



And with this realization, Dany was fully out of power





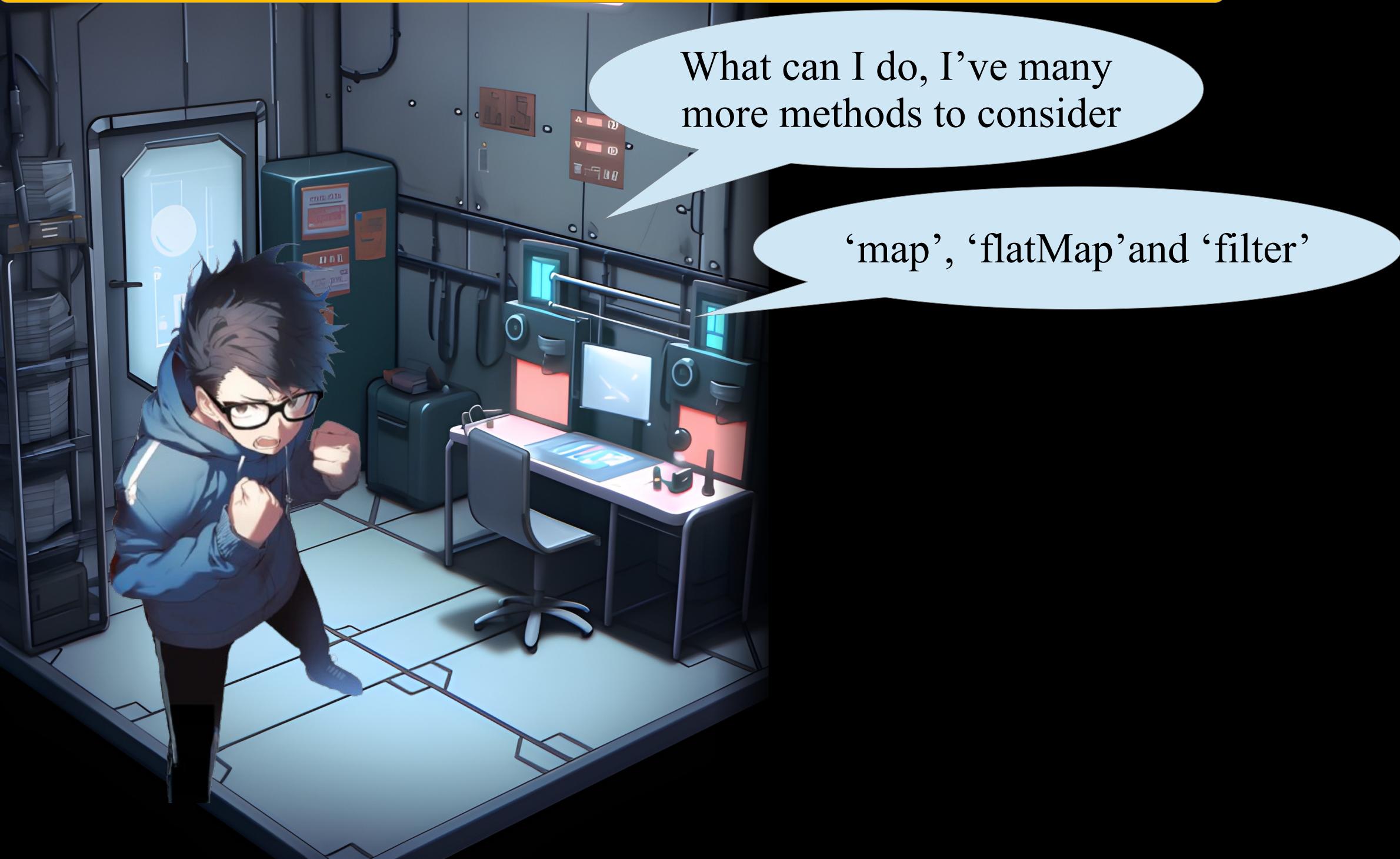
Now perceiving the surrounding reality again,  
Dany is blocked in the room with much more to answer



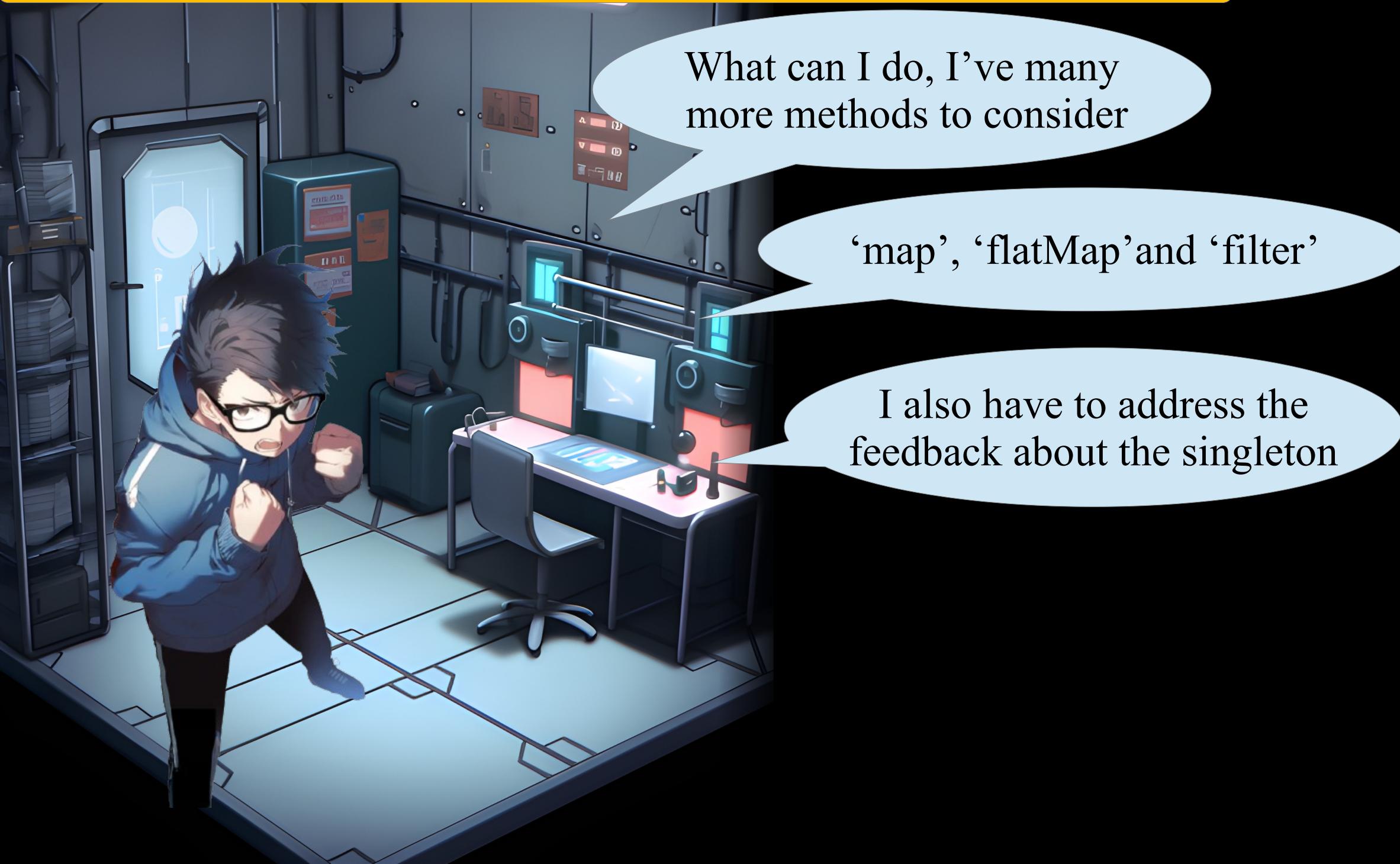
Now perceiving the surrounding reality again,  
Dany is blocked in the room with much more to answer



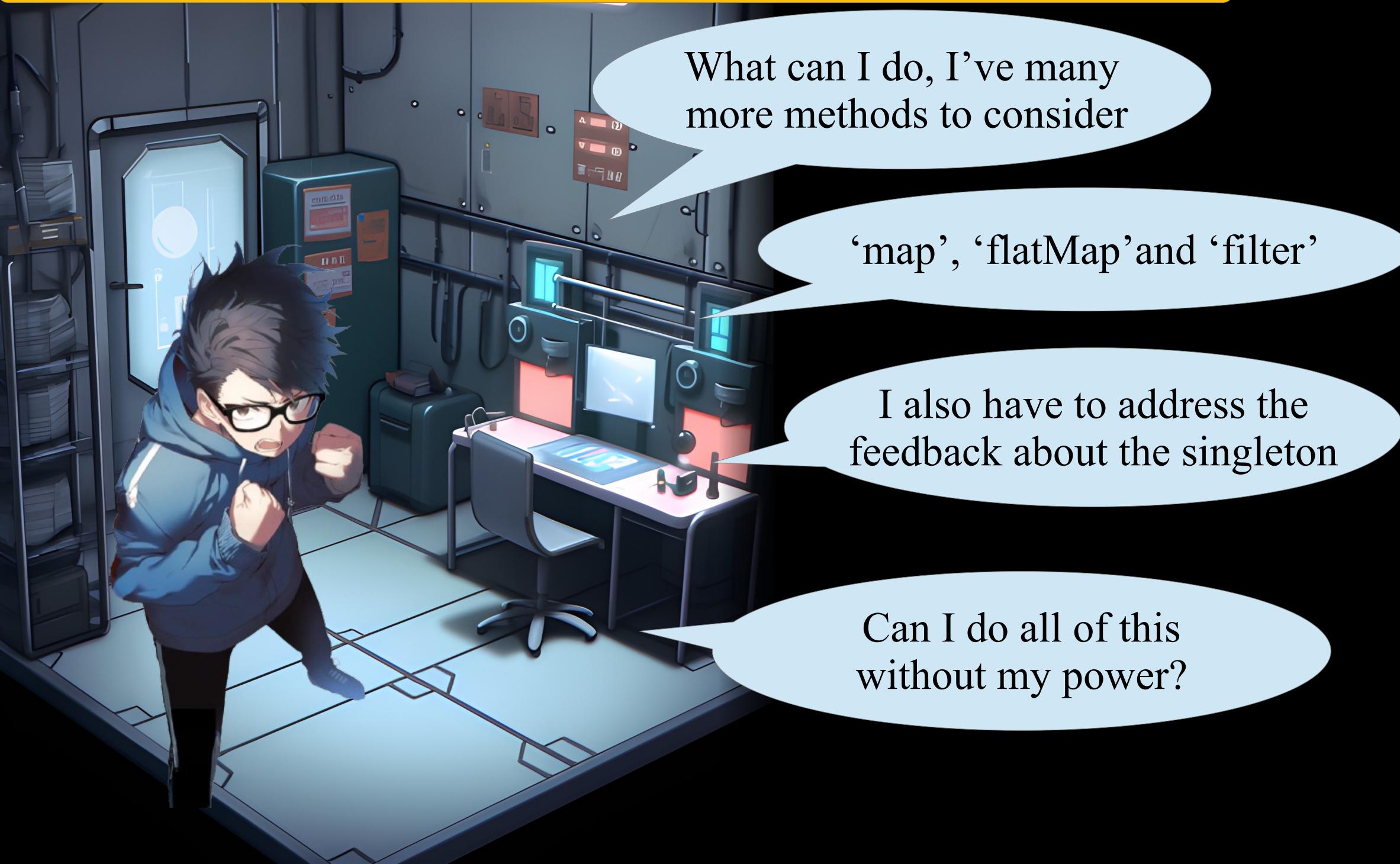
Now perceiving the surrounding reality again,  
Dany is blocked in the room with much more to answer



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



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







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?

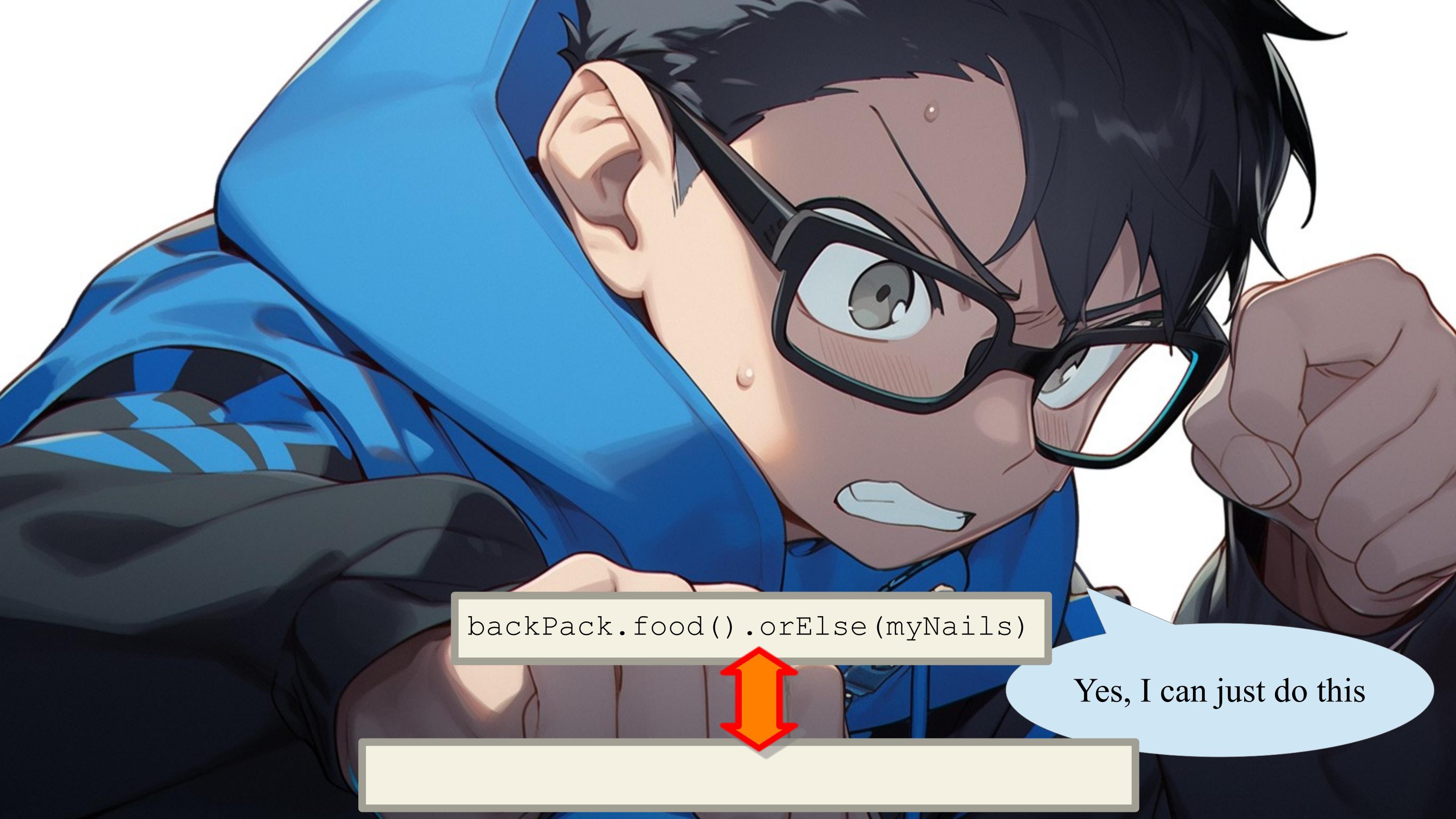


A close-up of a man with dark hair and glasses, wearing a blue suit. He has a determined expression and is holding a small device. A speech bubble originates from his mouth.

Yes, I can just do this



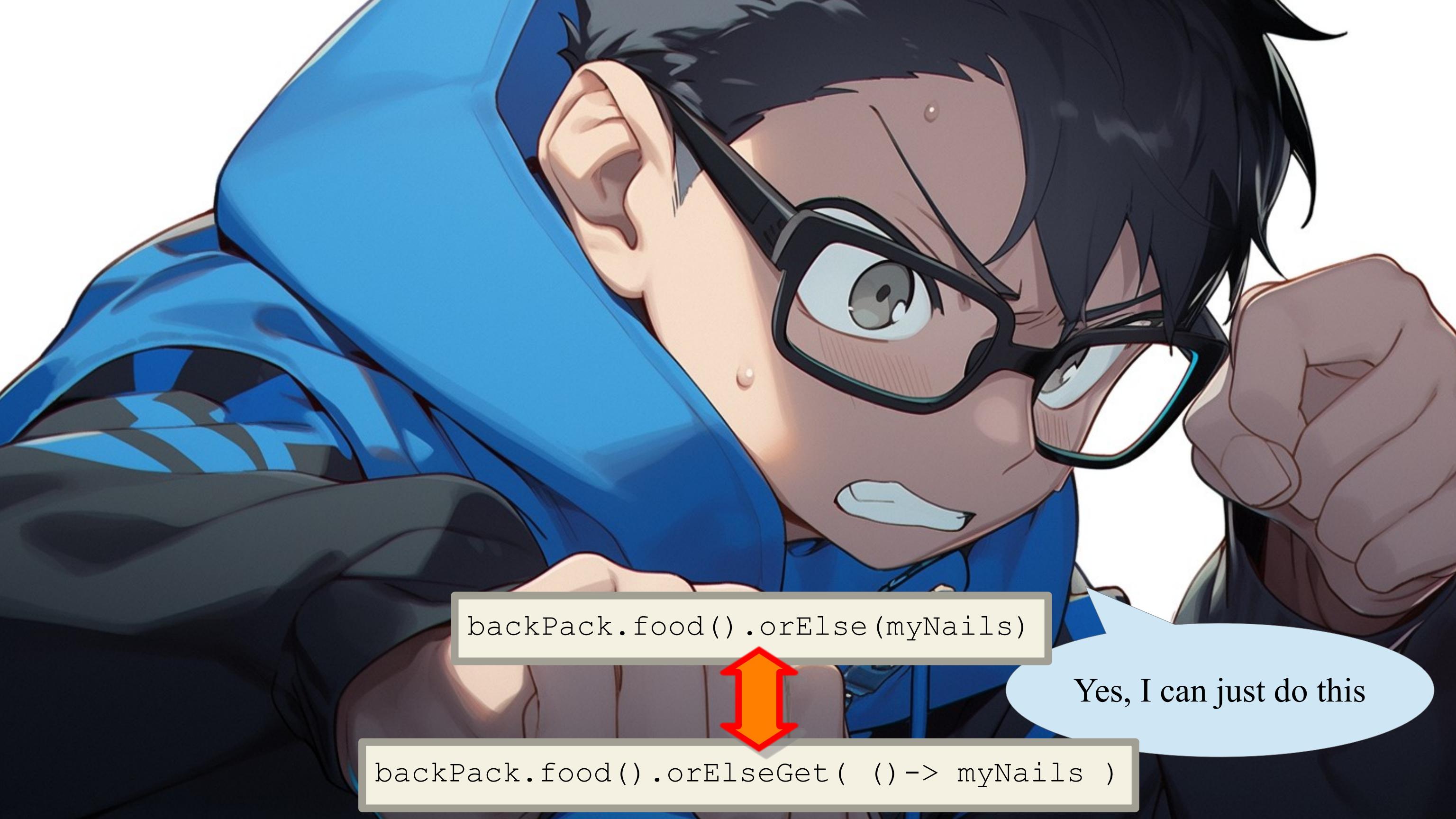
Yes, I can just do this



backPack.food().orElse(myNails)



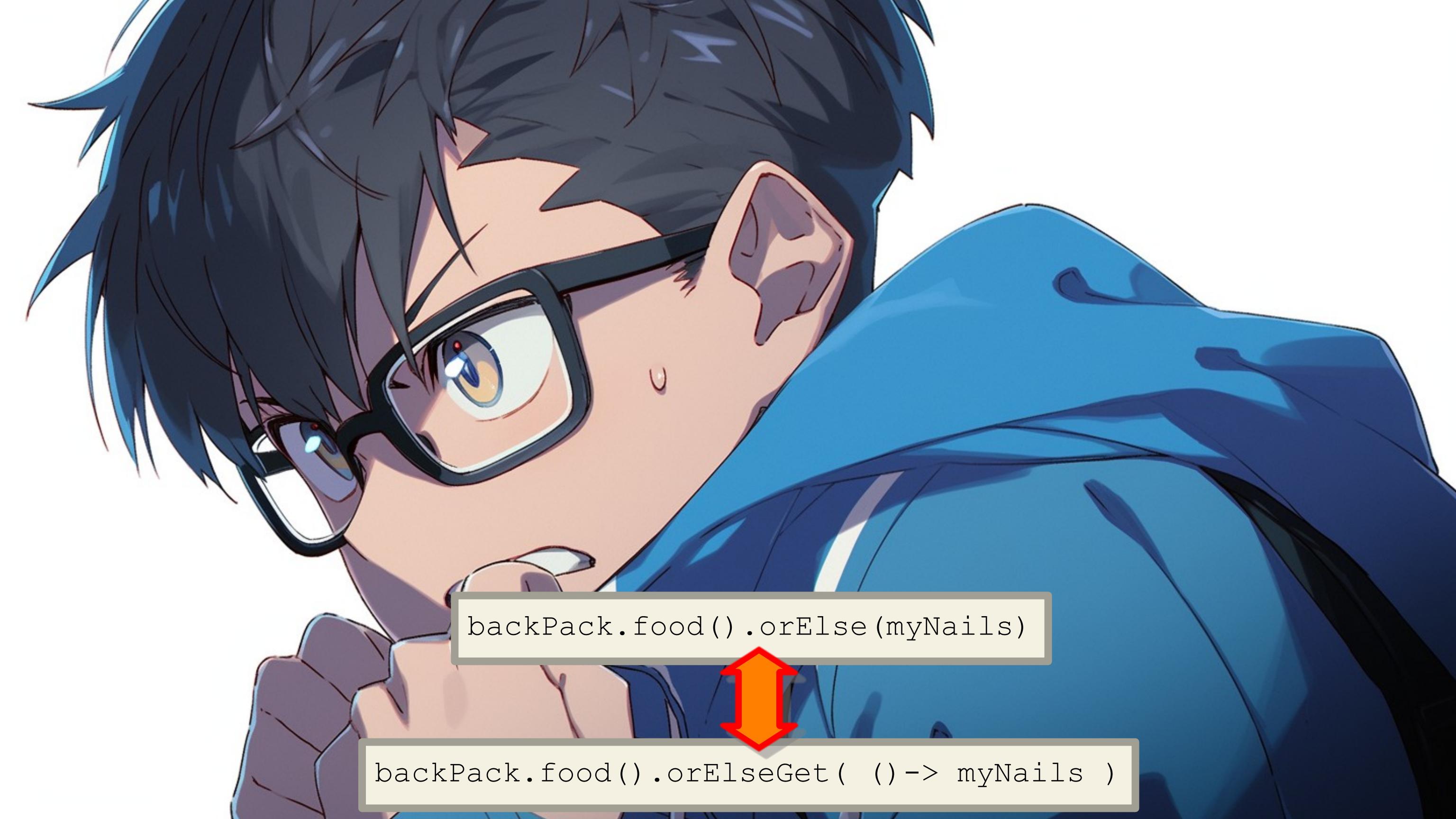
Yes, I can just do this



```
backPack.food().orElse(myNails)
```

Yes, I can just do this

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

A close-up of a character with dark hair and glasses looking at a mobile device. A red double-headed vertical arrow points between two code snippets.

```
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 trebuchet ‘()->’

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 trebuchet ‘()->’

backPack.food().orElse(myNails)

Yes, ‘orElse’ is  
truly redundant

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



A close-up of a character with long, dark blue hair and blue-rimmed glasses. They are wearing a purple turtleneck and a blue jacket. The character is looking down at a black smartphone held in their hands. A speech bubble originates from the character's ear, containing the text "Now for the singleton".

Now for the singleton



Now for the singleton

I remember one  
of the Pupon chants:

A close-up of a character with long, dark blue hair and blue-rimmed glasses. They have a thoughtful expression, with their hand near their chin. Three speech bubbles originate from their mouth, containing the text.

Now for the singleton

I remember one  
of the Pupon chants:

“The empty list is a list”

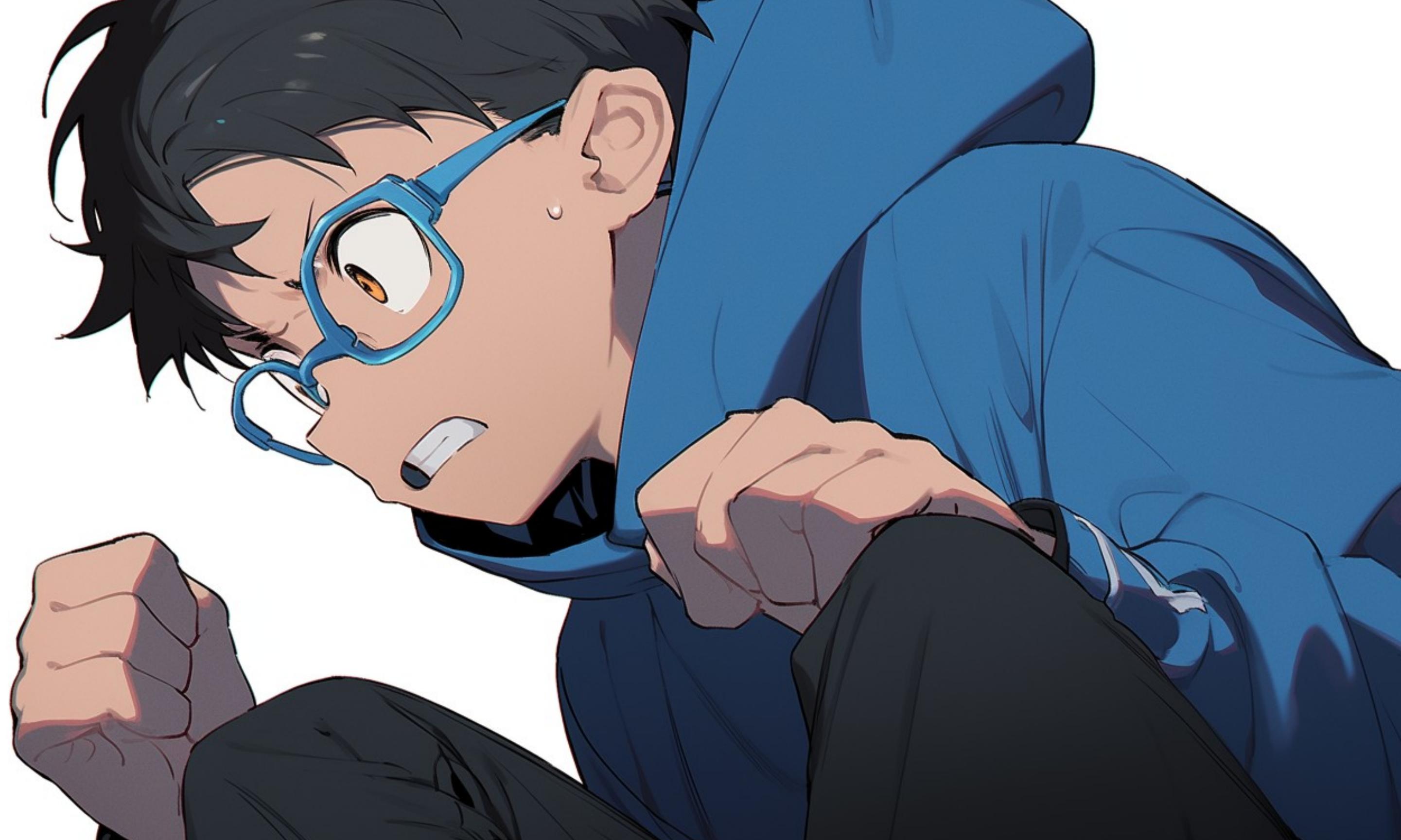


Now for the singleton

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

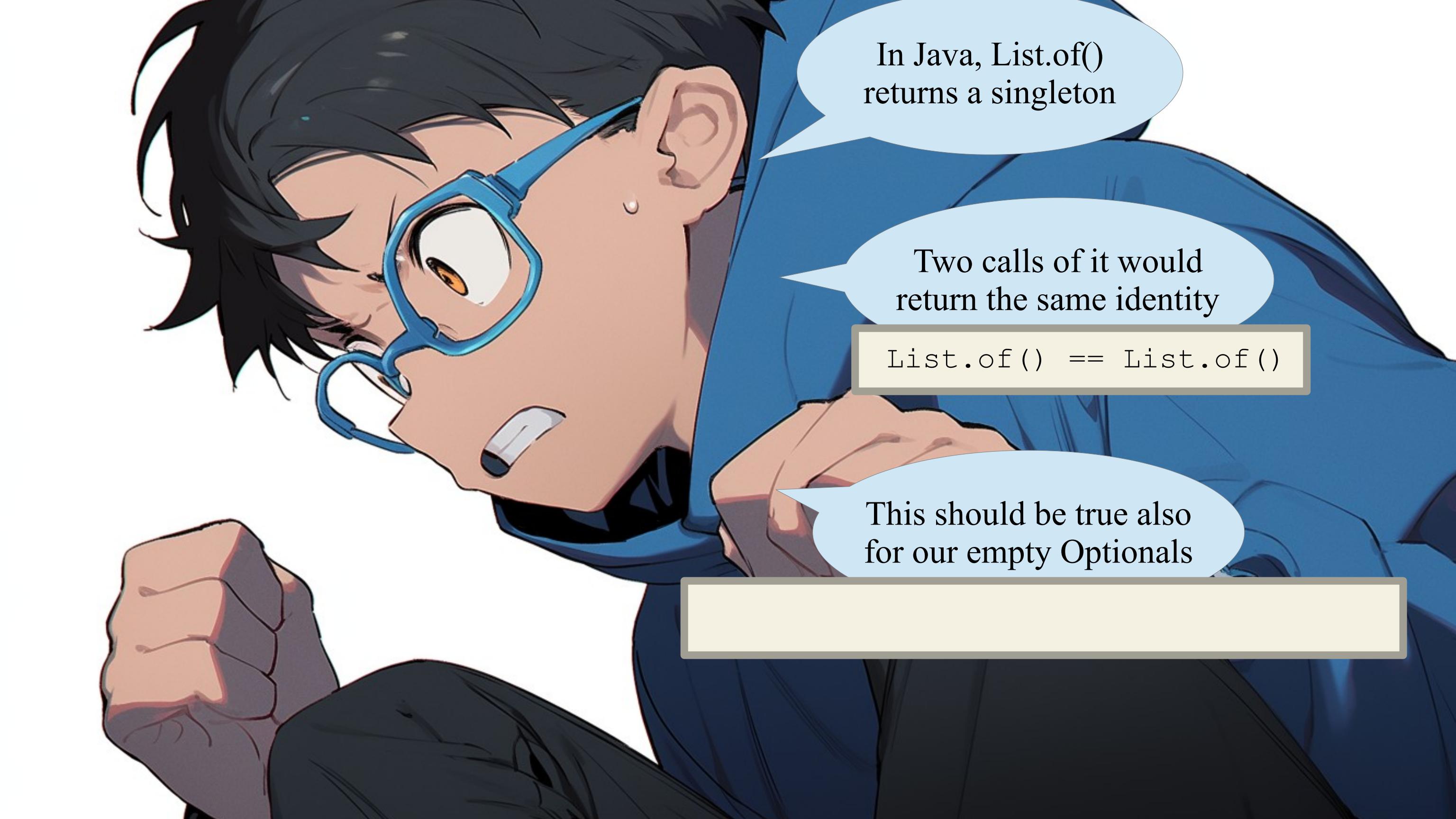
Two calls of it would  
return the same identity



In Java, `List.of()`  
returns a singleton

Two calls of it would  
return the same identity

`List.of() == List.of()`

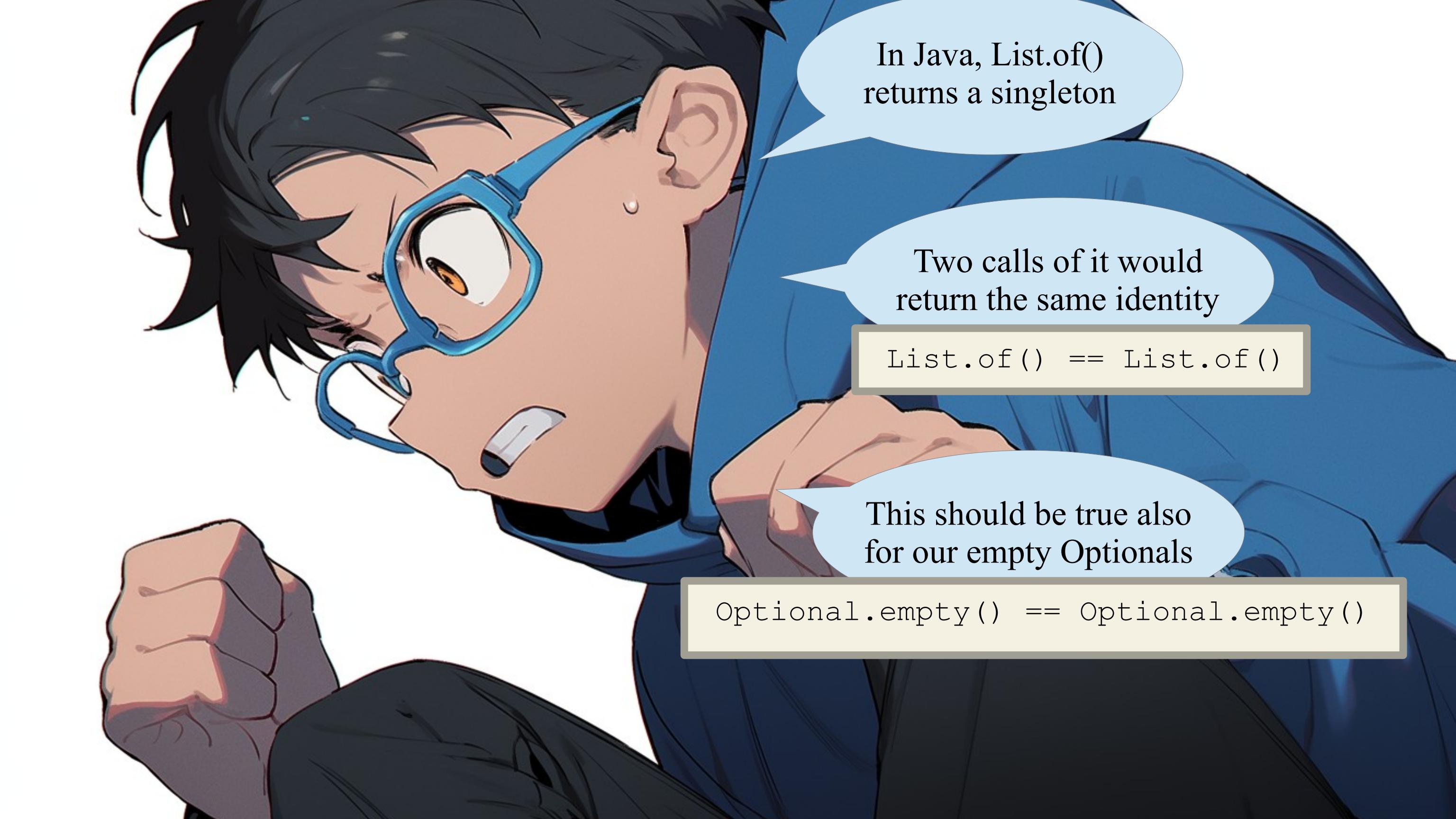


In Java, `List.of()`  
returns a singleton

Two calls of it would  
return the same identity

`List.of() == List.of()`

This should be true also  
for our empty Optionals

A close-up illustration of a man's face. He has dark hair, wears blue-rimmed glasses, and a white stethoscope around his neck. He is looking slightly upwards and to the right with a thoughtful expression.

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?



A close-up shot of a character with dark hair and glasses, wearing a blue suit. The character has a determined expression, with their hands clenched into fists near their chin. A speech bubble originates from the bottom left.

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() {}  
    ...  
}  
...  
return instance; }  
implements 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() {}
    ...
}
```



Who am I kidding?  
I can't do this!

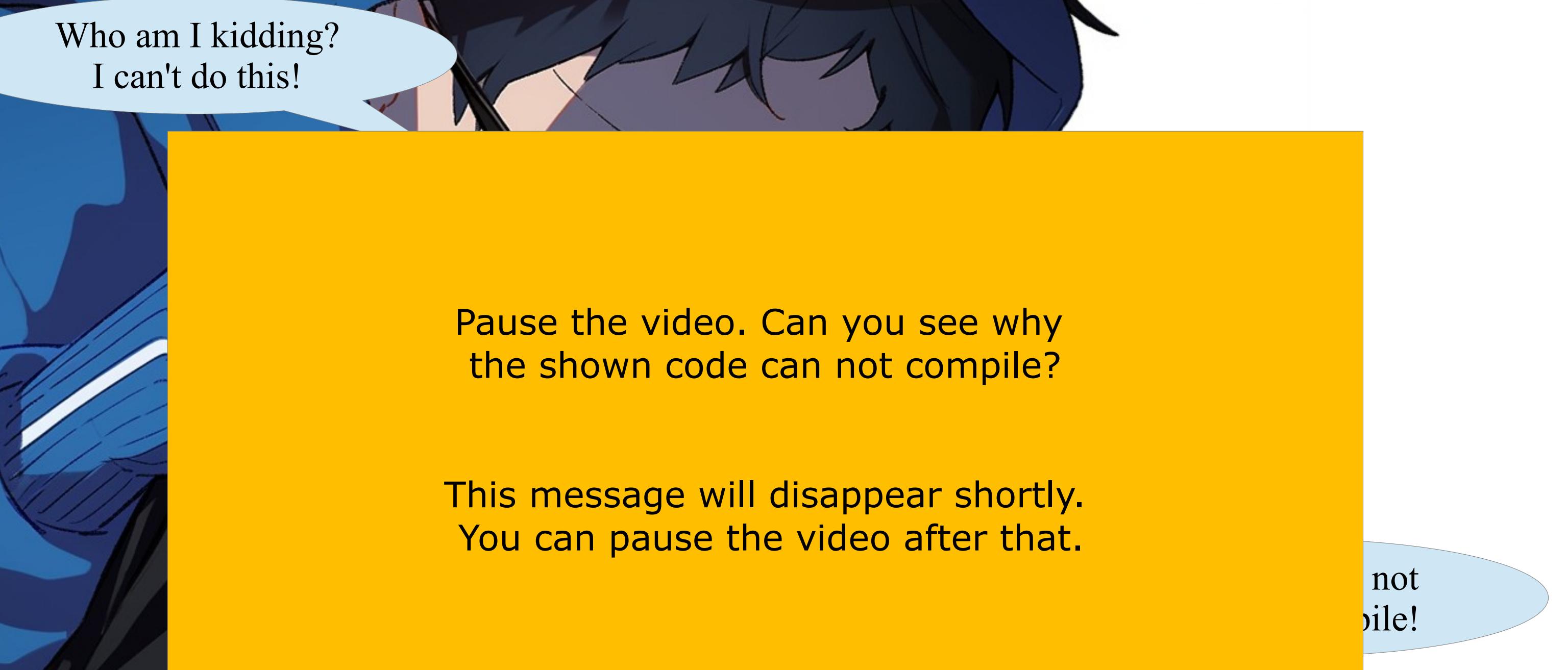
```
final class Empty<T> implements Optional<T>{
    static final Empty<T> instance= new Empty<T>();
    private Empty() {}
    ...
}
```



Who am I kidding?  
I can't do this!

This code can not  
possibly compile!

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



Who am I kidding?  
I can't do this!

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  
compile!

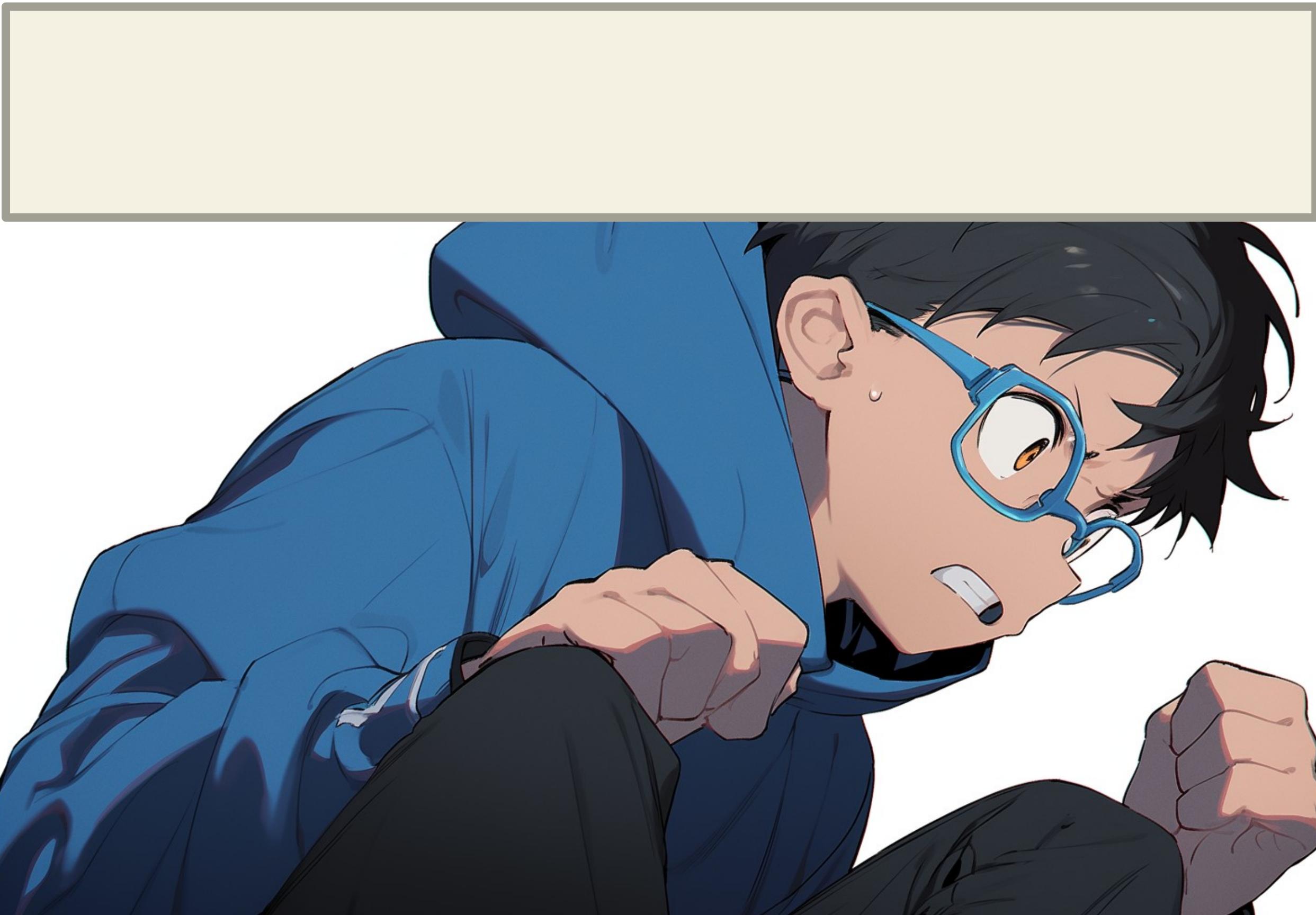
```
final class Empty {  
    static final Empty<T> instance= new Empty<T>();  
    private Empty() {}  
    ...  
}
```



Who am I kidding?  
I can't do this!

This code can not  
possibly compile!

```
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() {}  
    ...  
}
```

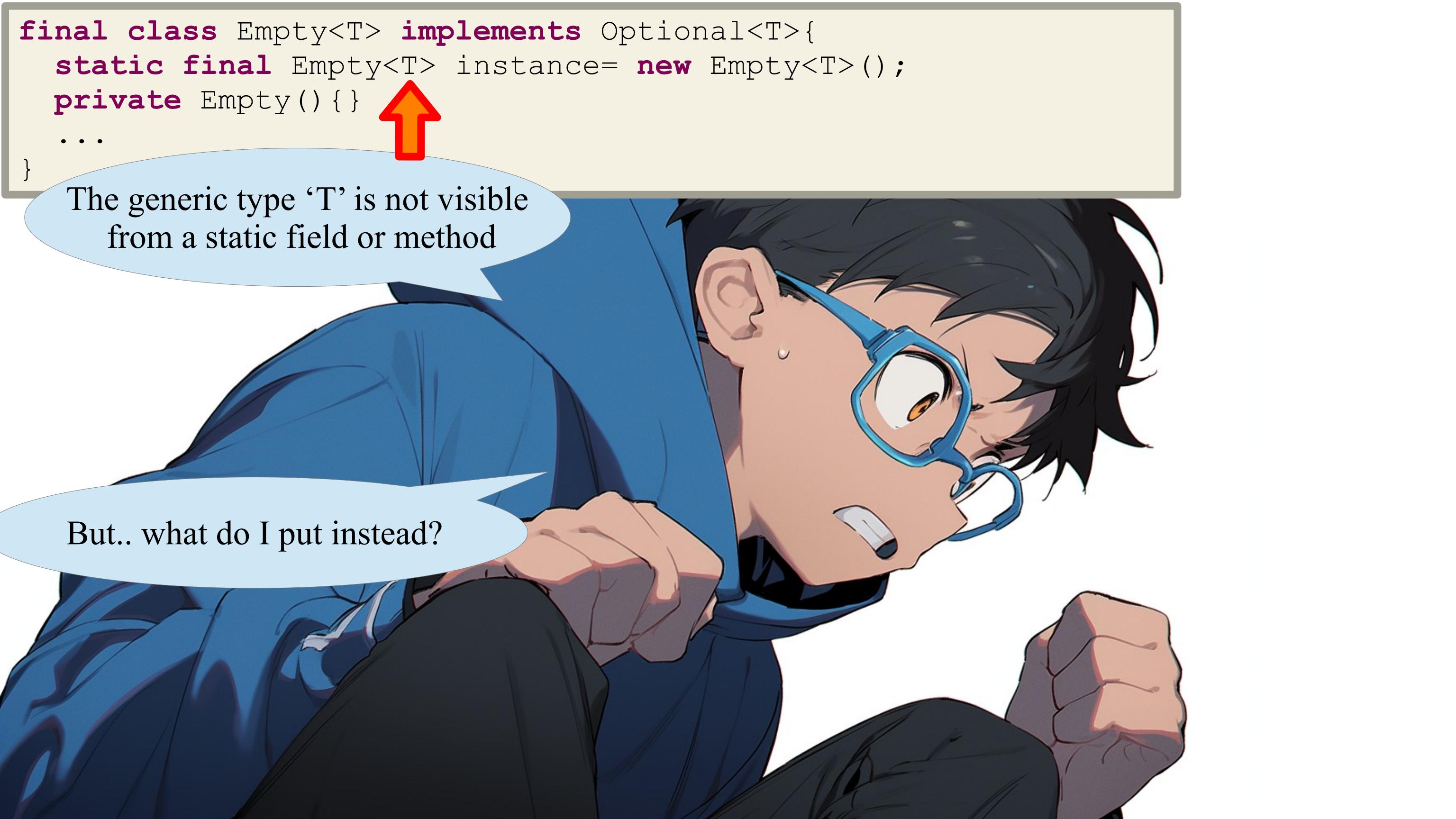


```
final class Empty<T> implements Optional<T>{
    static final Empty<T> instance= new Empty<T>();
    private Empty() { }
    ...
}
```

The generic type ‘T’ is not visible  
from a static field or method



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



The generic type ‘T’ is not visible  
from a static field or method

But.. what do I put instead?

```
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() { }
    ...
}
return Empty<T> instance;
```

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() { }
    ...
}
return
```

If I were to put Object

Then this line  
would not compile



```
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() {}  
    ...  
}
```

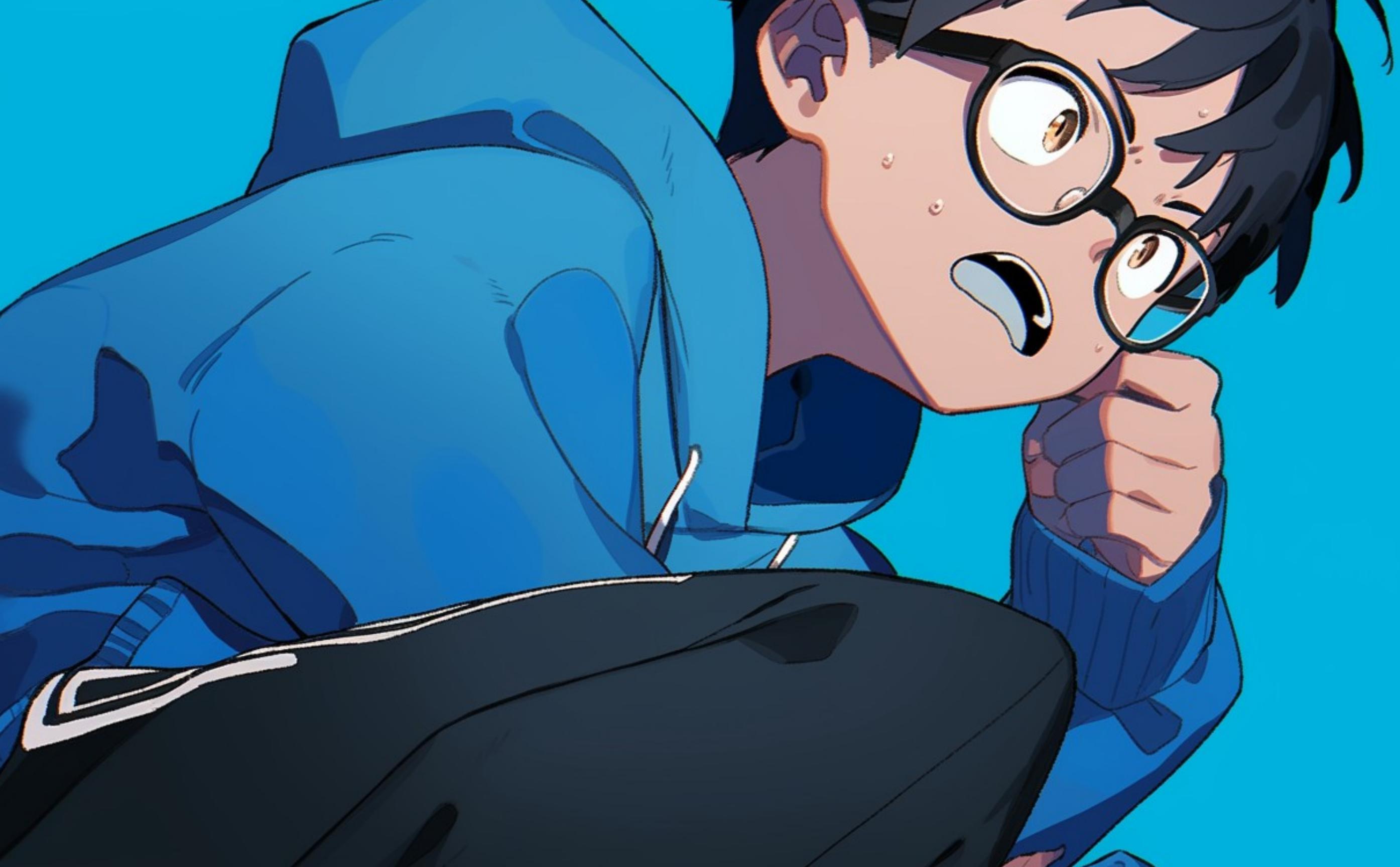
If I were to put Object

```
implements Optional<T>{ ... }
```

Then this line  
would not compile



Should I just cast my  
problem away?





I mean, there is  
no 'T' instance

A cartoon illustration of a man with dark hair and glasses, looking shocked at a laptop screen. He has a small tear on his forehead and is pointing his finger towards the screen. The laptop screen shows two speech bubbles.

I mean, there is  
no 'T' instance

Stored into an empty  
Optional of T



I mean, there is  
no 'T' instance

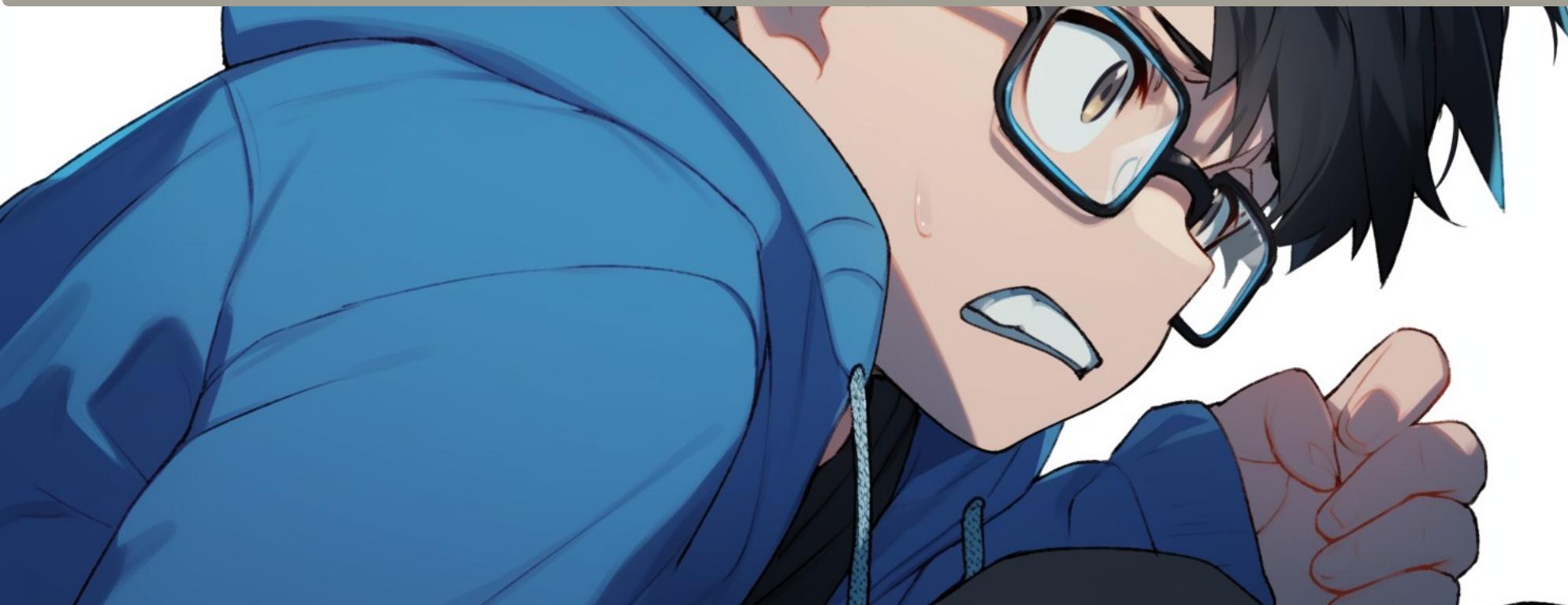
Stored into an empty  
Optional of T

So, maybe  
the cast is safe?

```
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>{ ... }
```

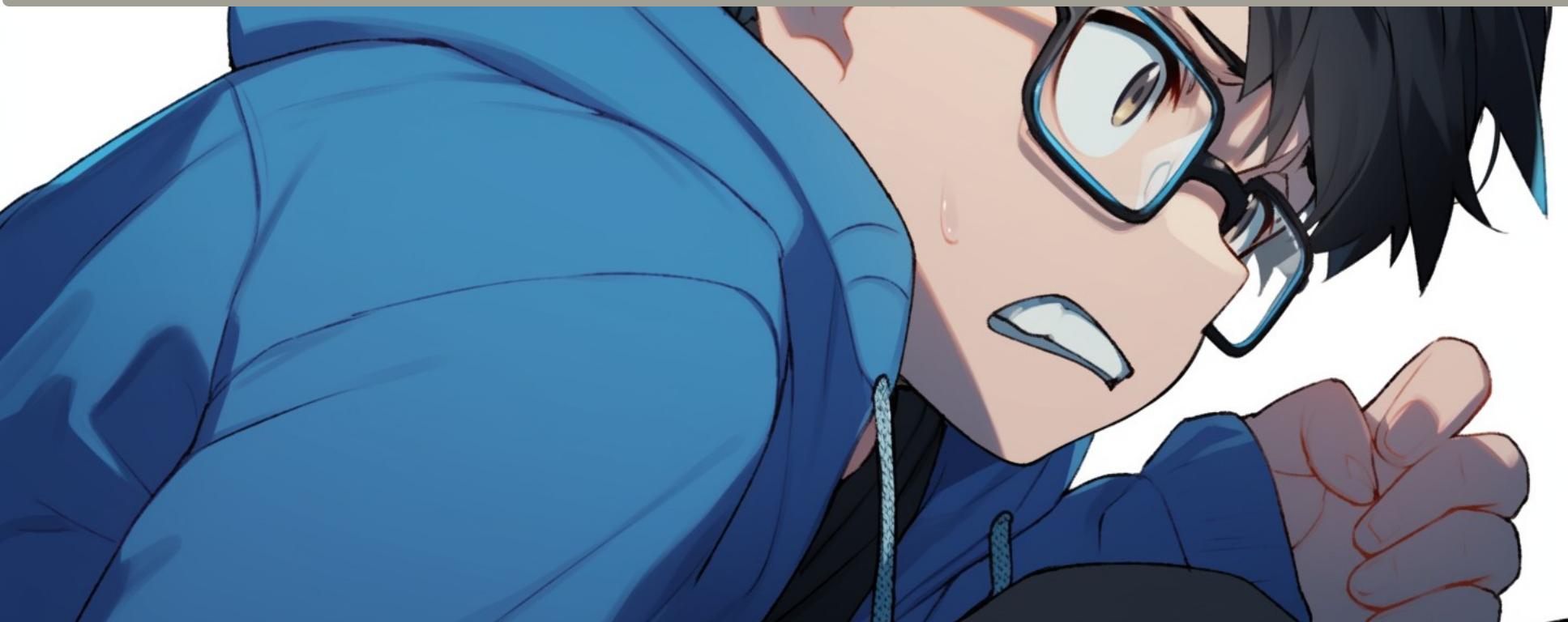


```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
    @SuppressWarnings("unchecked")
    static <E> Optional<E> empty() { return (Optional<E>) Empty.Instance; }
    ...
}

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



```
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

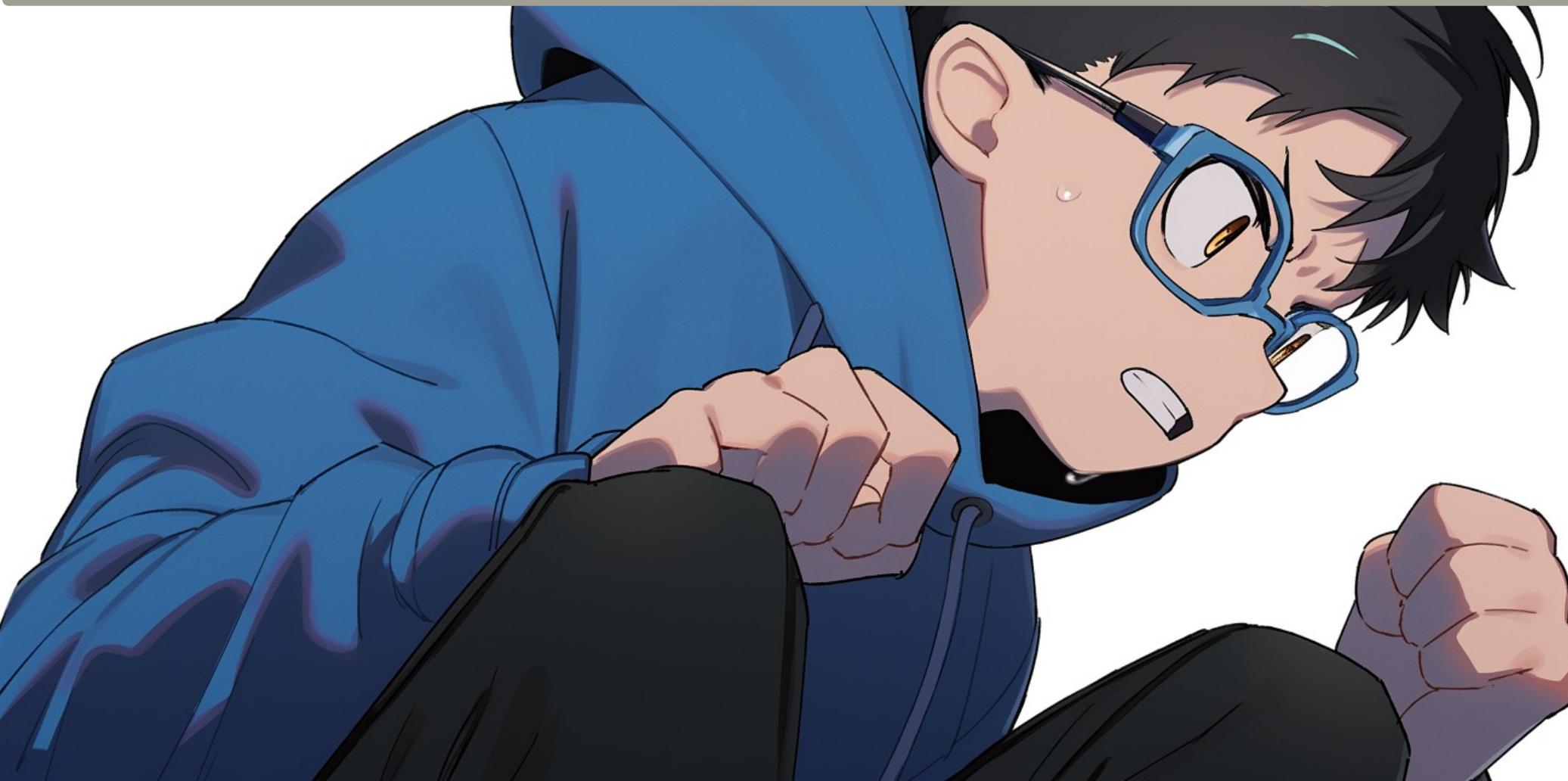
And of course we need the  
suppress warnings when  
we cast generics around



```
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??
}
```

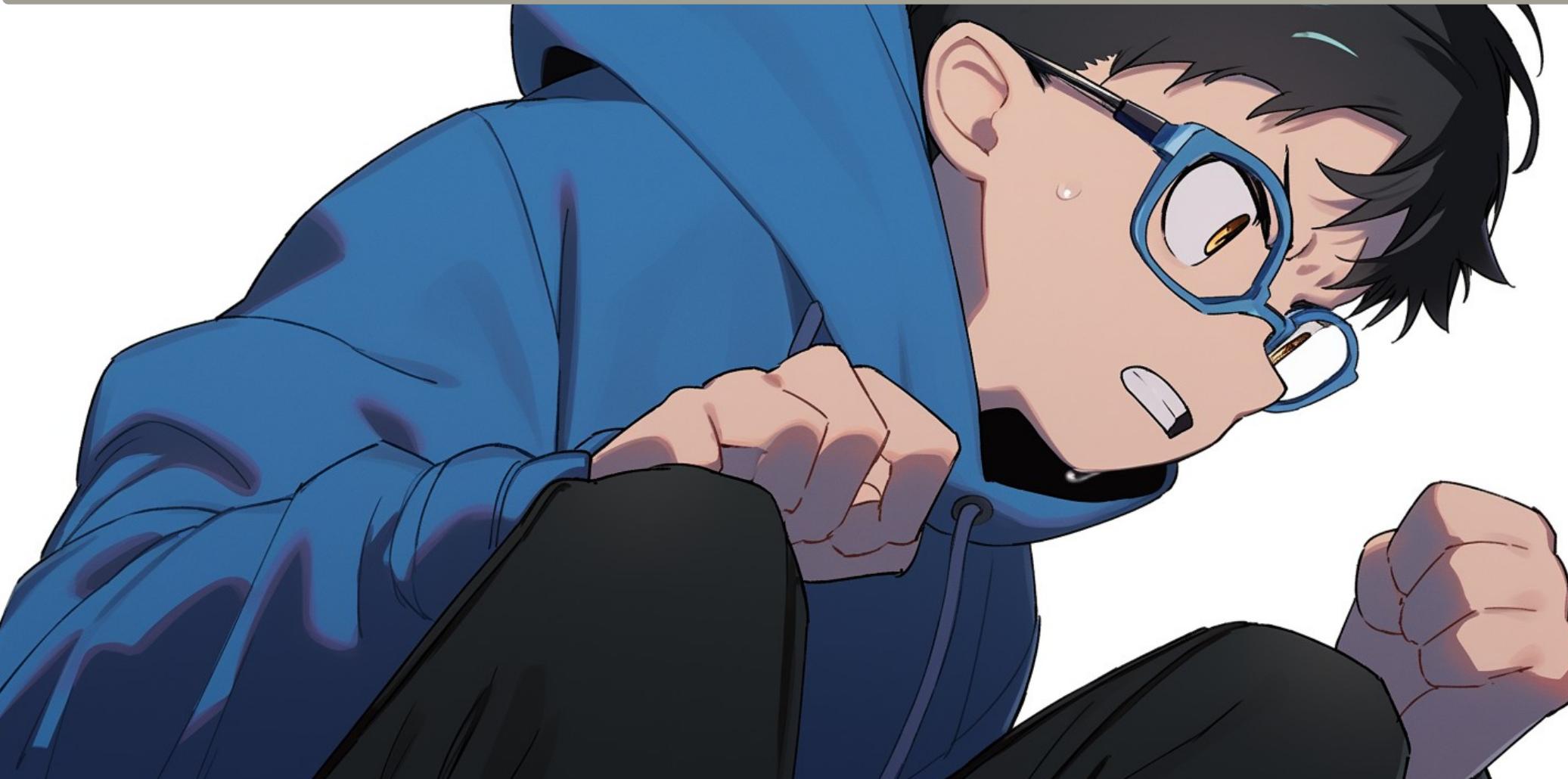


```
public sealed interface Optional<T> permits Empty<T>, Some<T>{
    MAP??
}

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...



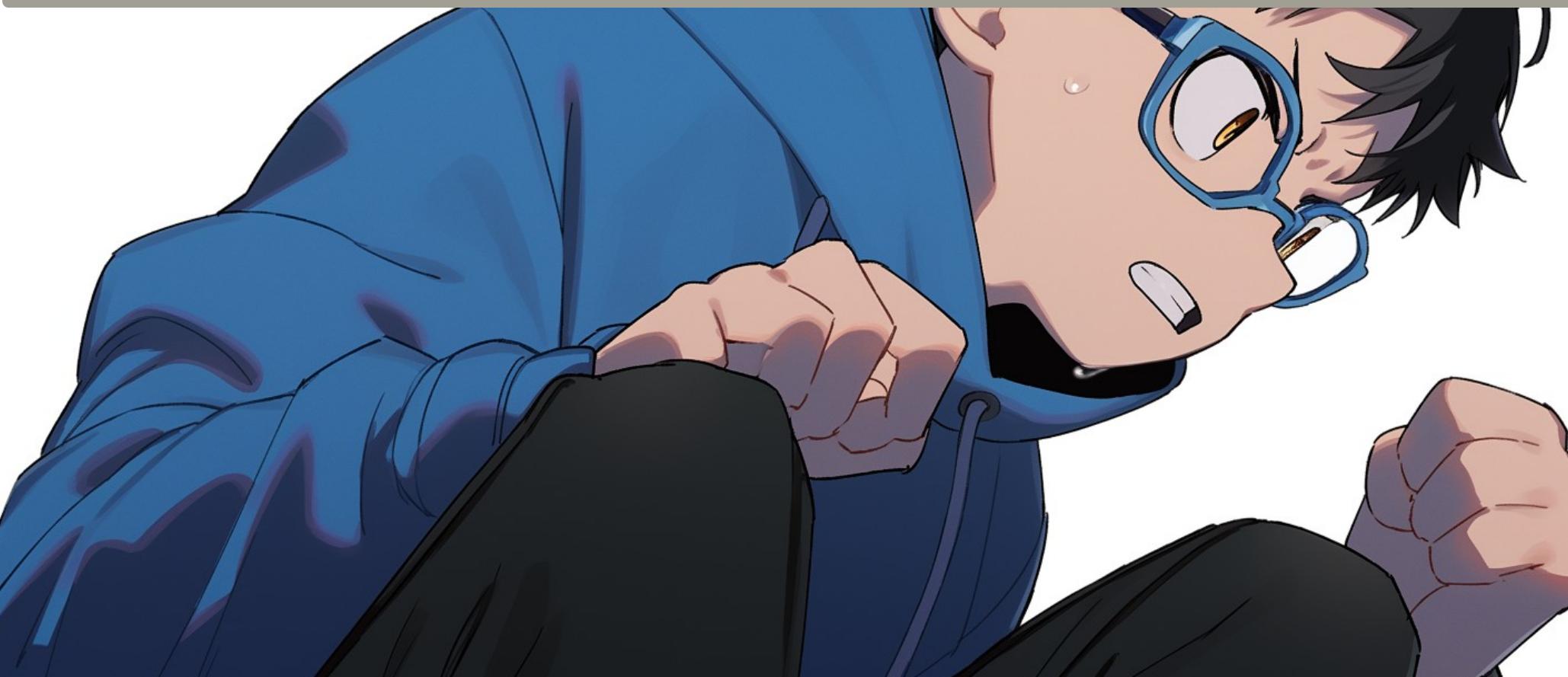
If I can not get the  
value out...

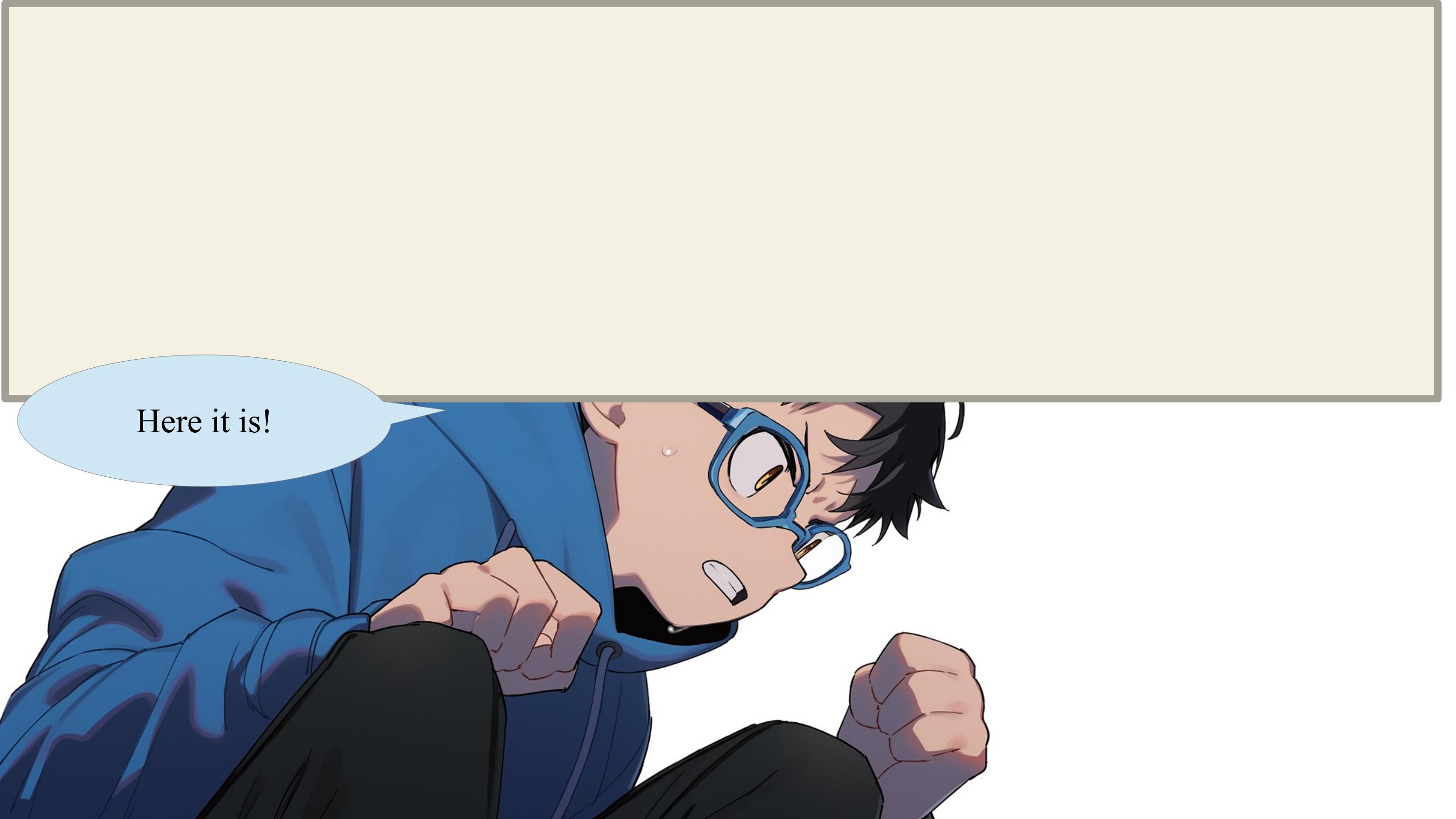
can I instead...





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);  
}
```

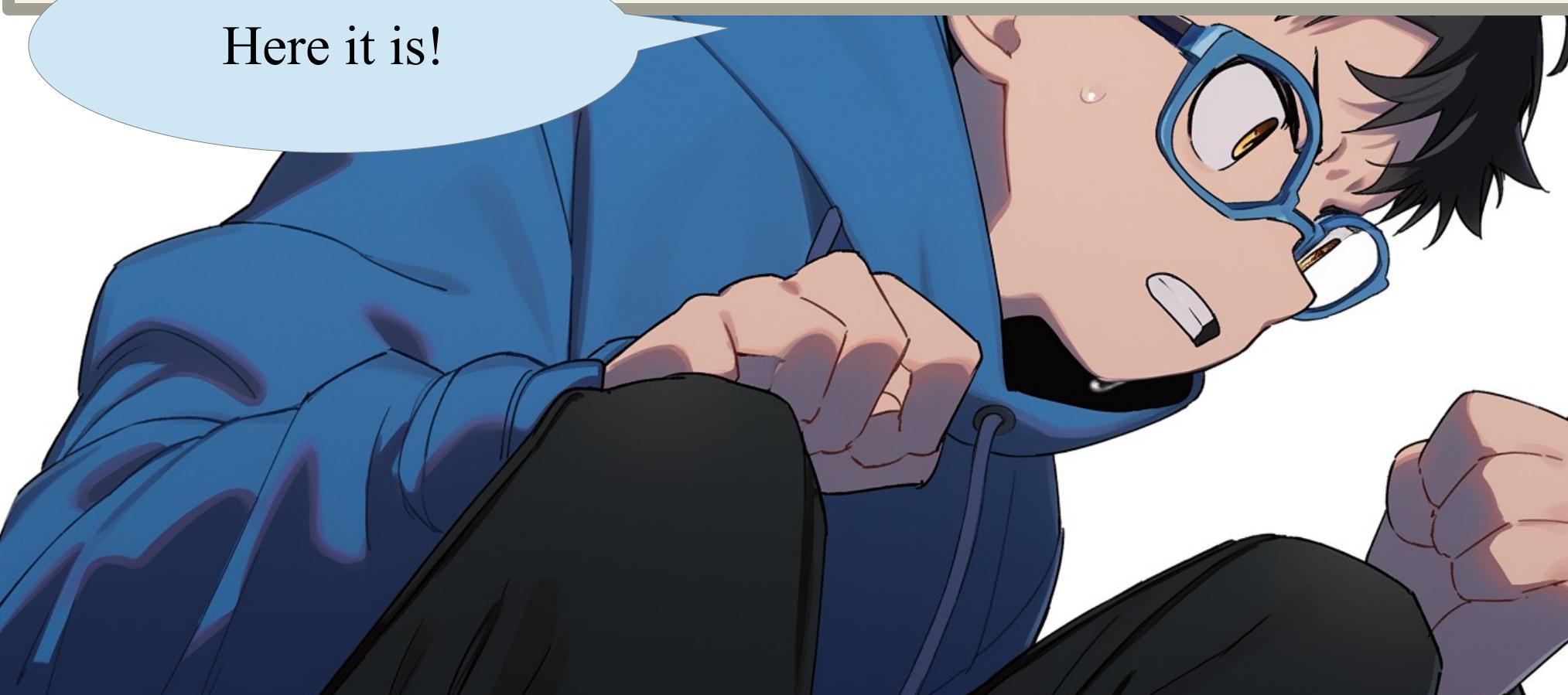


Here it is!

```
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(); }  
}
```



Here it is!



```
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));
    }
}
```



Here it is!

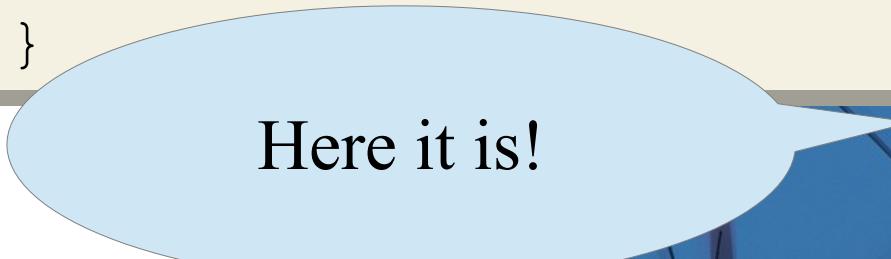
```
public sealed interface Optional<T> permits Empty<T>, Some<T> {
    <U> Optional<U> map(Function<T, U> m);
}

final class Empty<T> implements Optional<T>{
    public <U> Optional<U> map(Function<T, U> m) { return Optional.empty(); }
}

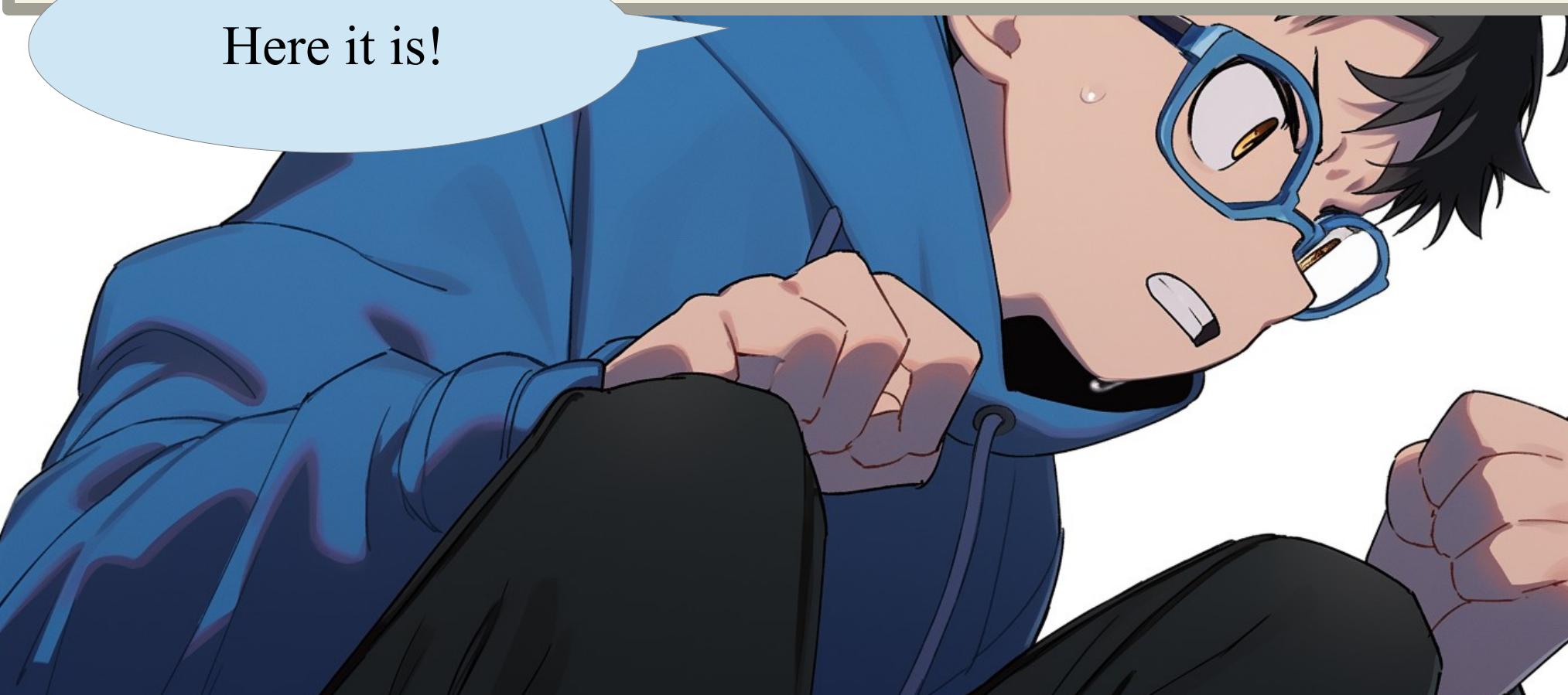
record Some<T>(T get) implements Optional<T>{
    public <U> Optional<U> map(Function<T, U> m) {
        return Optional.ofNullable(m.apply(get));
    }
}
```



Takes a function from T to U



Here it is!



```
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

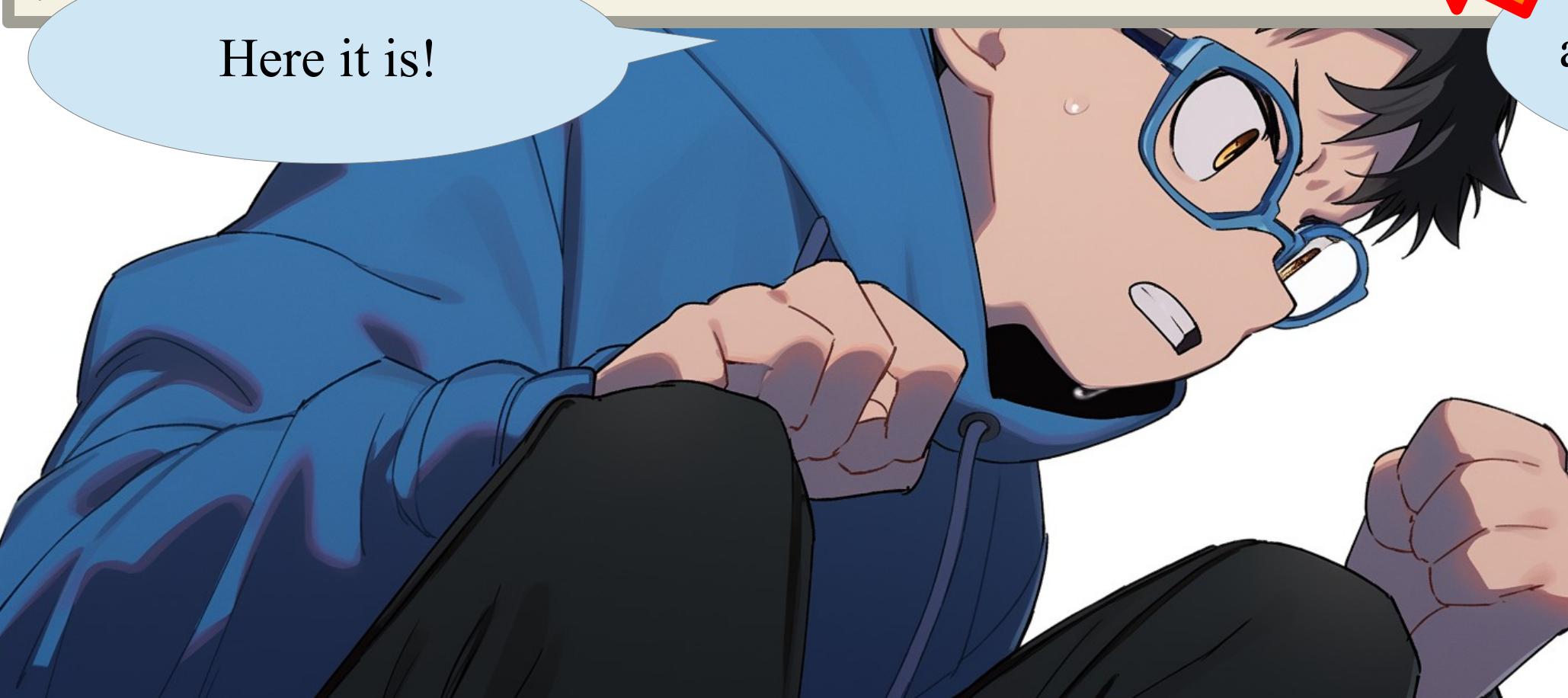


Here it is!

```
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

```
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));
    }
}
```



Here it is!

```
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



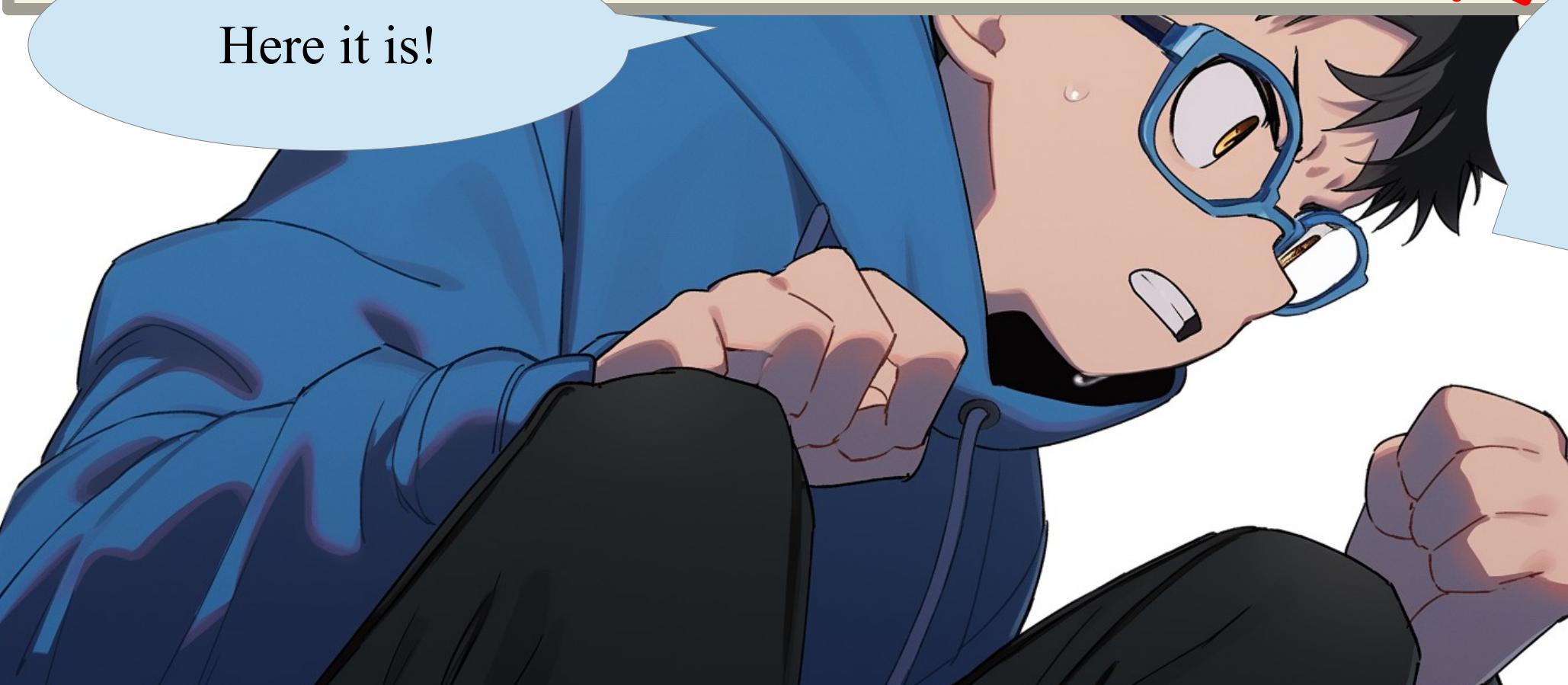
Empty is just a casted  
version of 'this'

Here it is!

```
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));
    }
}
```



Here it is!

Returns an Optional of U

Empty is just a casted version of 'this'

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





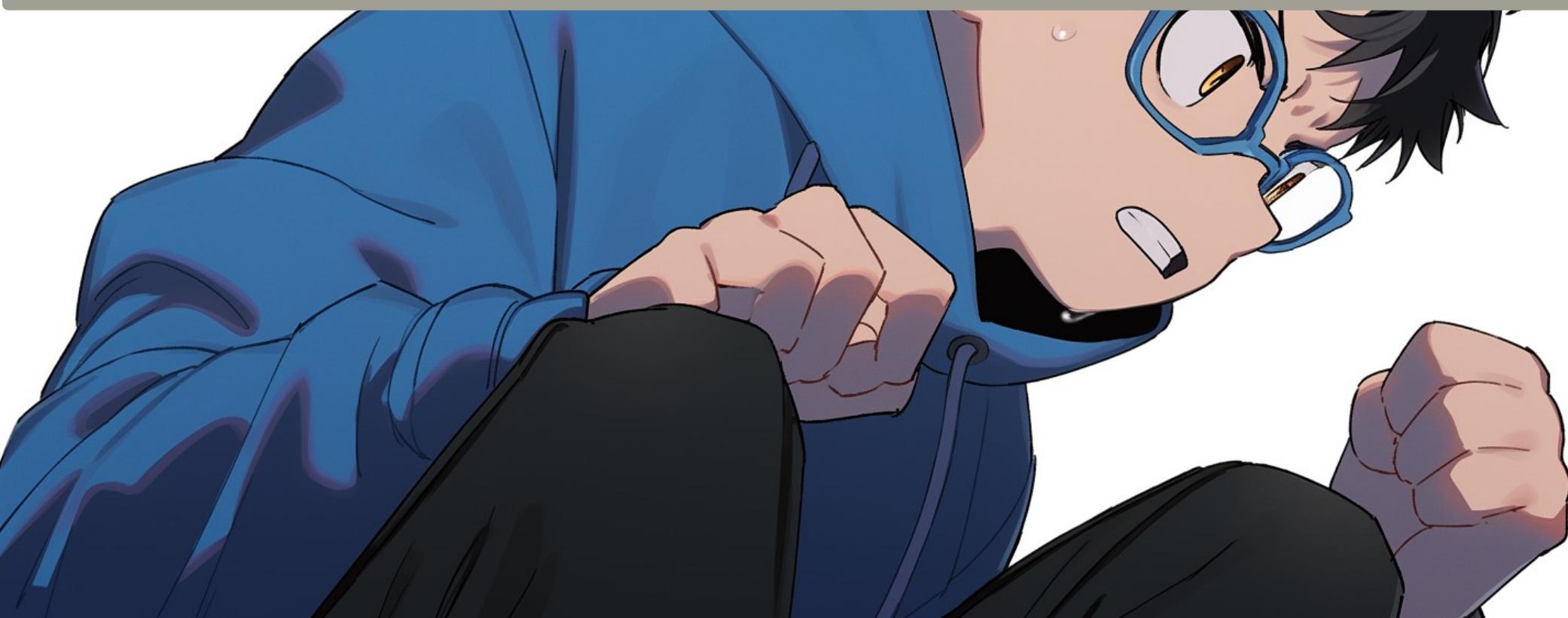
I remember this corner  
case because of Hanton!  
Personally, I would have just  
thrown an error in that case.

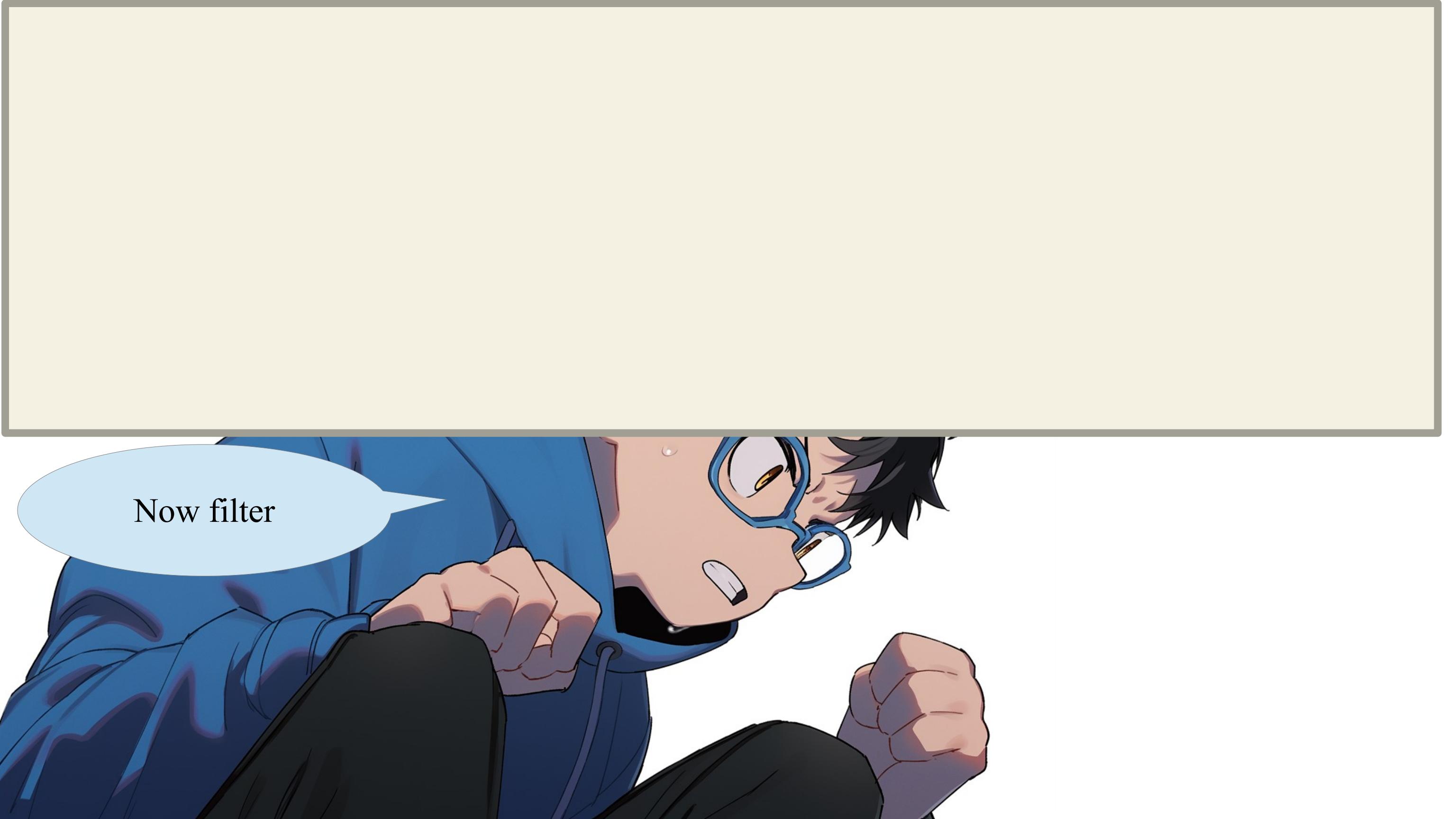


I remember this corner  
case because of Hanton!  
Personally, I would have just  
thrown an error in that case.



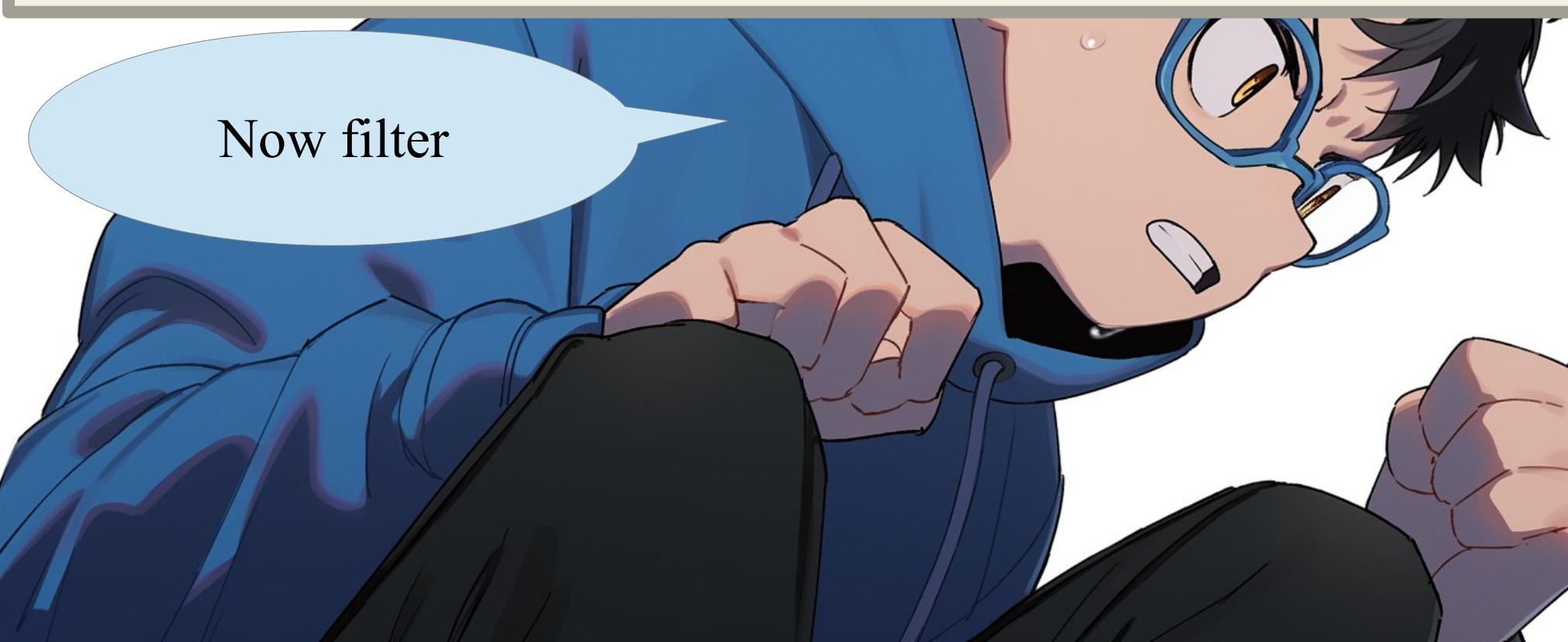






Now filter

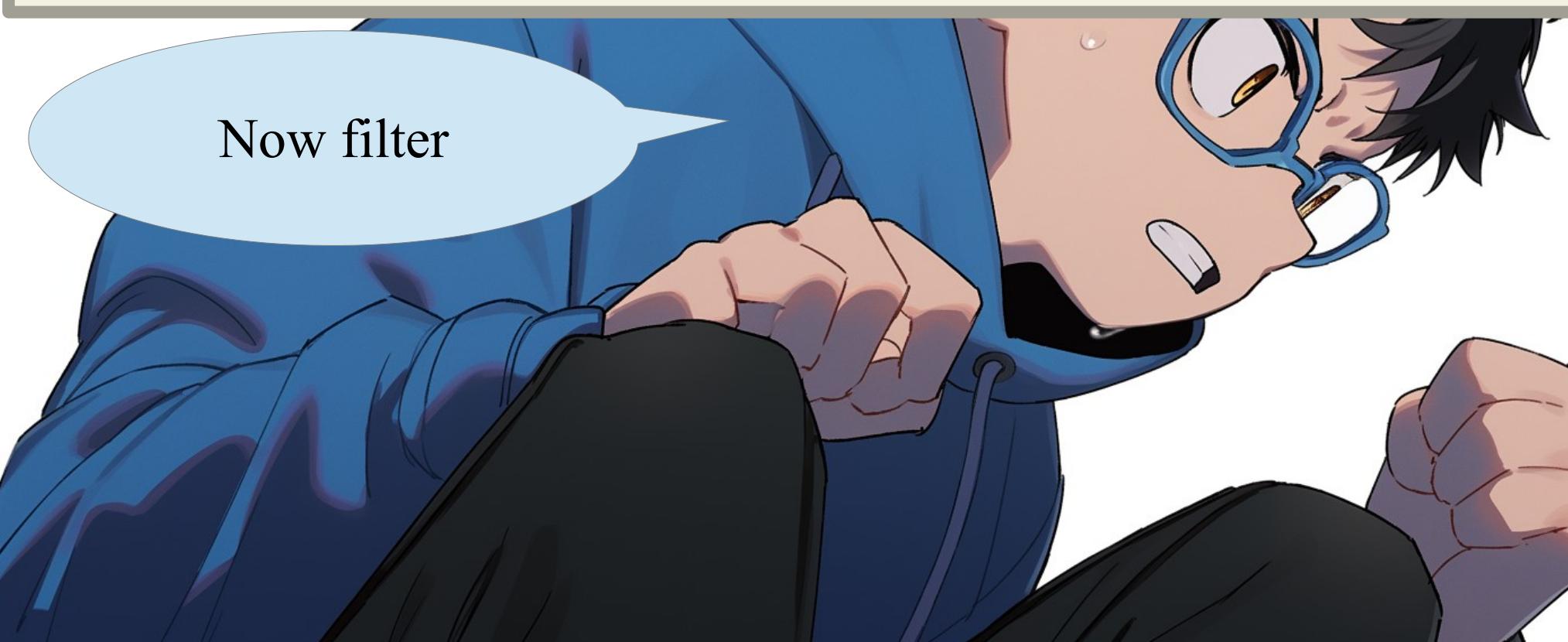
```
public sealed interface Optional<T> permits Empty<T>, Some<T>{  
    Optional<T> filter(Predicate<T> p);  
}
```



Now filter

```
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; }
}
```

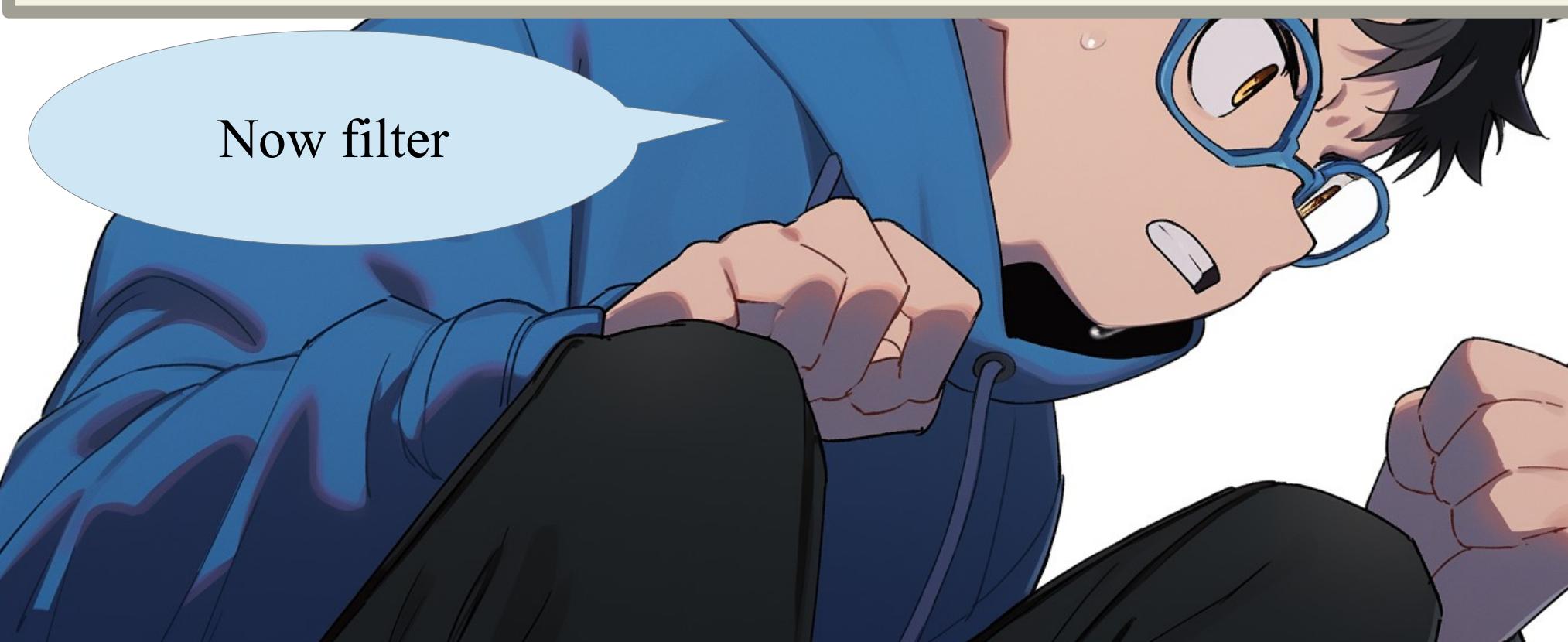


Now filter

```
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();
    }
}
```



Now filter

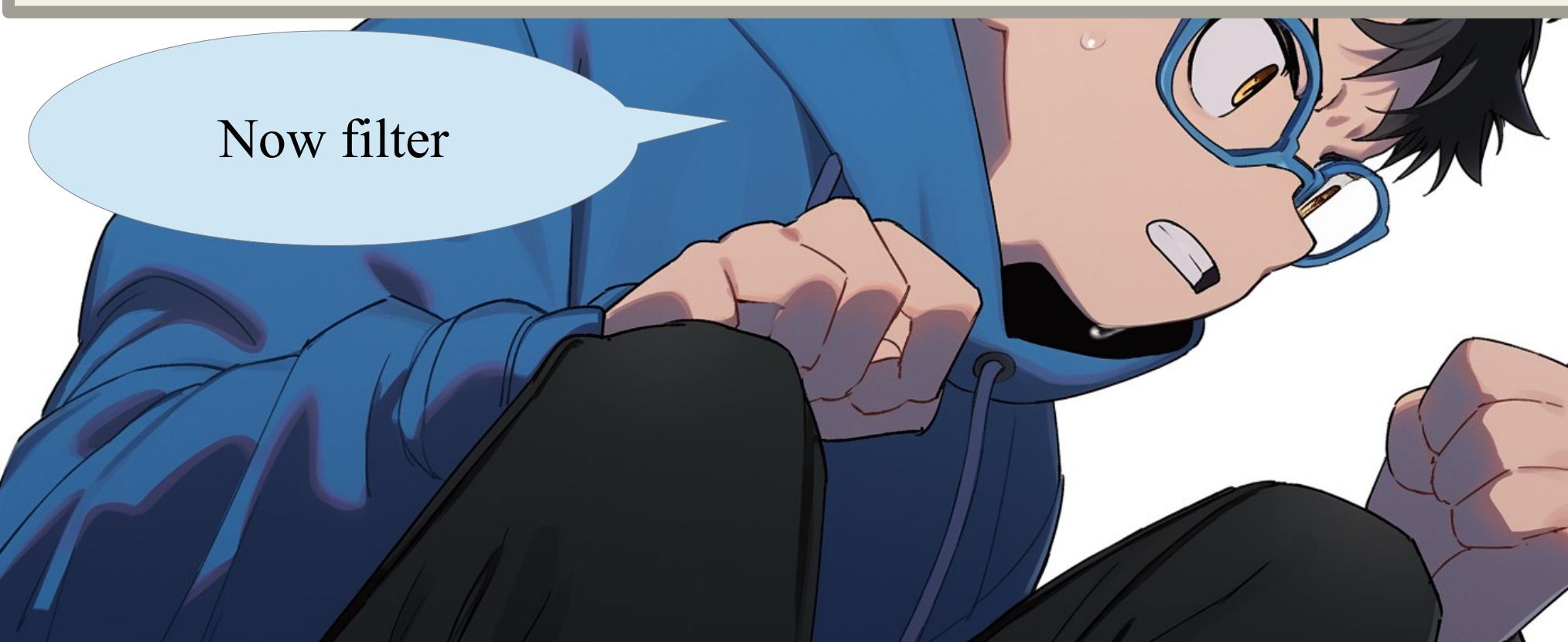
```
public sealed interface Optional<T> permits Empty<T>, Some<T> {
    Optional<T> filter(Predicate<T> p);
}

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



Now filter

```
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

Now filter

```
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

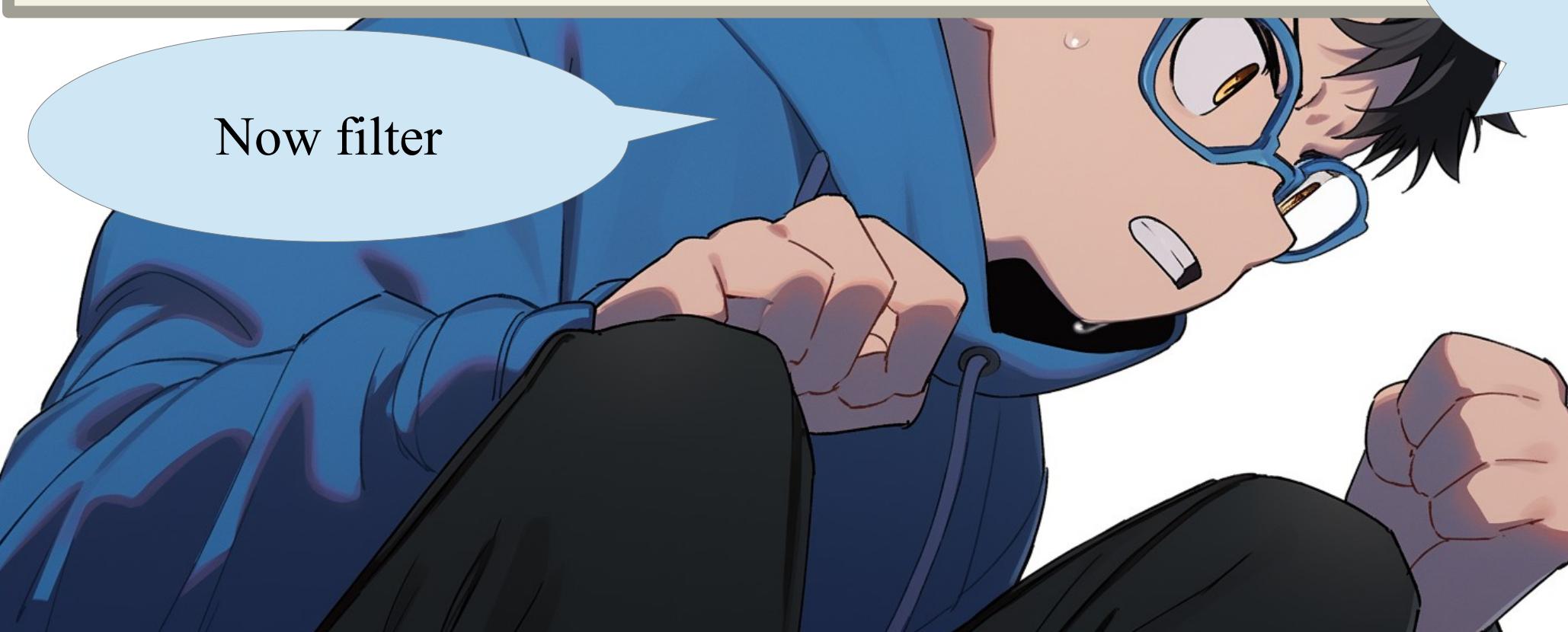


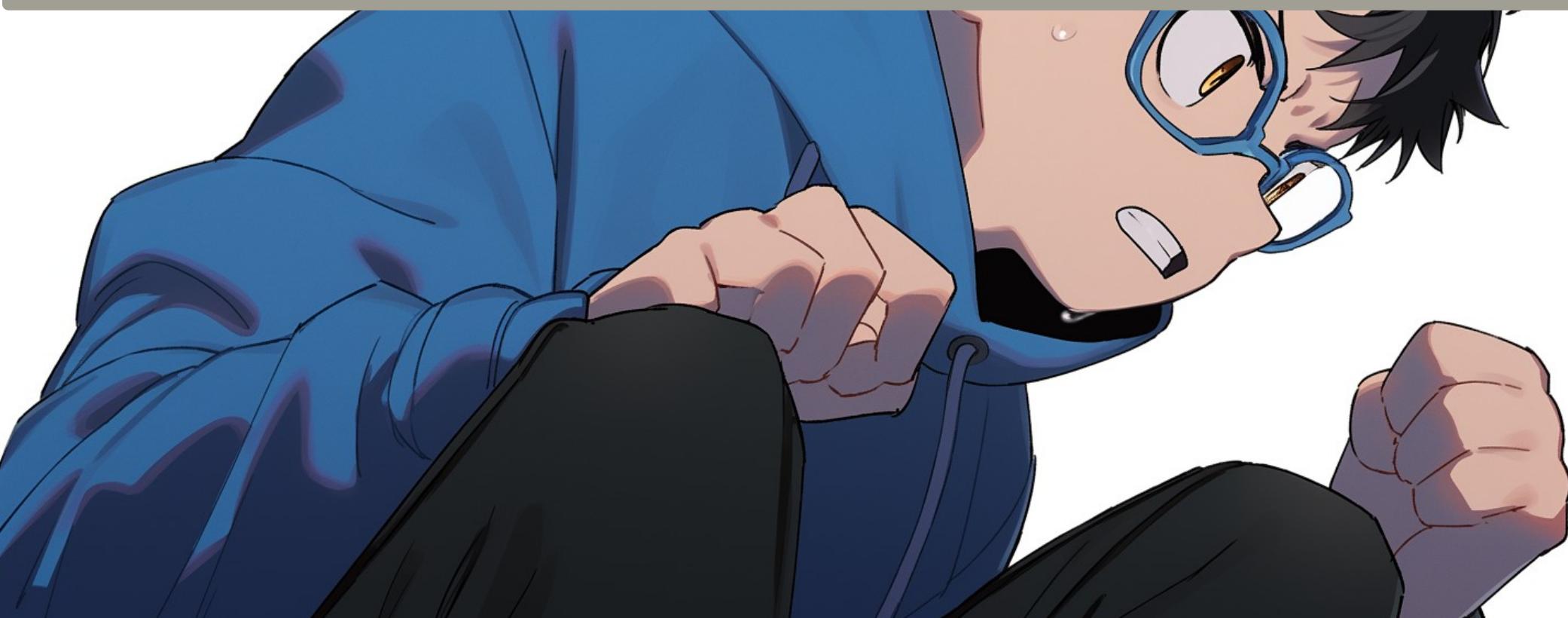
On an empty 'this', returns empty

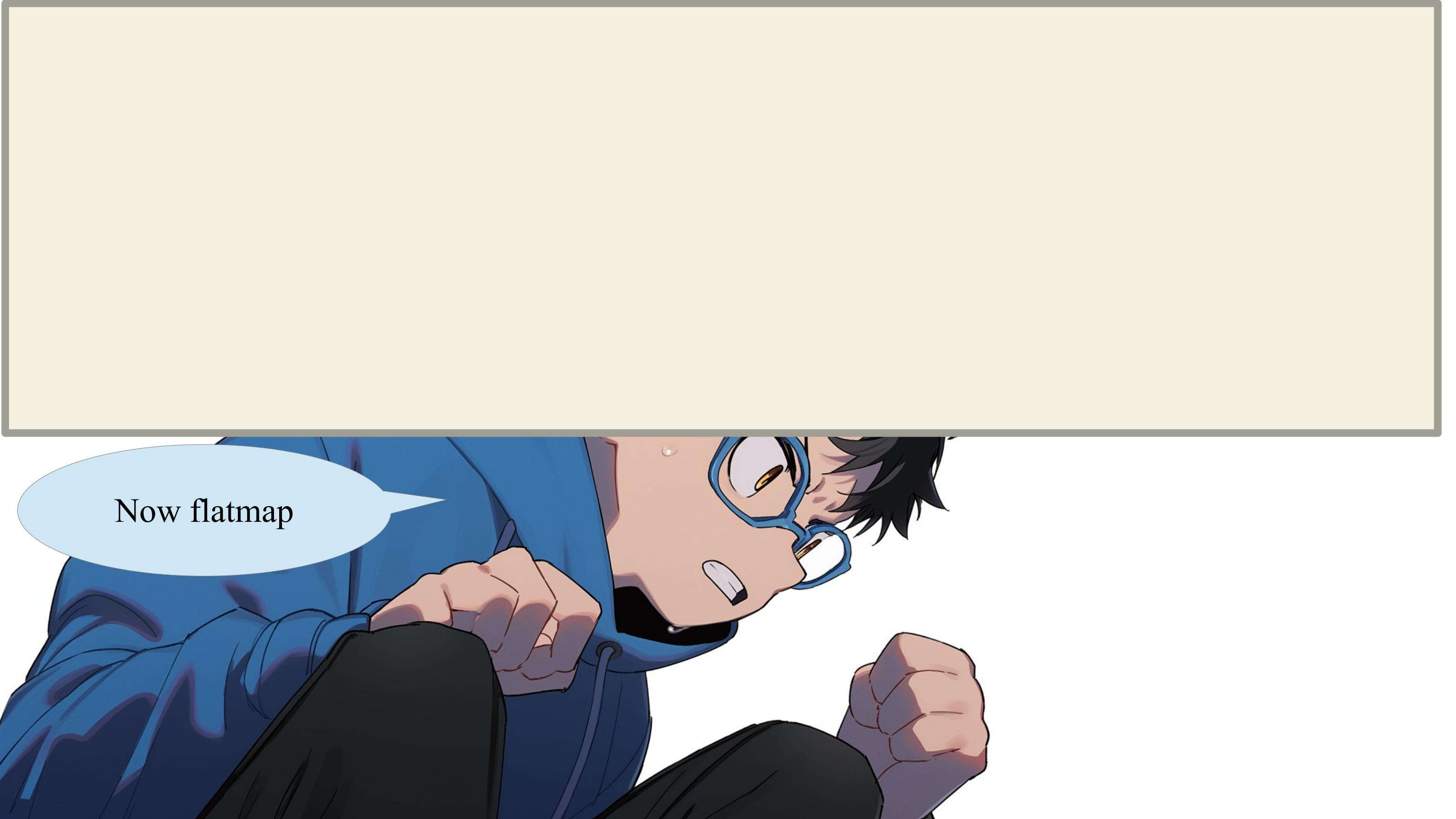


When there is a value,  
return the same optional  
if the predicate passes.  
Otherwise, empty

Now filter







Now flatmap

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



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

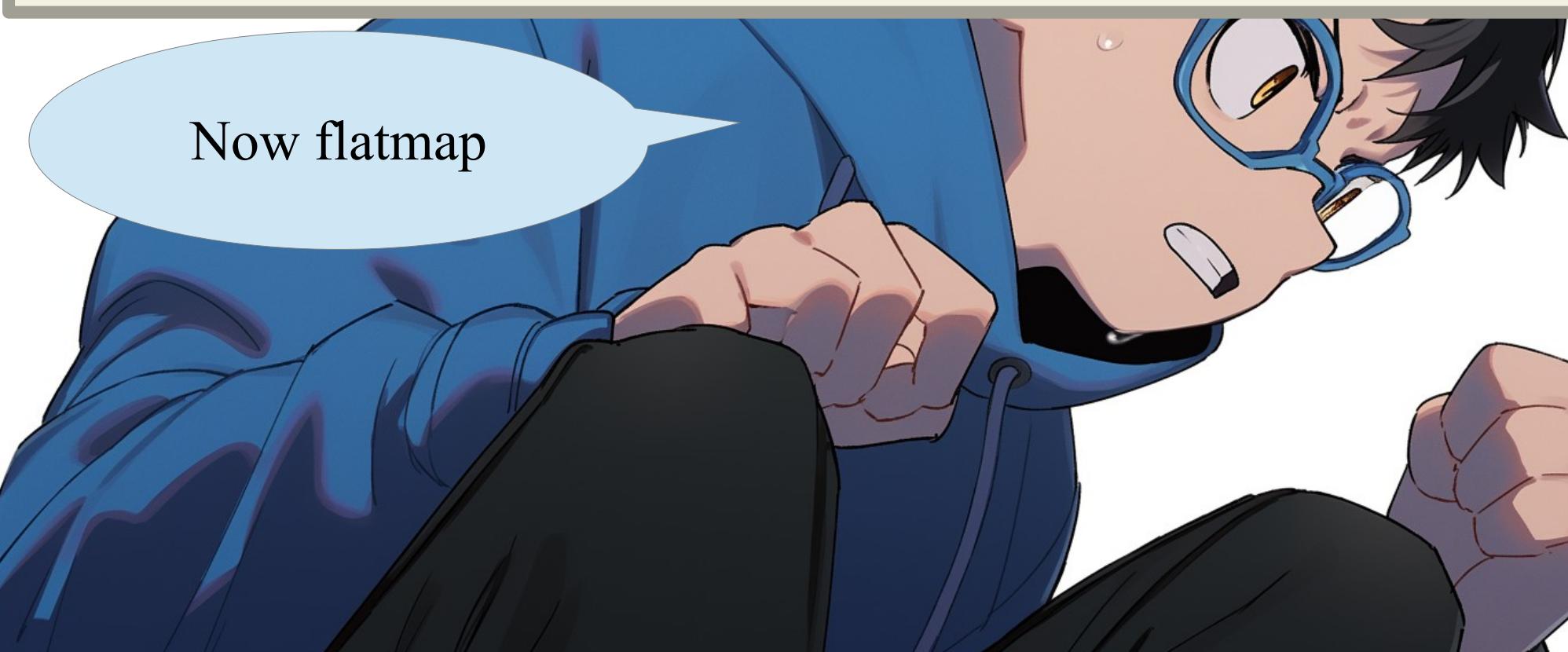
final class Empty<T> implements Optional<T>{
    public <R> Optional<R> flatMap(Function<T, Optional<R>> m) {
        return Optional.empty();
    }
}
```



```
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);
    }
}
```

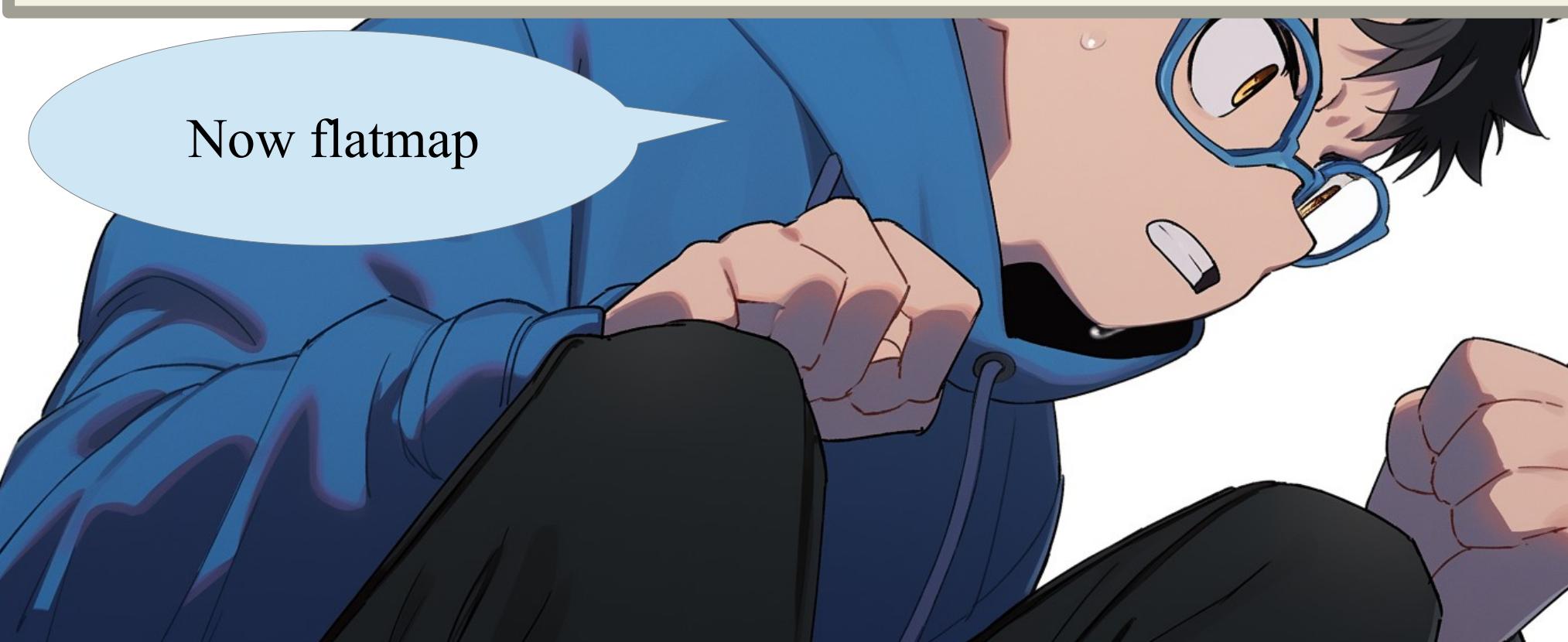


Now flatmap

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

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);
    }
}
```



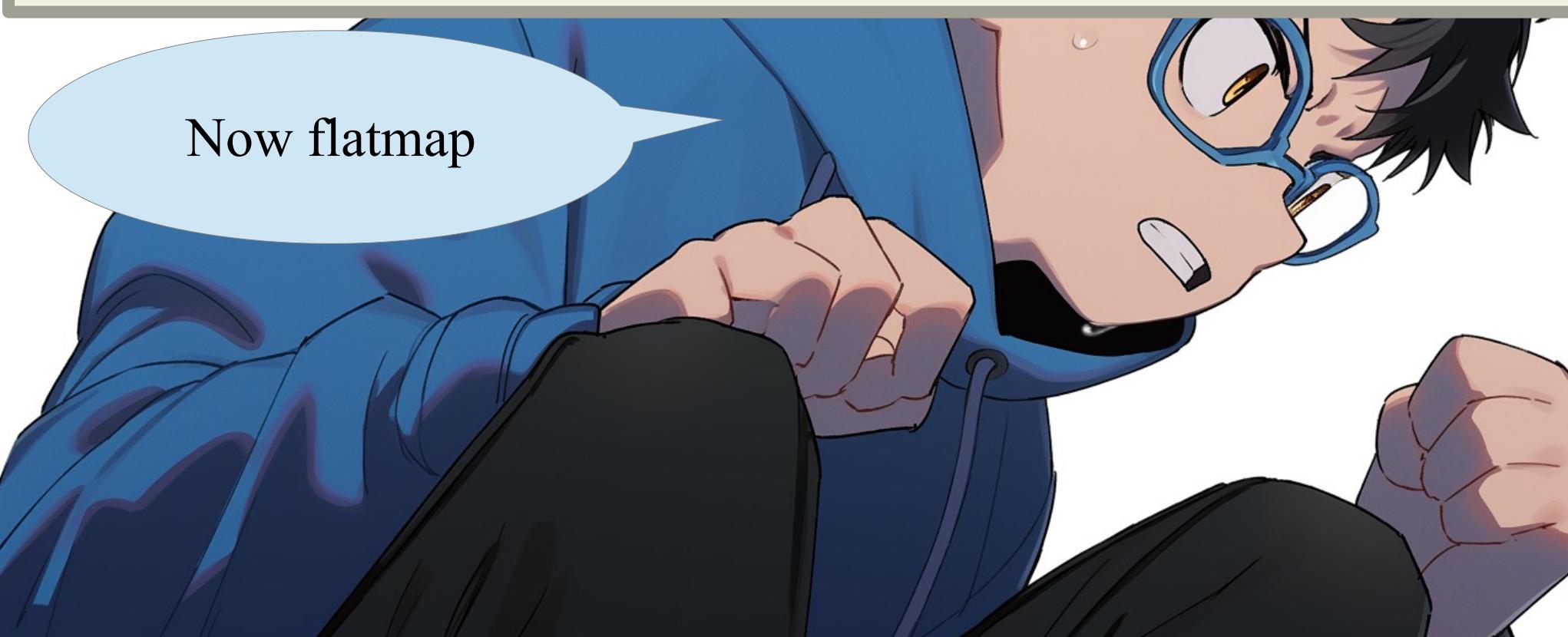
Now flatmap

```
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

Now flatmap



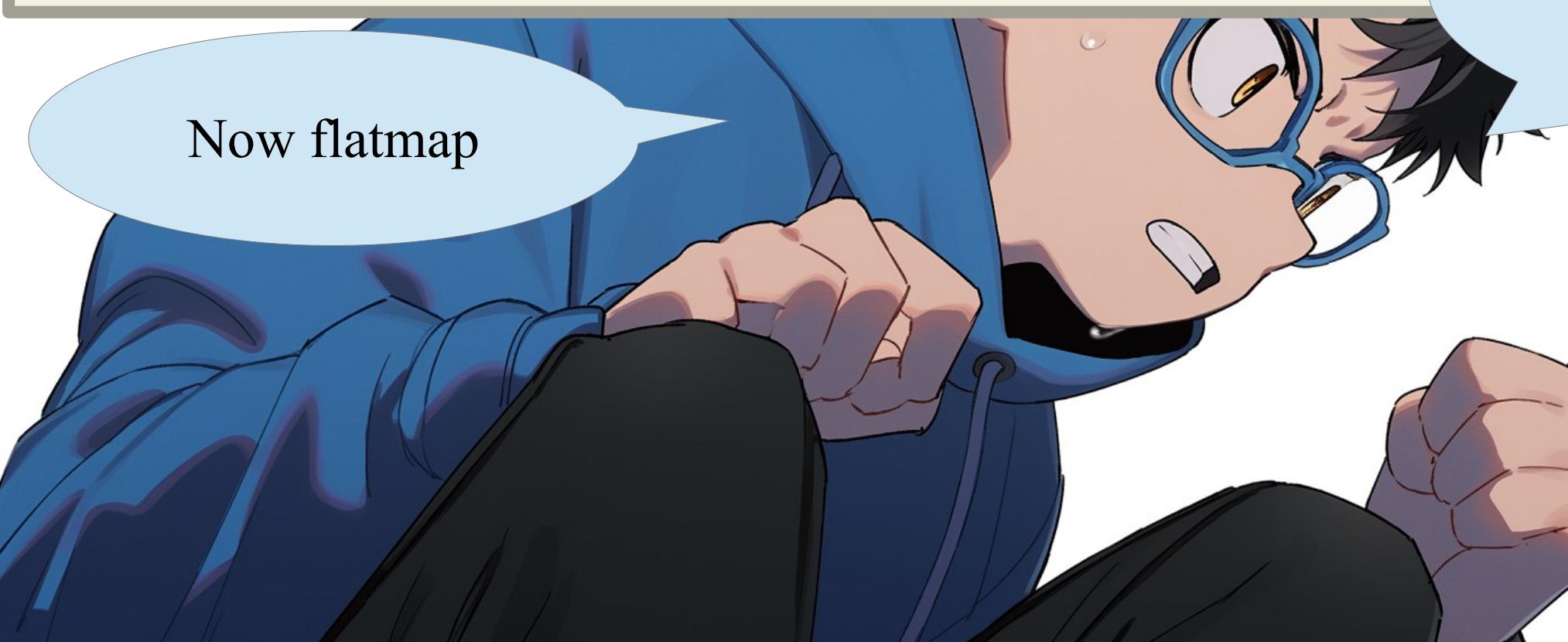
```
public sealed interface Optional<T> permits Empty<T>, Some<T> {
    <R> Optional<R> flatMap(Function<T, Optional<R>> m); } final class Empty<T> implements Optional<T> {
    public <R> Optional<R> flatMap(Function<T, Optional<R>> m) {
        return Optional.empty(); } record Some<T>(T get) implements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

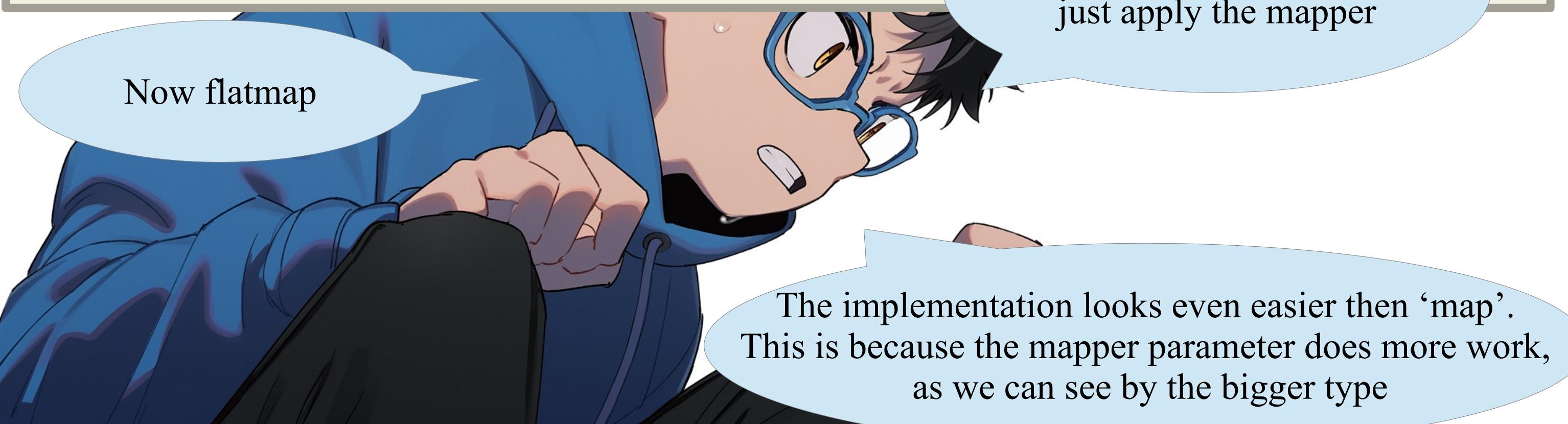
Now flatmap



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

final class Empty<T> implements Optional<T> {
    public <R> Optional<R> flatMap(Function<T, Optional<R>> m) {
        return Optional.empty(); } } ← On an empty 'this', returns empty

record Some<T>(T get) implements Optional<T> {
    public <R> Optional<R> flatMap(Function<T, Optional<R>> m) {
        return m.apply(get); } } ← When there is a value,
                                just apply the mapper
```







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'*

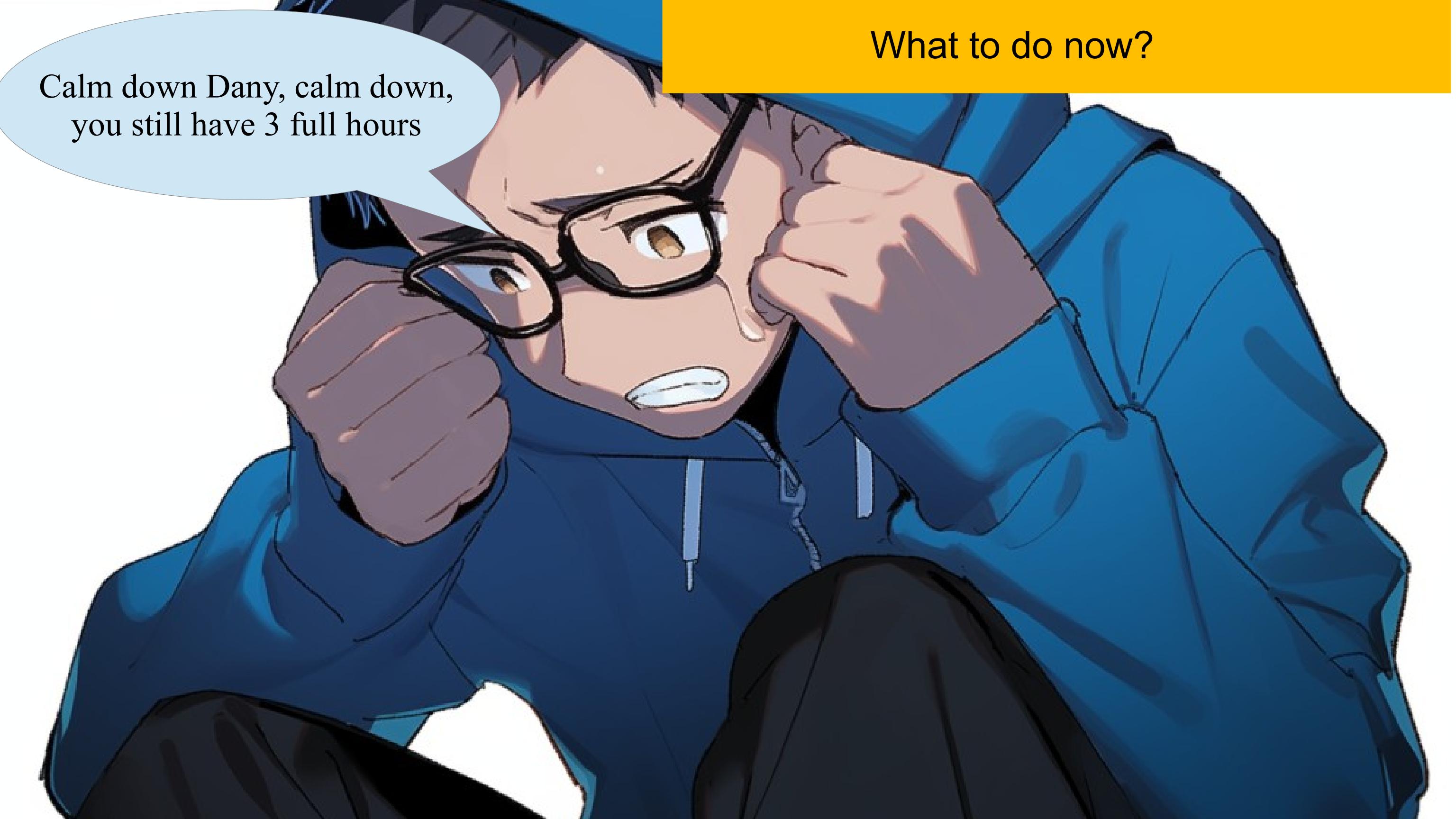
Dang, the submission is not accepted!

*You are now  
down to 60%*



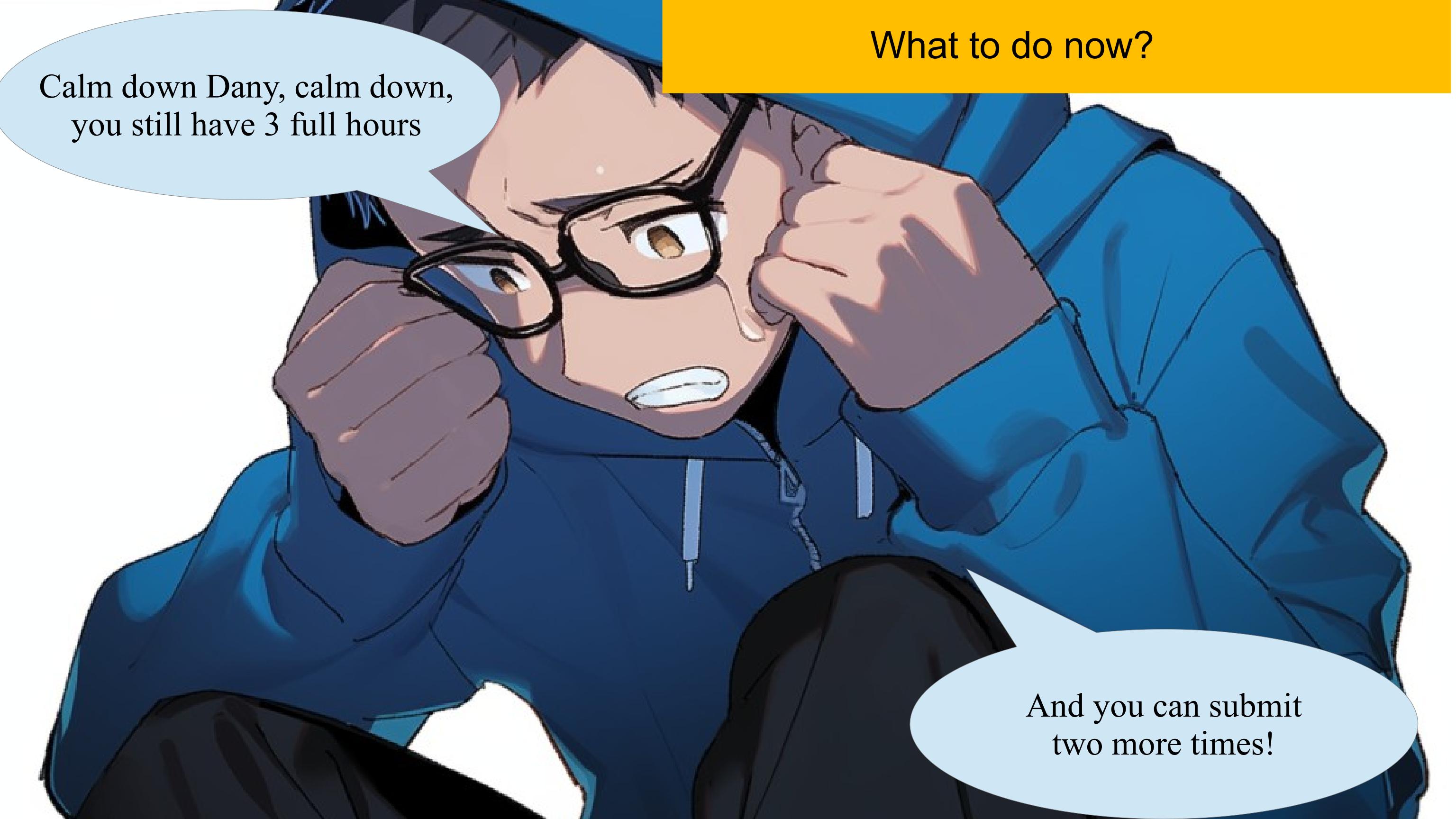


What to do now?



What to do now?

Calm down Dany, calm down,  
you still have 3 full hours

A close-up illustration of a man with short brown hair and glasses, wearing a blue zip-up hoodie. He has a distressed expression, with his hands clasped behind his head. The background is split into two colors: light blue on the left and yellow on the right.

What to do now?

Calm down Dany, calm down,  
you still have 3 full hours

And you can submit  
two more times!





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





Who I'm I kidding!



A close-up shot from an anime featuring two characters. On the left, a male character with dark blue hair and glasses has a determined expression, his fist clenched. On the right, another character with dark hair and a blue jacket with white stripes is shown in profile, looking towards the left character. A speech bubble originates from the left character's mouth.

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

A close-up shot of two anime characters. On the left, a male character with dark blue hair and glasses has a determined expression, his fist clenched. On the right, another male character with short dark hair and a blue jacket with white stripes on the shoulders looks slightly off-camera. Two light blue speech bubbles are positioned above them. The top bubble contains the first character's dialogue, and the bottom bubble contains the second character's.

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





An anime-style illustration of a character with dark hair and green eyes wearing glasses and a blue and white striped beanie. The character has a distressed expression and is looking upwards. A light blue speech bubble originates from their mouth.

I'm so desperate  
I can not move

An anime-style illustration of a character with dark hair and glasses, wearing a blue jacket and a blue and white striped hat. The character is looking upwards with a distressed expression. A light blue speech bubble originates from their mouth.

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 than  
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

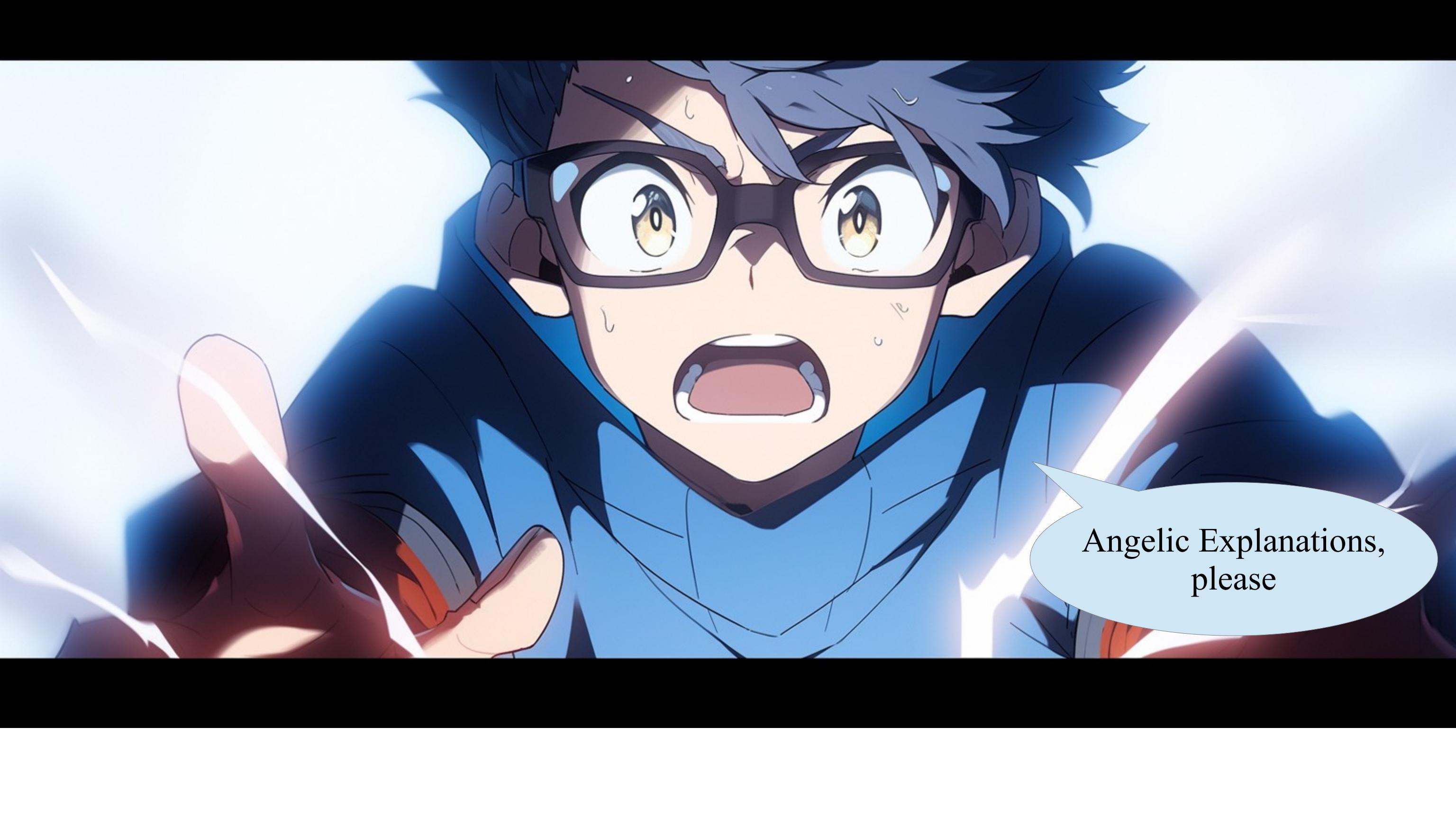
But, I had no idea that  
Optionals had an  
'or' method at all!





Can I figure it out  
with just reasoning?





Angelic Explanations,  
please



Answer to my call again!



An anime-style illustration of a young man with dark blue hair and glasses. He has a shocked expression, with wide eyes and an open mouth. He is wearing a blue t-shirt with a small orange knot at the collar. A large amount of bright blue energy or lightning surrounds him, particularly on his arms and chest. A white speech bubble originates from the top left, pointing towards his head.

The only 'or' I know  
is from booleans.

An anime-style illustration of a young man with dark blue hair and glasses. He has a shocked expression, with wide eyes and an open mouth. A single tear is falling from his left eye. He is wearing a blue t-shirt with a small orange knot at the collar. The background is a bright, overexposed white with blue energy or lightning bolts visible around him.

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

An anime-style illustration of a young man with spiky blue hair and black-rimmed glasses. He has a wide-eyed, shocked expression with his mouth slightly open. His hands are raised near his head, palms facing outwards. The background is a soft-focus blue.

The only 'or' I know  
is from booleans.

It takes two booleans

tooCold



or



tooHot

sadClimate



tooCold



or



tooHot

sadClimate



tooCold



or



tooHot



sadClimate

It produces a boolean result