
To override the content of a list completely,
we can use listIterator.next and listIterator.set

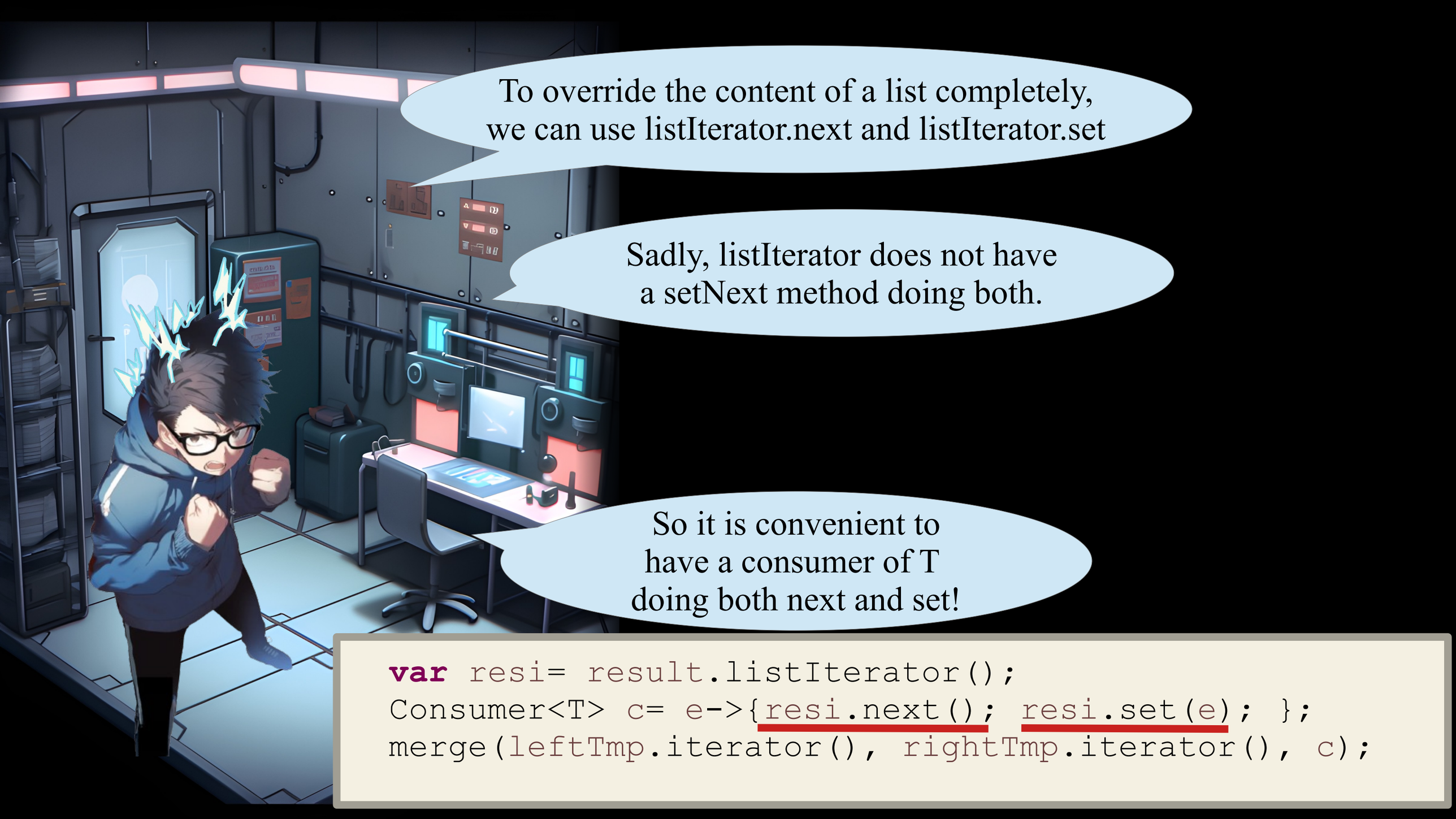
To override the content of a list completely,
we can use listIterator.next and listIterator.set

Sadly, listIterator does not have
a setNext method doing both.

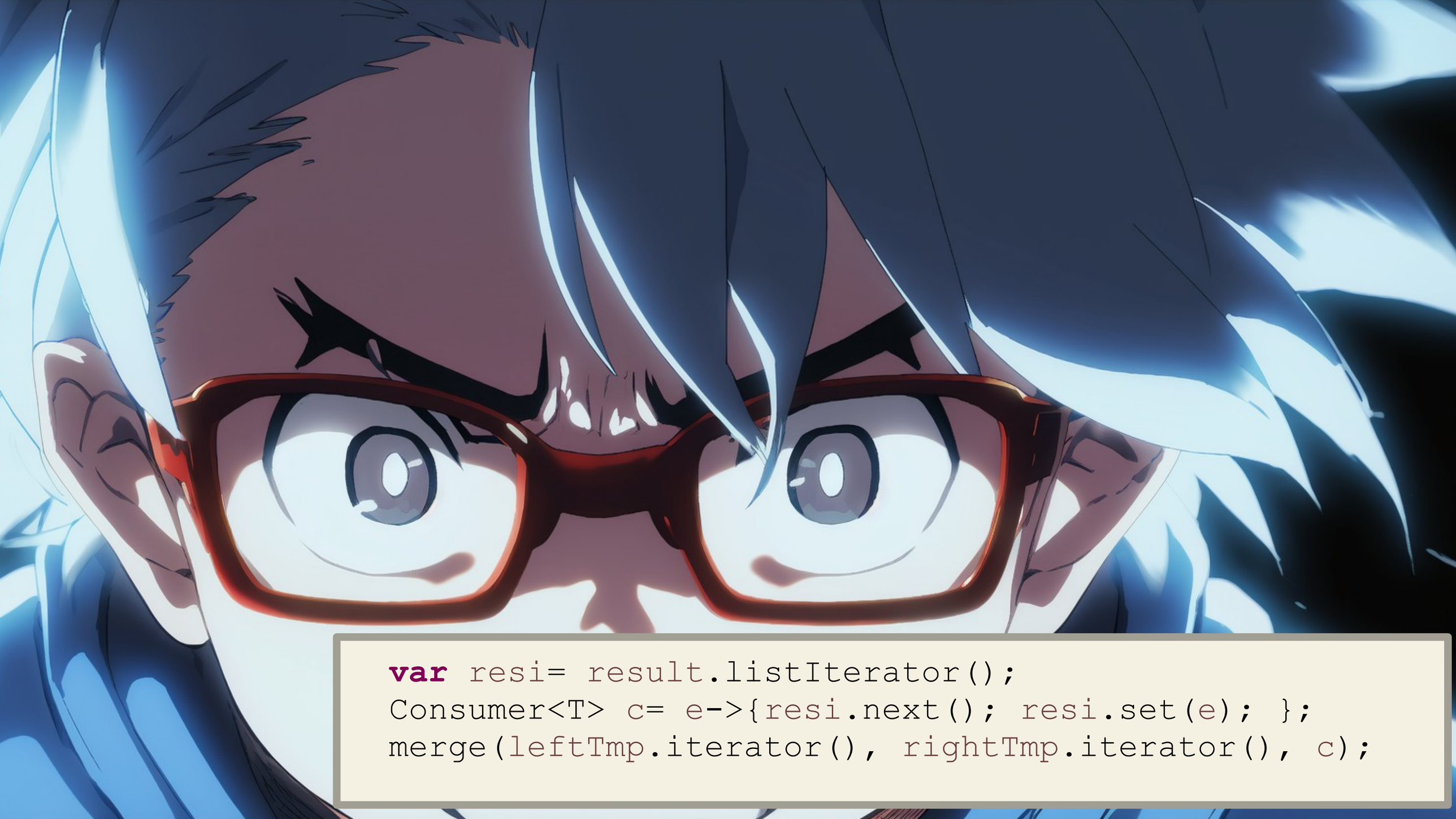
To override the content of a list completely,
we can use listIterator.next and listIterator.set

Sadly, listIterator does not have
a setNext method doing both.

So it is convenient to
have a consumer of T
doing both next and set!



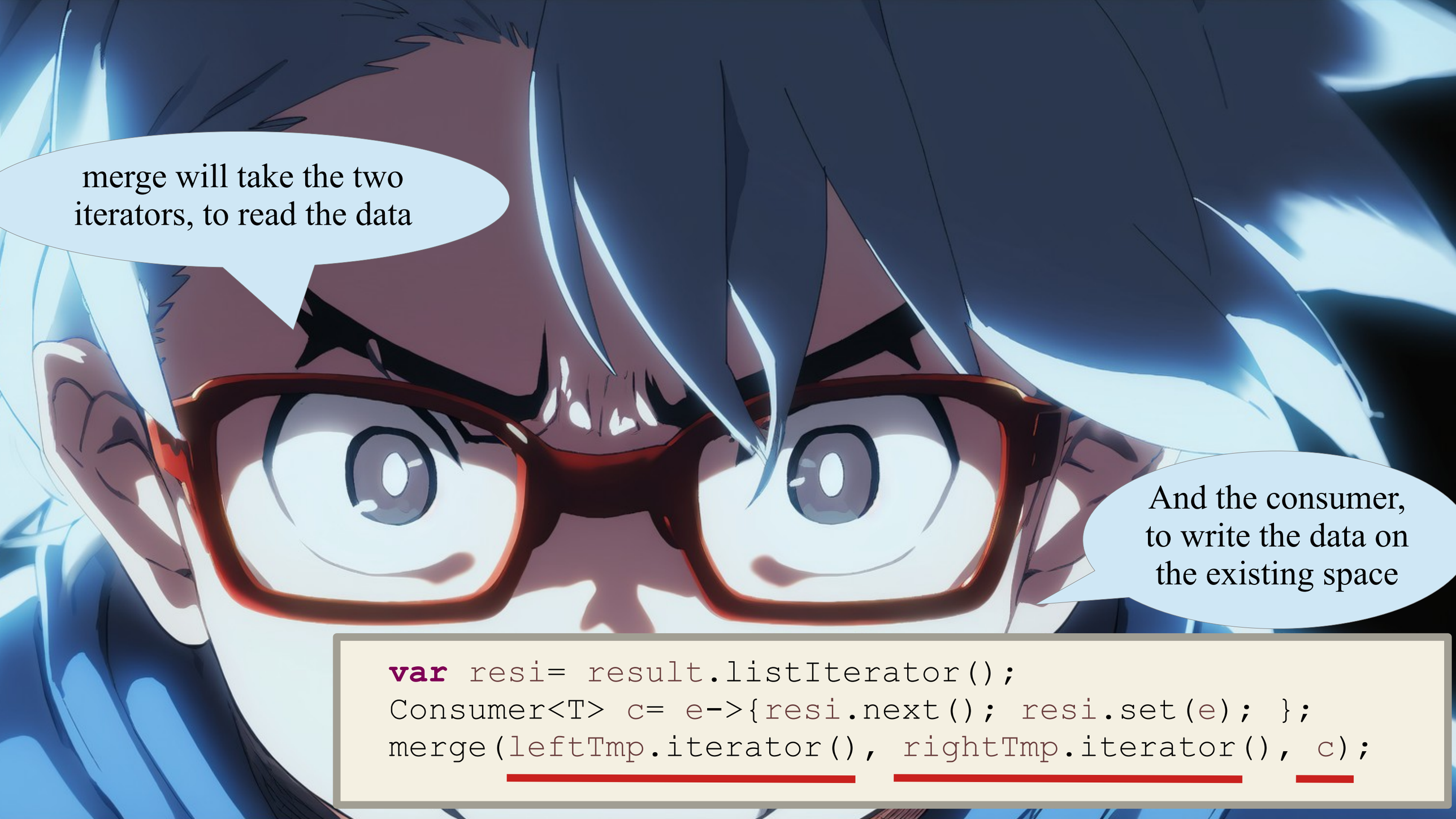

```
var resi= result.listIterator();
Consumer<T> c= e->{resi.next(); resi.set(e); };
merge(leftTmp.iterator(), rightTmp.iterator(), c);
```



```
var resi= result.listIterator();
Consumer<T> c= e->{resi.next(); resi.set(e); };
merge(leftTmp.iterator(), rightTmp.iterator(), c);
```


```
var resi= result.listIterator();
Consumer<T> c= e->{resi.next(); resi.set(e); };
merge(leftTmp.iterator(), rightTmp.iterator(), c);
```


```
var resi= result.listIterator();
Consumer<T> c= e->{resi.next(); resi.set(e); };
merge(leftTmp.iterator(), rightTmp.iterator(), c);
```




```
var resi= result.listIterator();
Consumer<T> c= e->{resi.next(); resi.set(e); };
merge(leftTmp.iterator(), rightTmp.iterator(), c);
```

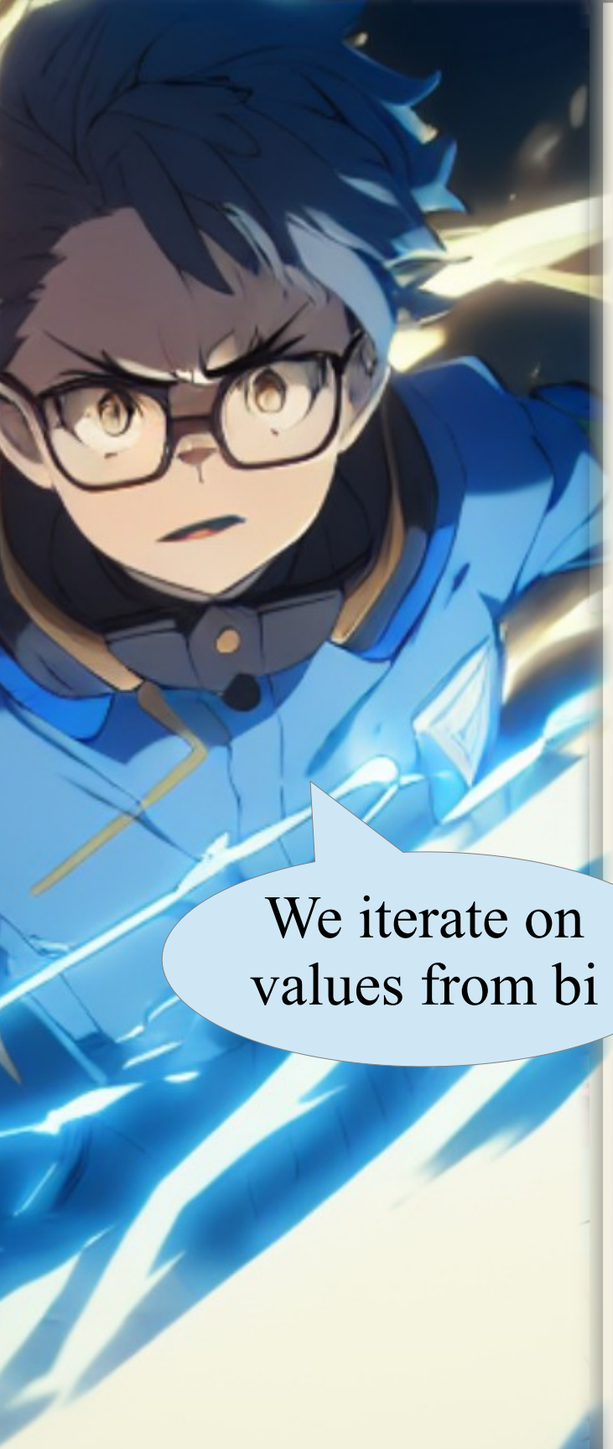
```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```


```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```

```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```

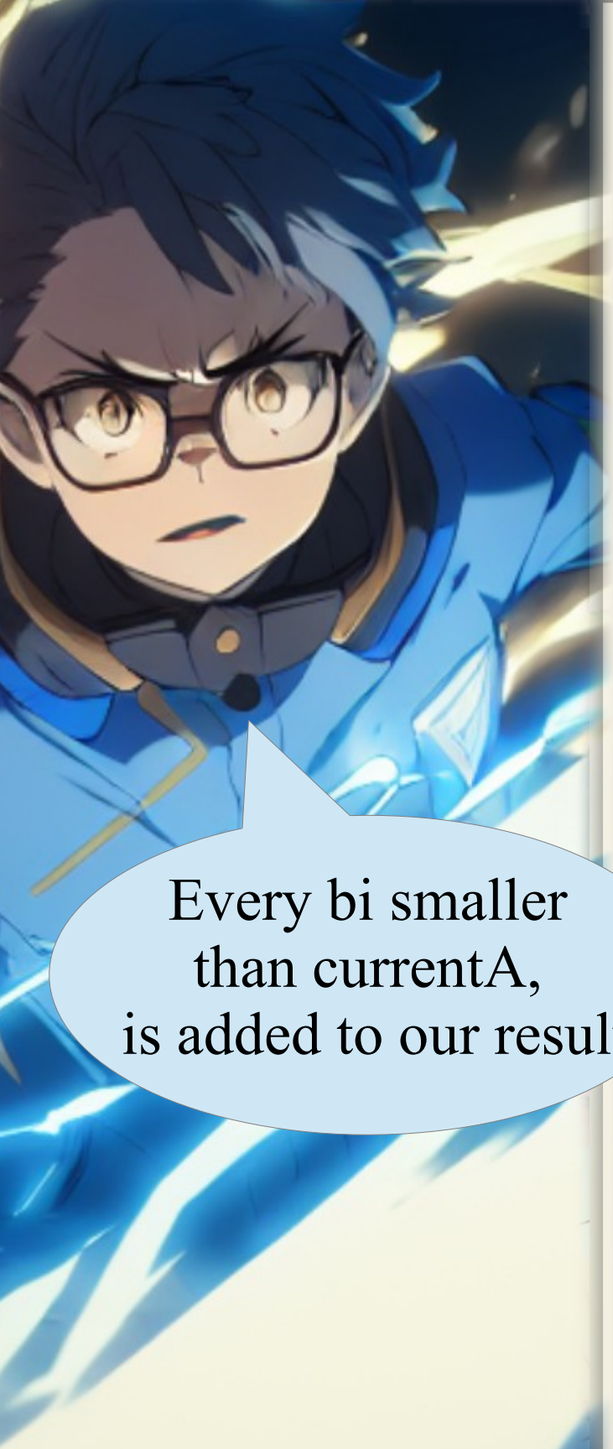

```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```

```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```




```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```

```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```



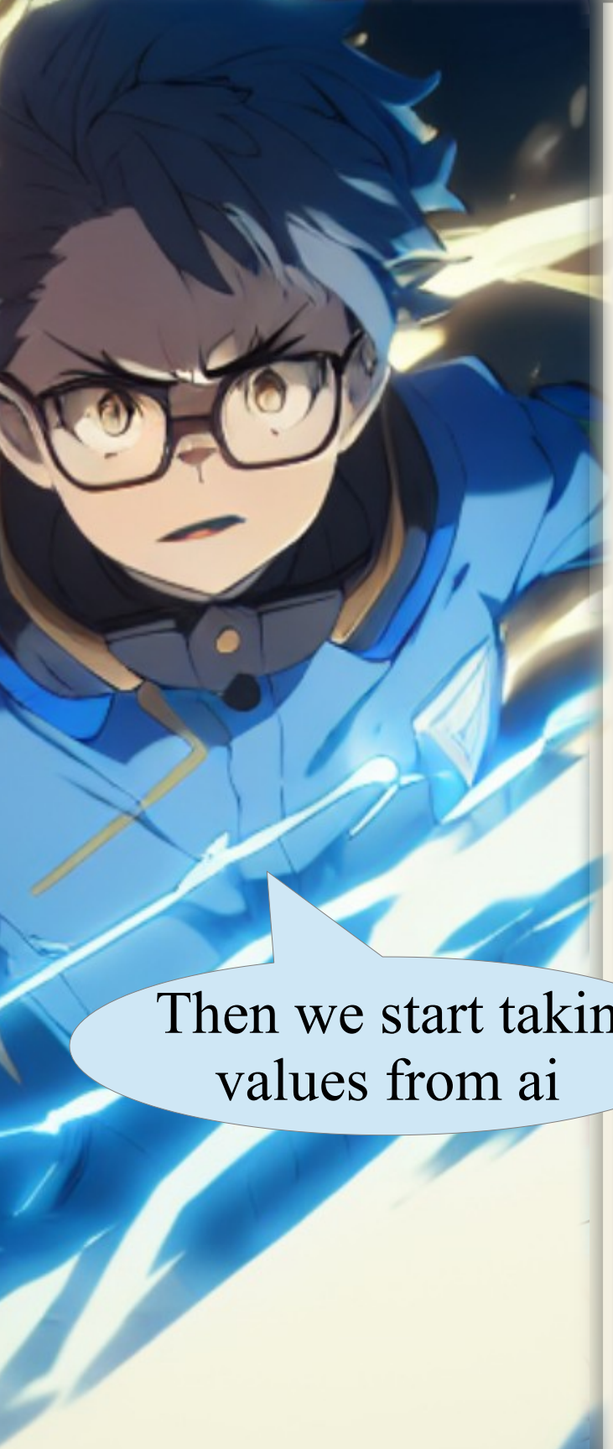

```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```

```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```

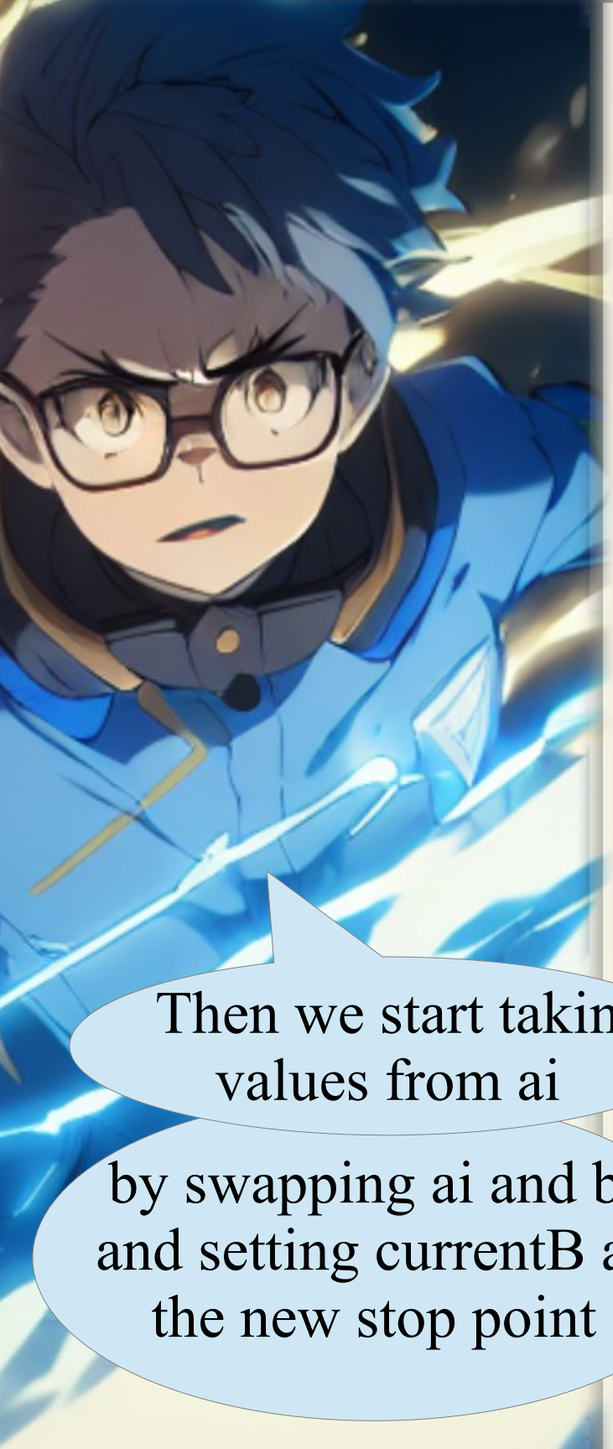

```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```


```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```

```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```




```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```




```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;

  res.accept(currentA);
  ai.forEachRemaining(res);
}
```

```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```


```java
private void sortHelper(List<T> tmp, List<T> result){
  int size= tmp.size();
  if (size < 2) { return; }
  int half = size / 2;
  List<T> leftTmp    = tmp.subList(0, half);
  List<T> rightTmp   = tmp.subList(half, size);
  List<T> leftResult = result.subList(0, half);
  List<T> rightResult= result.subList(half, size);
  sortHelper(leftResult, leftTmp);//swap result/tmp
  sortHelper(rightResult, rightTmp);//swap result/tmp
  var resi= result.listIterator();
  Consumer<T> c= e->{resi.next(); resi.set(e); };
  merge(leftTmp.iterator(), rightTmp.iterator(), c);
}
private void merge(Iterator<T> ai, Iterator<T> bi, Consumer<T> res){
  T currentA= ai.next();
  while (bi.hasNext()){
    T currentB= bi.next();
    var bSmaller= currentB.compareTo(currentA) <= 0;
    if (bSmaller){ res.accept(currentB); continue; }
    res.accept(currentA);
    Iterator<T> tmp = ai; ai = bi; bi = tmp;
    currentA = currentB;
  }
  res.accept(currentA);
  ai.forEachRemaining(res);
}
```

It is the moment of truth again

# It is the moment of truth again


Congratulations!

Congratulations!

You have managed to submit an answer for one of the two exam questions!

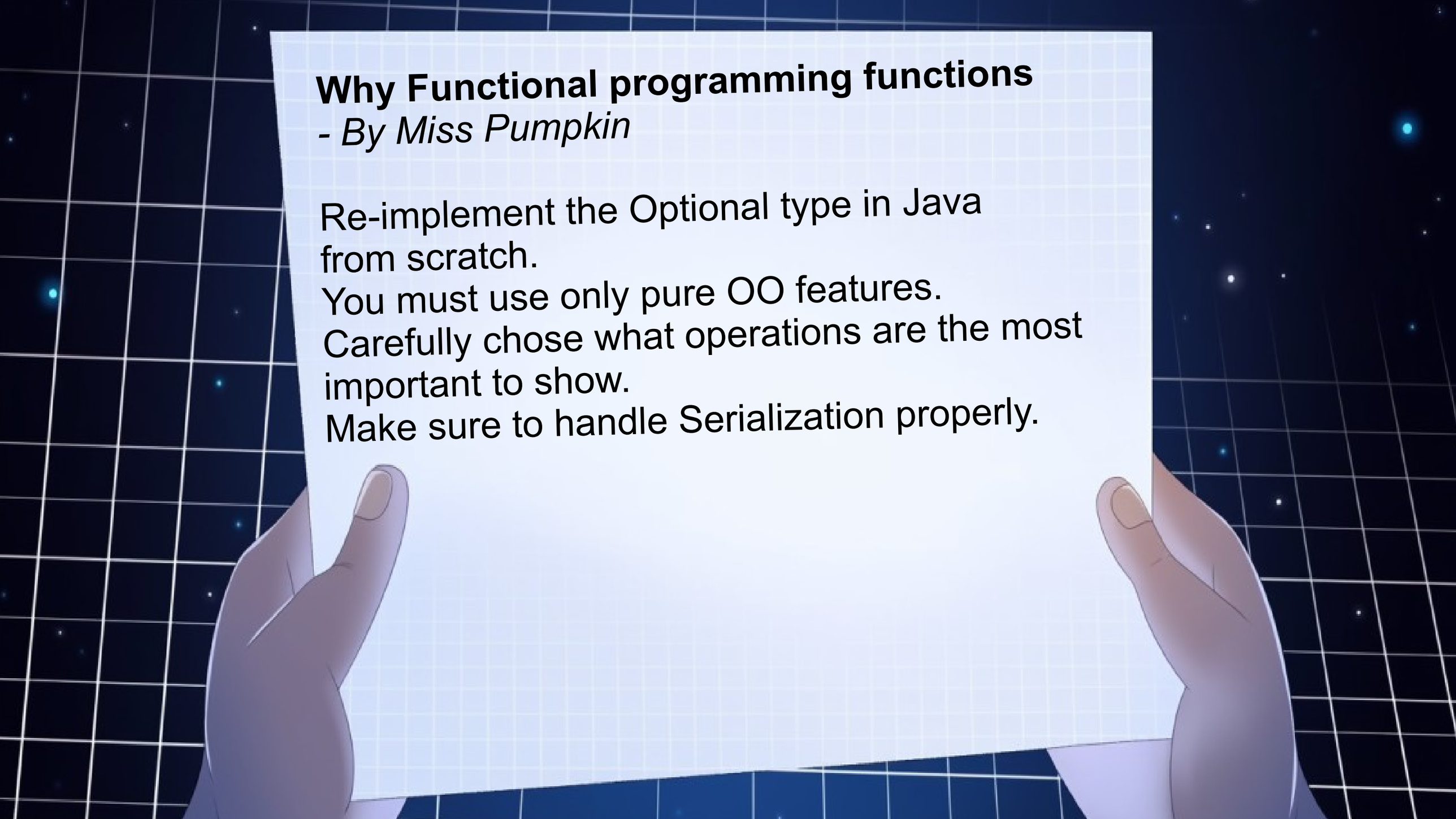

Hurry up and submit the second one!


One is done!


One is done!
Now I can focus on the other one!


And I still have
about 4 hours left!


And I still have about 4 hours left!
What was the other question about?


**Why Functional programming functions**
*- By Miss Pumpkin*
Re-implement the Optional type in Java from scratch.
You must use only pure OO features.
Carefully chose what operations are the most important to show.
Make sure to handle Serialization properly.

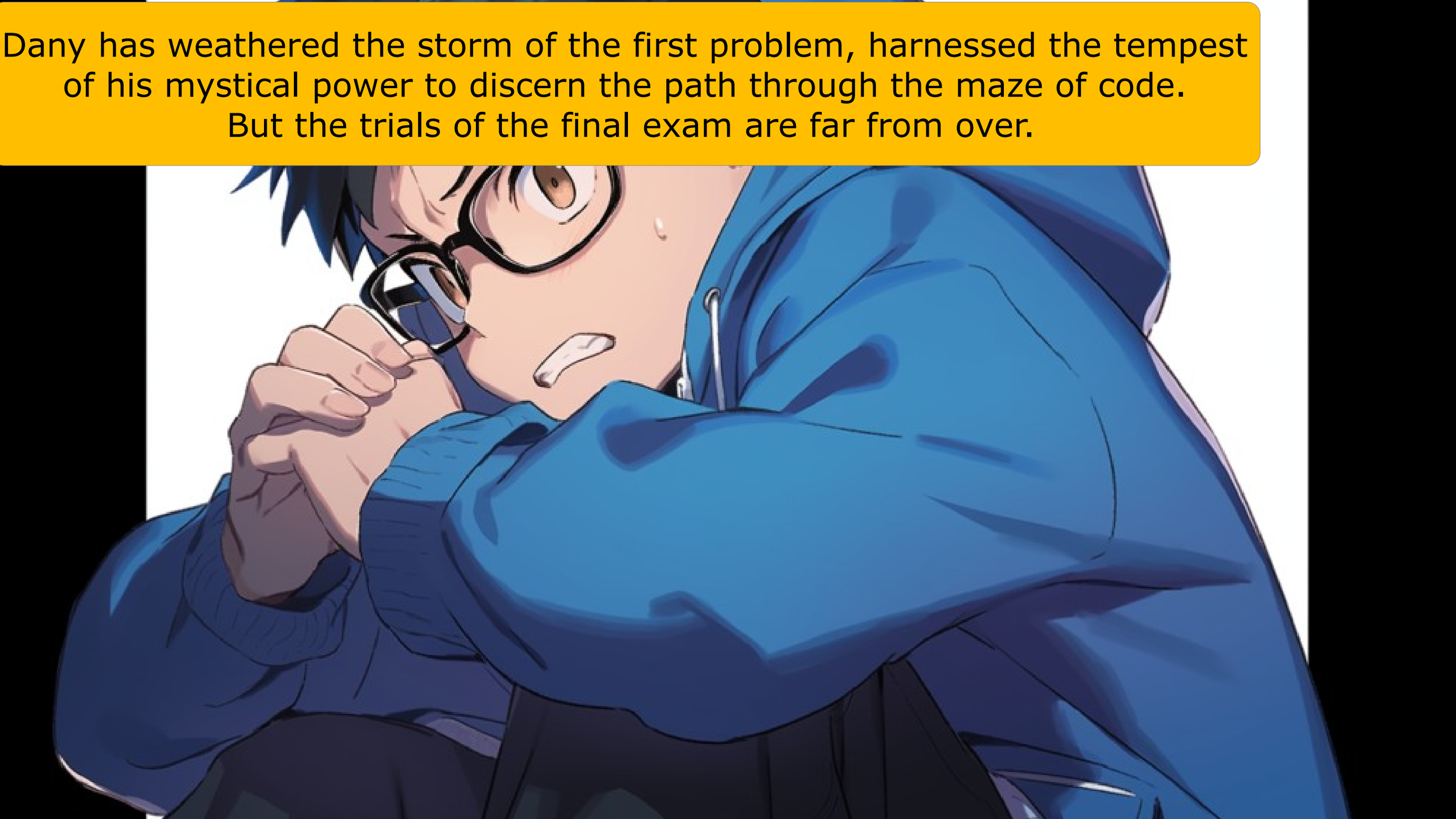

I would have no idea how to answer it without my power


But I have already used my power to answer the first question

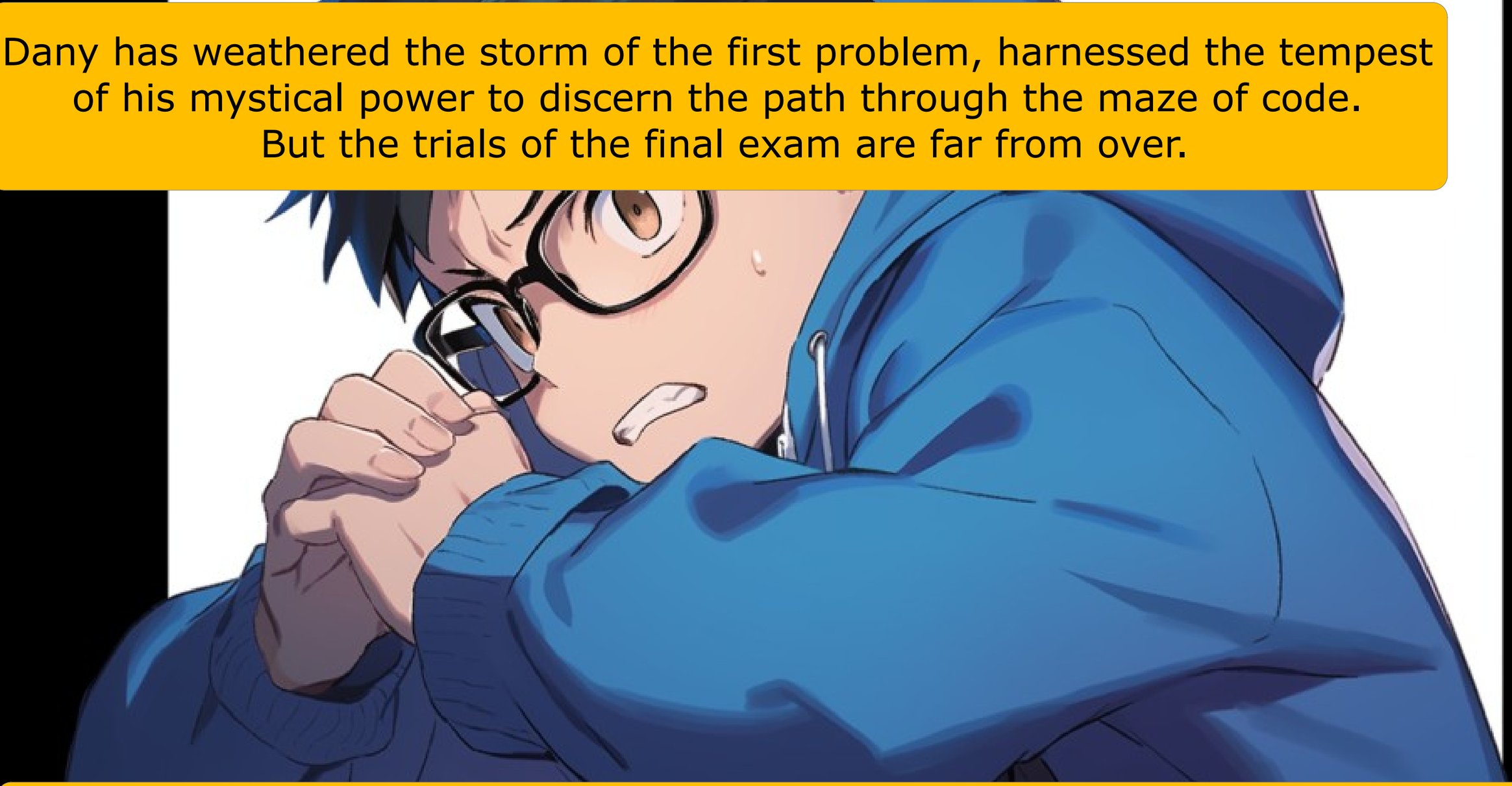

Luckily I’ve been able to trip into my power twice in a row lately!


Will I unlock the secrets of the next problem, or stumble into a pitfall of errors?


Will I unlock the secrets of the next problem, or stumble into a pitfall of errors?
Let’s do it!
Time to start question two!


Dany has weathered the storm of the first problem, harnessed the tempest of his mystical power to discern the path through the maze of code.
But the trials of the final exam are far from over.

Dany has weathered the storm of the first problem, harnessed the tempest of his mystical power to discern the path through the maze of code.
But the trials of the final exam are far from over.



Question two may be a formidable beast yet to be tackled
His unique ability, his secret weapon, flickers uncertainly within him.
Will Dany manage to answer question two with just one shot of his power left?

The code of destiny awaits.
With just a single shot of his power left, will Dany manage to triumph
over the ominous question two and reach the ivory tower?
Stay tuned for the next chapter of our code warrior's journey into the heart of Java!


Credits
- Story: Marco
- Art: MidJourney, NijiJourney, Dall-E
- Wording: Marco, chatGPT, jfw01
- Composition: Marco
- Thanks to all my friends for providing great feedback!