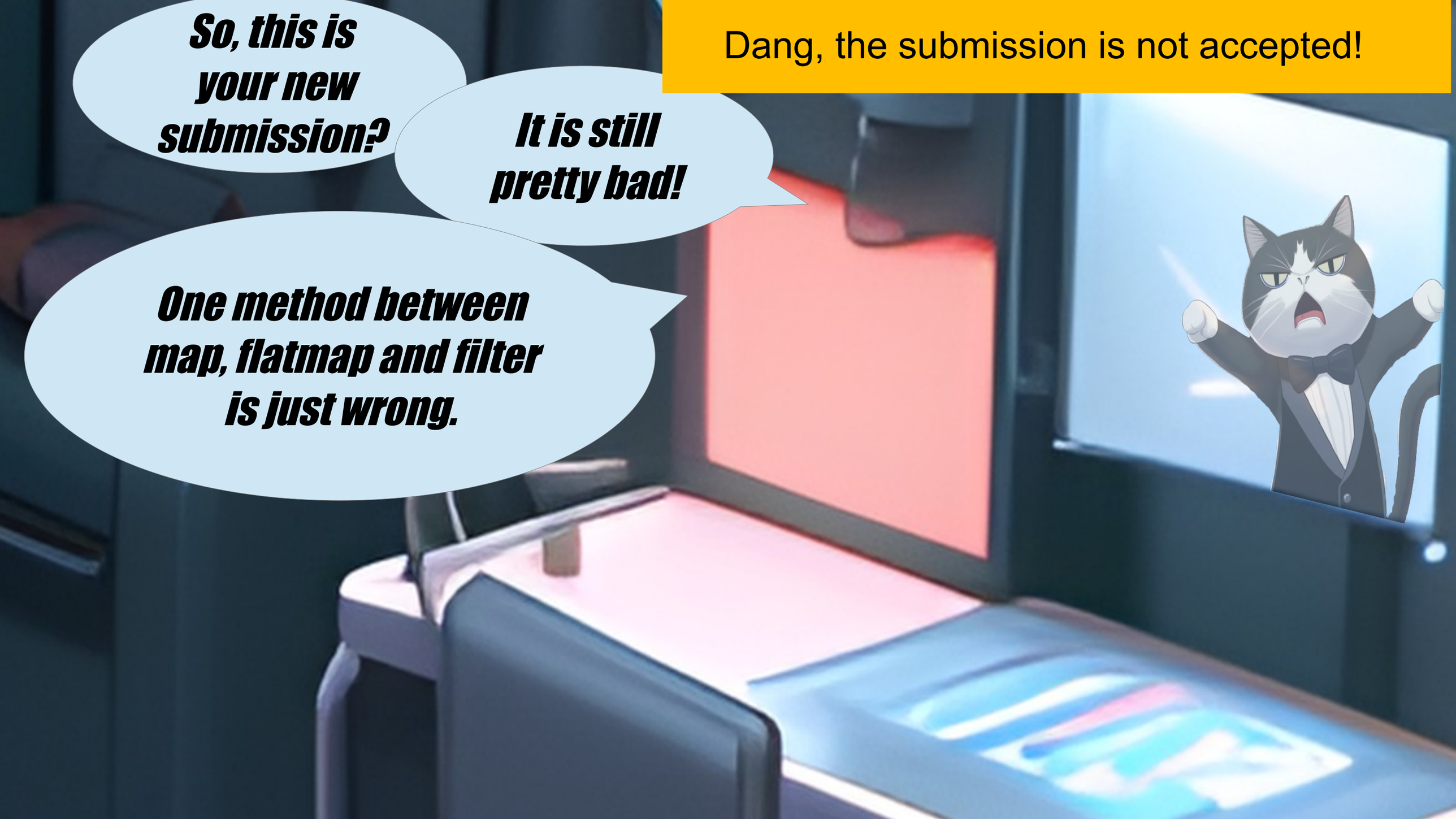
The implementation looks even easier then ‘map’.
This is because the mapper parameter does more work,
as we can see by the bigger type


And now, it is time
to submit again!

Dang, the submission is not accepted!


So, this is your new submission?
Dang, the submission is not accepted!

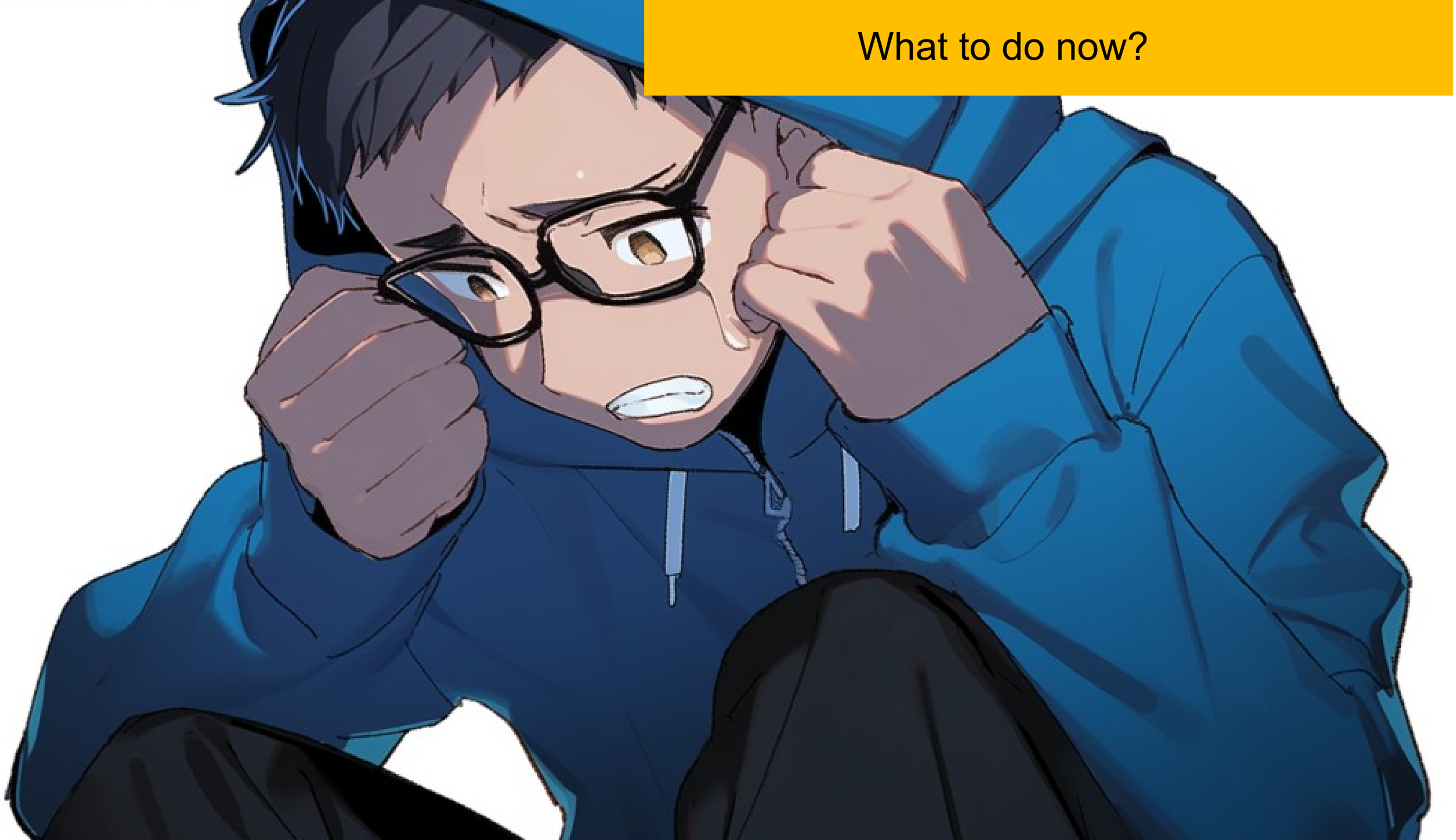

So, this is your new submission?
It is still pretty bad!
Dang, the submission is not accepted!


So, this is your new submission?
It is still pretty bad!
One method between map, flatmap and filter is just wrong.
Dang, the submission is not accepted!

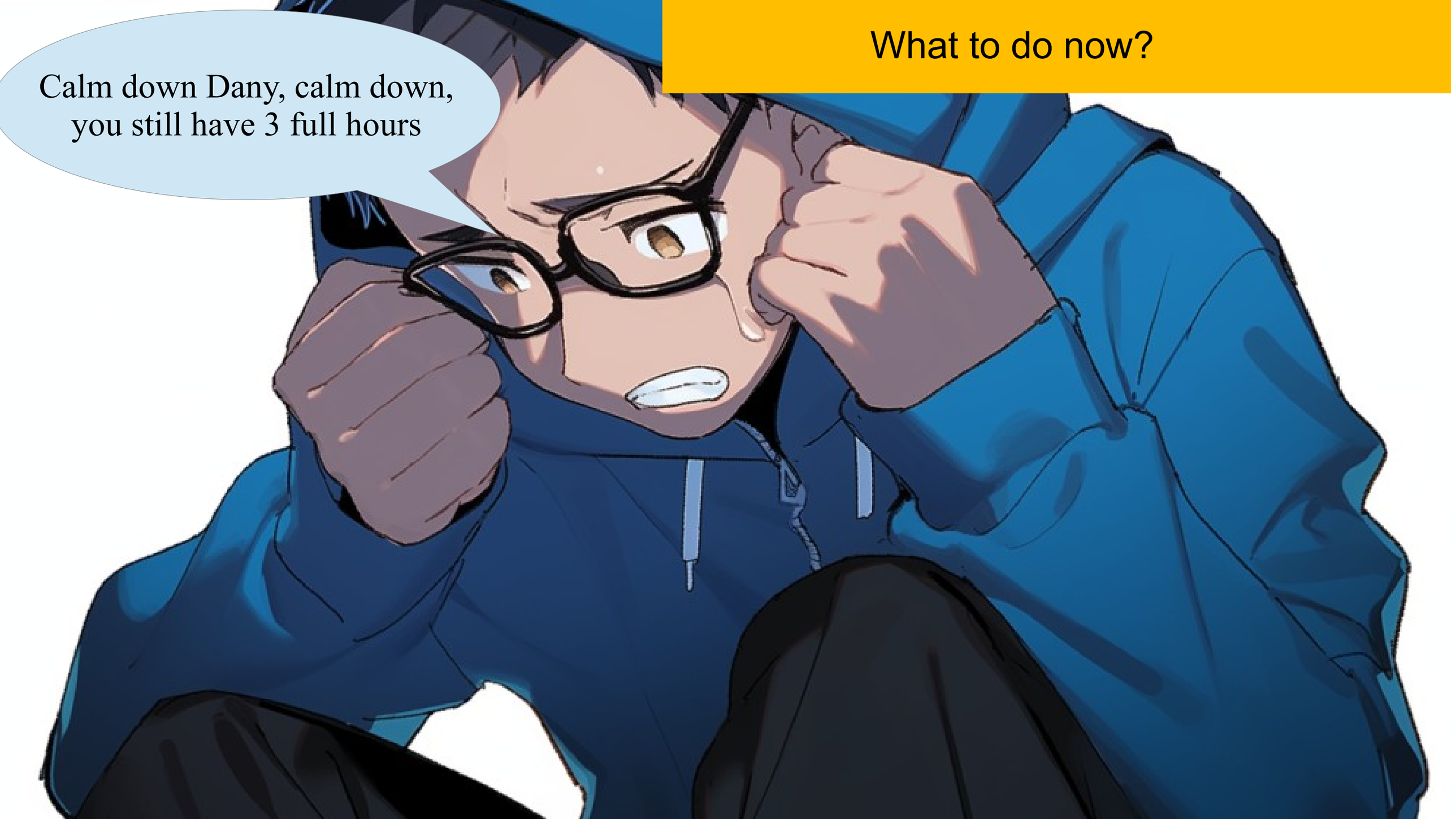

So, this is your new submission?
It is still pretty bad!
One method between map, flatmap and filter is just wrong.
Moreover, you completely forgot the method ‘or’
Dang, the submission is not accepted!


So, this is your new submission?
It is still pretty bad!
One method between map, flatmap and filter is just wrong.
Moreover, you completely forgot the method ‘or’
You are now down to 60%
Dang, the submission is not accepted!

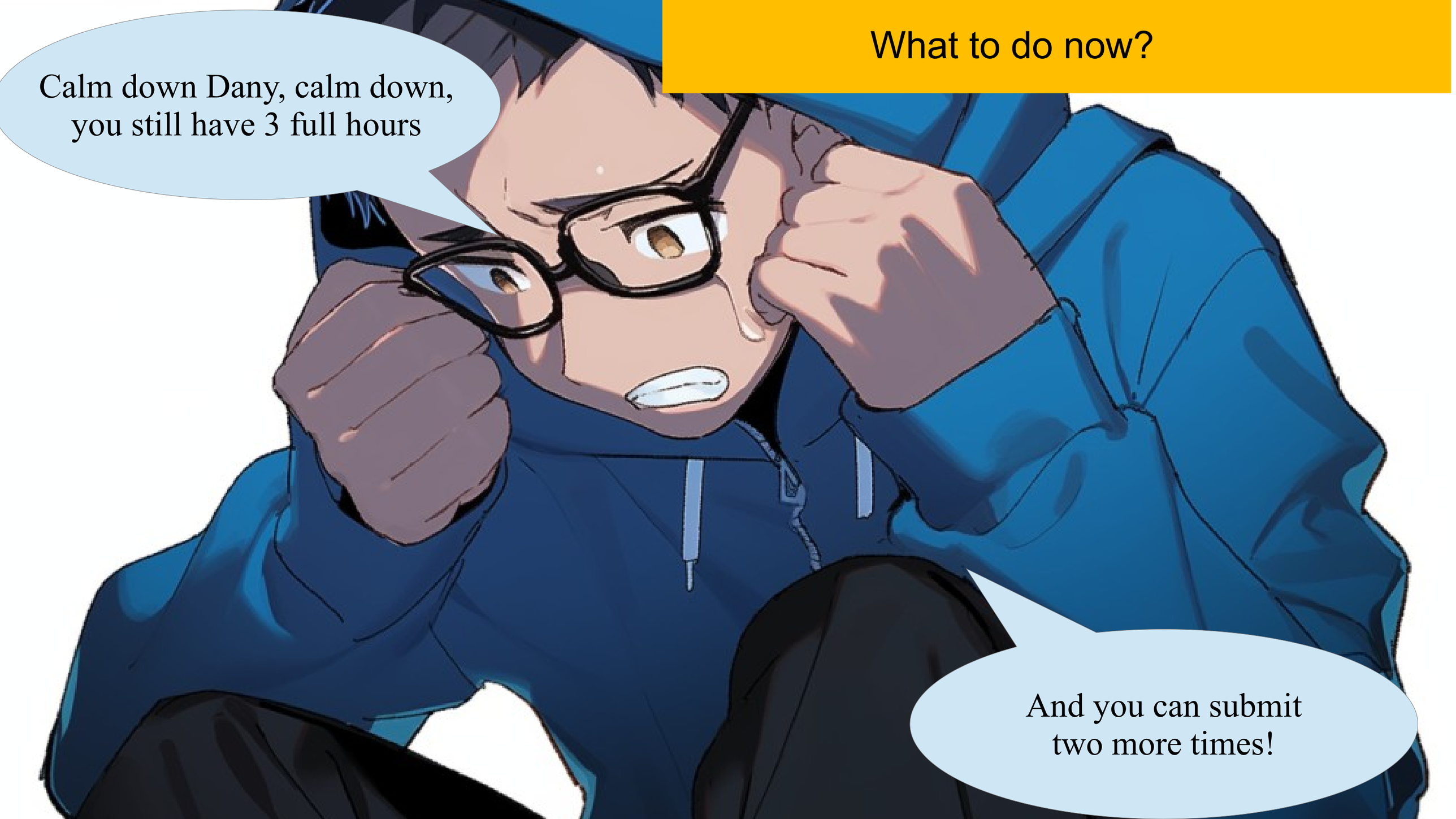
What to do now?



What to do now?



What to do now?




I will take it slow,
read and re-read it all,
and get to the solution


Who I’m I kidding!


For the first problem
I failed a submission while
using my power!


For the first problem
I failed a submission while
using my power!
Without it, I will just waste
a submission for nothing!


BANG!


I’m done for!
BANG!


My time at UPU
ends today


I’m so desperate
I can not move


I’m so desperate
I can not move
I think I will
just take a nap


Ouch, the floor was hard and cold


Ouch, the floor was hard and cold
How long did I slept?


Two hours
and a half!


Two hours
and a half!
Technically, I still have
half an hour left to submit


Not that it matters!


Not that it matters!
I never managed to use my power more then twice a day!


But, … I just slept for more then two hours


But, … I just slept for more then two hours
Maybe, … my mind recovered?


Ok focus


Ok focus
The feedback asked to implement the ‘or’ method


Ok focus
The feedback asked to implement the ‘or’ method
But, I had no idea that Optionals had an ‘or’ method at all!


Can I figure it out
with just reasoning?


Angelic Explanations, please


Answer to my call again!


The only ‘or’ I know is from booleans.


The only ‘or’ I know is from booleans.
It takes two booleans


The only ‘or’ I know is from booleans.
It takes two booleans
tooCold
tooHot


The only ‘or’ I know is from booleans.
It takes two booleans
tooCold
tooHot


The only ‘or’ I know is from booleans.
It takes two booleans
tooCold
or
tooHot


The only ‘or’ I know is from booleans.
It takes two booleans
tooCold
or
tooHot


The only ‘or’ I know is from booleans.
It takes two booleans
tooCold
or
tooHot
sadClimate


tooCold
or
tooHot
sadClimate


It produces a boolean result
tooCold
or
tooHot
sadClimate