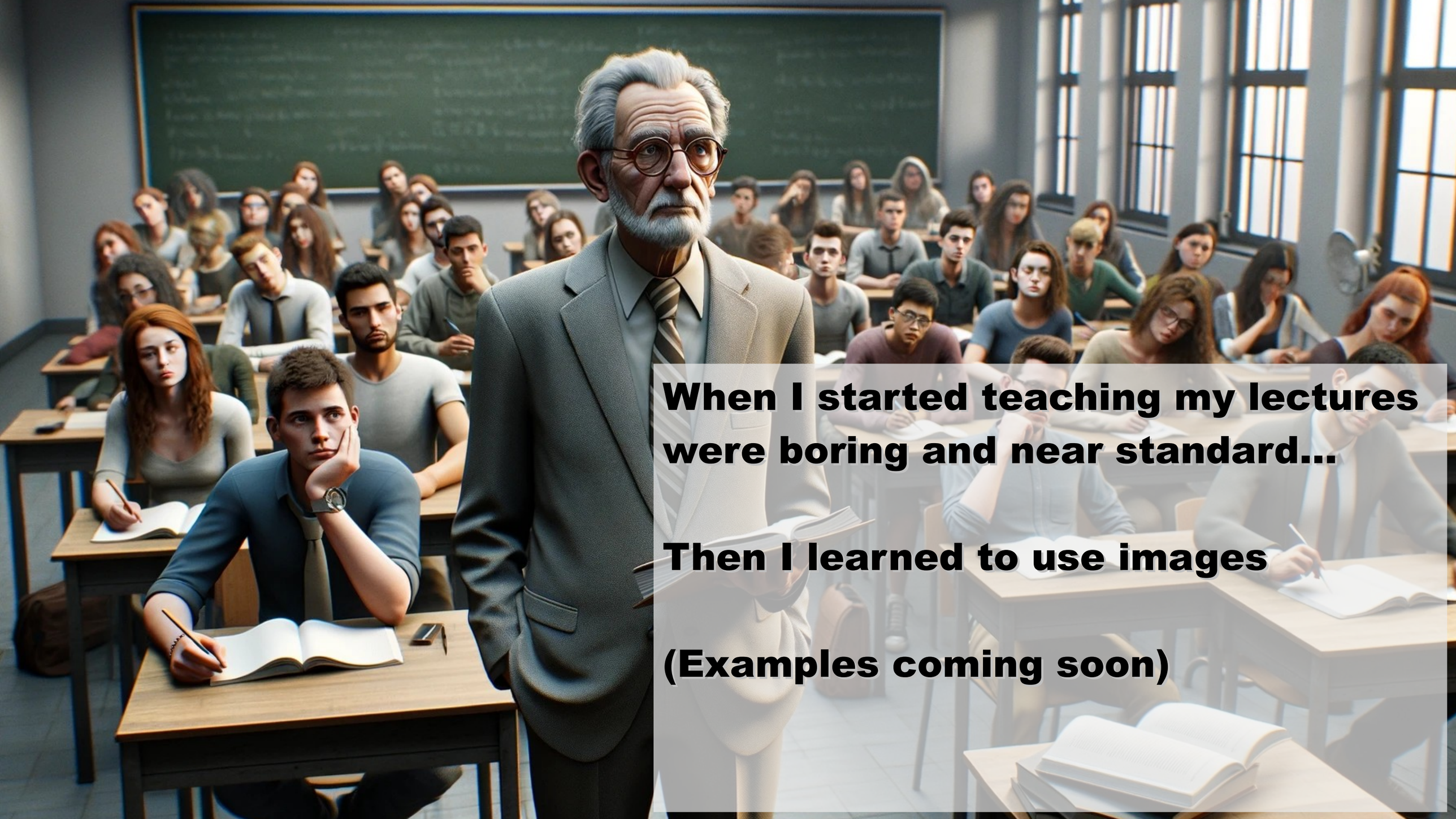
Marco Servetto

Victoria University of Wellington

New Zealand

My Teaching Journey.
How I tech challenging concepts


When I started teaching my lectures were boring and near standard...
Then I learned to use images
(Examples coming soon)


Java

Java is now an old language(1996)

But, recently…..

(after training in secret for 20 years)

Java: now a modern language!

Now with

- Records,
- Interfaces with default methods
- Lambdas

- We will use those features very intensively in this course!
- We will use ‘class’ more rarely.

Hint!

When searching for Java examples, Java documentation or Java help

- Never search for “Java”
  you will get the old stuff.

- Search for “Java 21”, “Java 22”
  or “Java 2024” to get good stuff!

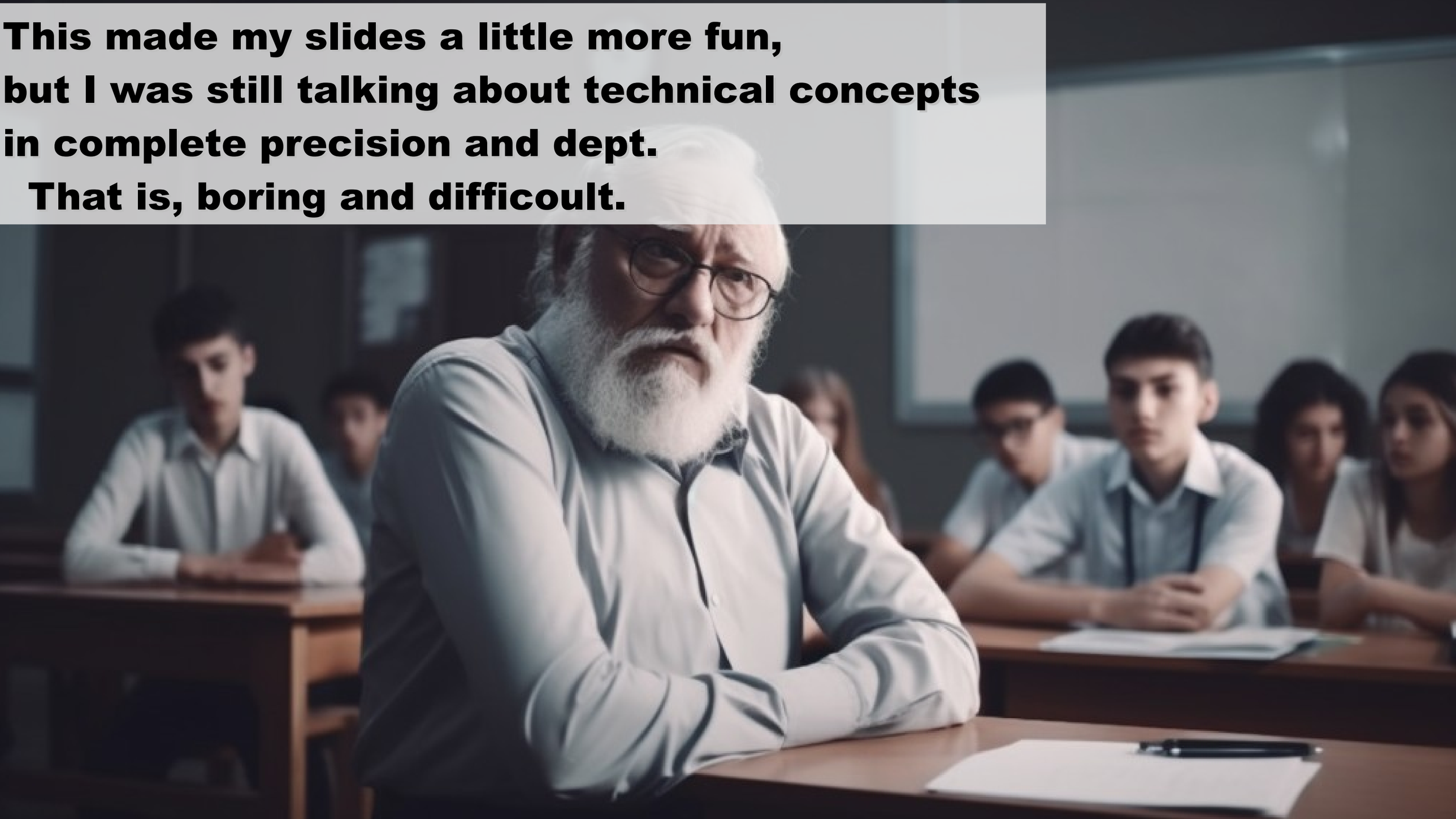
Hint!

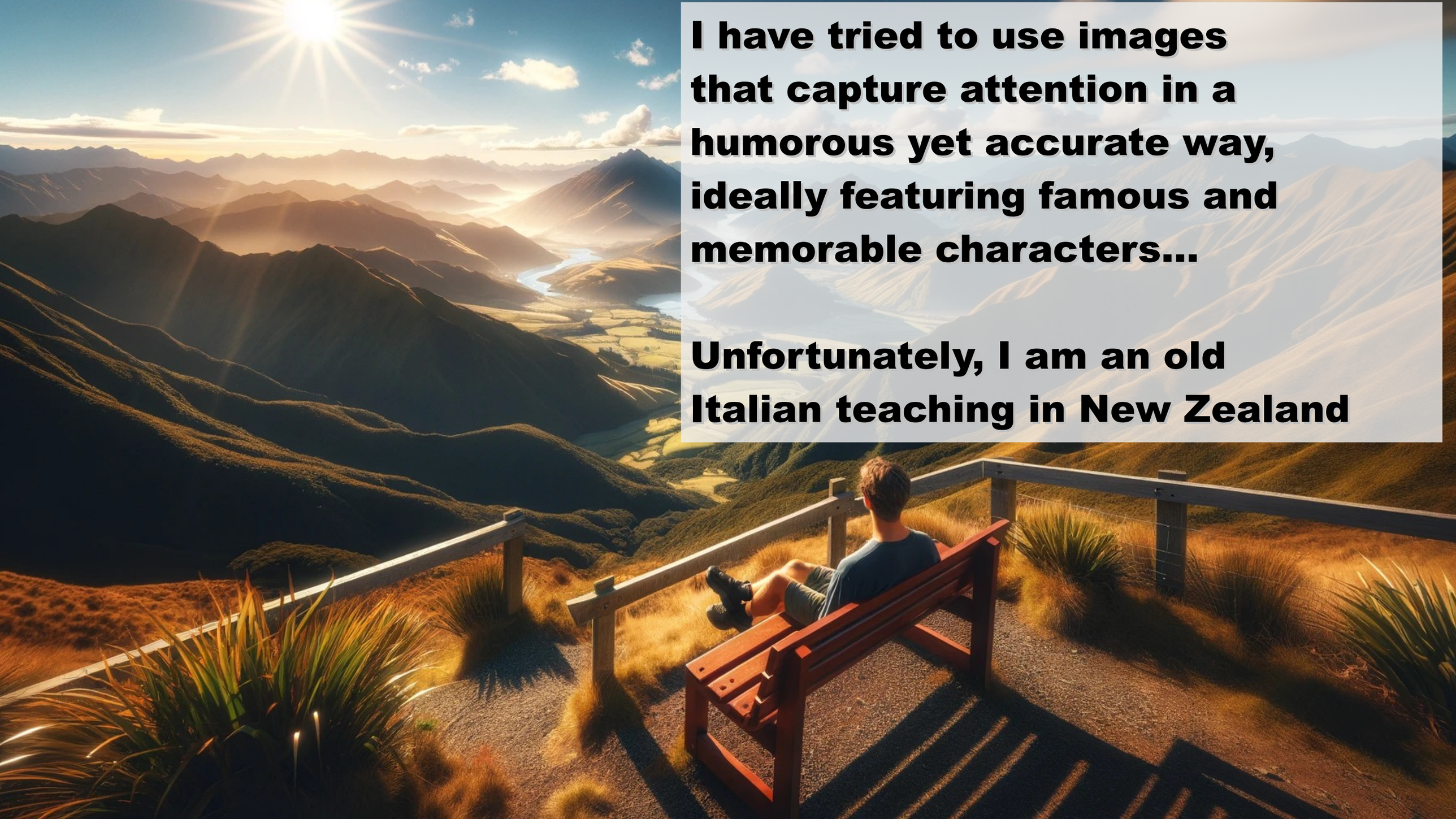
When searching for Java examples, Java documentation or Java help

- Never search for “Java”
  you will get the old stuff.

- Search for “Java 21”, “Java 22”
  or “Java 2024” to get good stuff!

I tried to introduce memorable images and to reuse them to recall a concept. But I was limited by the images I could find online. This has now changed..


This made my slides a little more fun,
but I was still talking about technical concepts
in complete precision and dept.
That is, boring and difficoult.



I have tried to use images that capture attention in a humorous yet accurate way, ideally featuring famous and memorable characters...

Unfortunately, I am an old Italian teaching in New Zealand

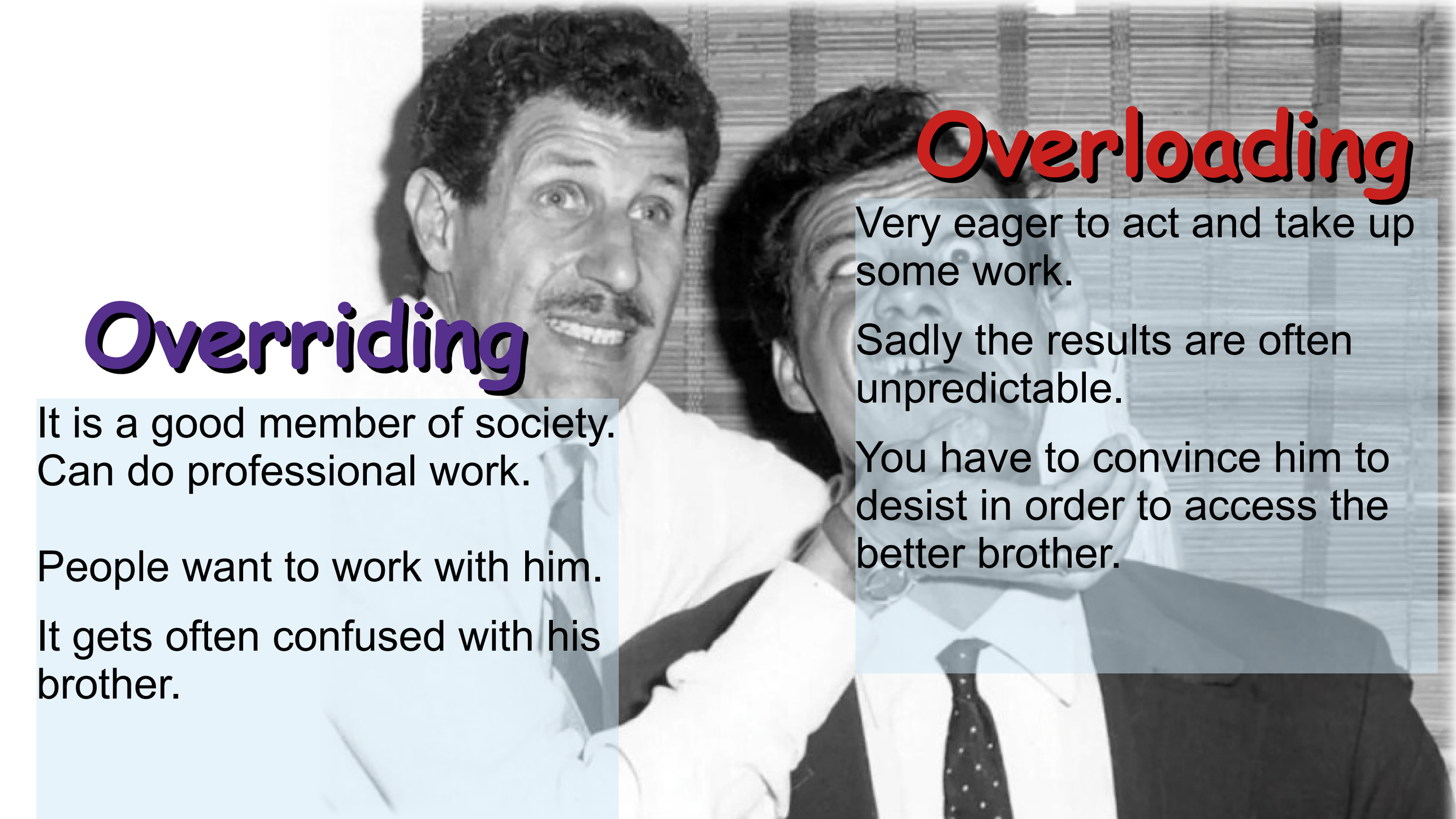

Overloading
Overriding
The two brothers

Overriding

It is a good member of society.
Can do professional work.

People want to work with him.

It gets often confused with his brother.



Overriding

It is a good member of society.
Can do professional work.

People want to work with him.

It gets often confused with his brother.

Overloading

Very eager to act and take up some work.

Sadly the results are often unpredictable.

You have to convince him to desist in order to access the better brother.

It is easy to accidentally assign work to the wrong brother

Overriding

It is a good member of society. Can do professional work.

People want to work with him.

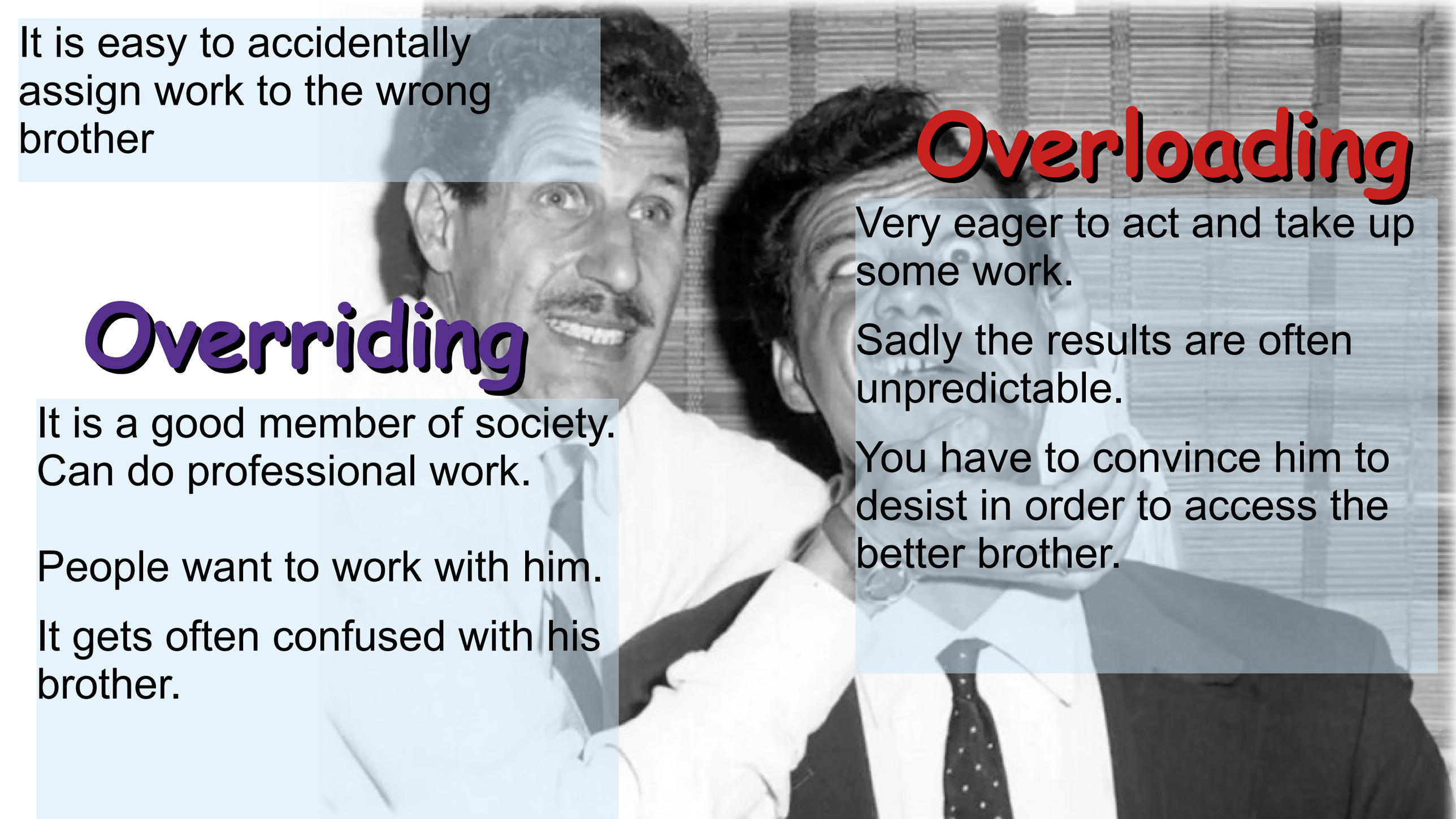
It gets often confused with his brother.

Overloading

Very eager to act and take up some work.

Sadly the results are often unpredictable.

You have to convince him to desist in order to access the better brother.



Sometimes my metaphors are of great success

Java exceptions and Norse mythology

Java exceptions and Norse mythology

- Odin.
- The father of all gods.
- Throwable
- The father of all kinds of exceptions.
- It is very rare to do new Throwable(..).
- Sometime used as a type.
- Quite rare to do catch(Throwable t){..}

Java exceptions and Norse mythology

- Thor.
- The most iconic.
- Exception
- The most iconic kind of Exception, the first that comes to mind.
- Rare to do
  new Exception(…)
- catch(Exception e){..}
  is often overkill

Java exceptions and Norse mythology

- Loki.
- The sneaky guy
- Error
- The one whose name causes the most dread.
- Common to do
  new Error(e)
- catch(Error e){..}
  is common in library code

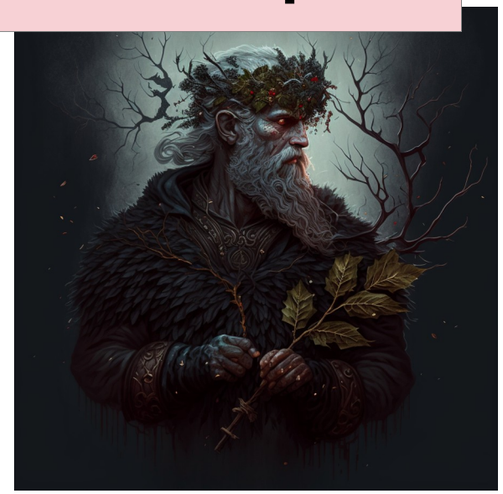
Java exceptions and Norse mythology

- Hoðr.
- The good sneaky guy with a tragic past: tricked by Loki to kill Baldr, the god of light.
- RuntimeException
- Helps us to solve limitations of Java by tunneling errors around the type system.
- Common to do
  new RuntimeException(e)
- catch(RuntimeException e){..} is common in library code

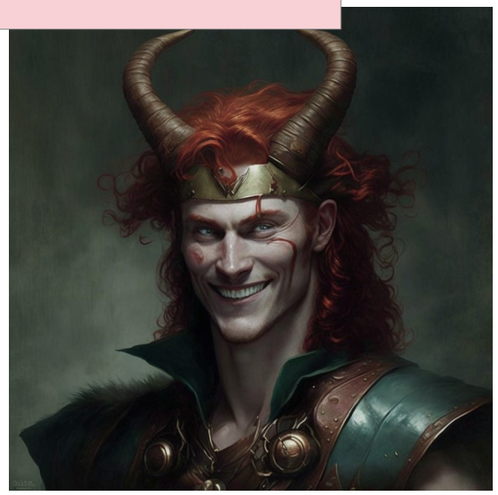
Throwable

Exception

RuntimeException



Error



Family relationships:
Throwable(Odin) is the father (supertype) of

- Error (Loki) and
- Exception (Thor). Exception is the father of
  - RuntimeException (Hoðr)

They all spawn their minions

Throwable

Checked

Exception

Checked

Checked

RuntimeException

Unchecked

Unchecked

Error

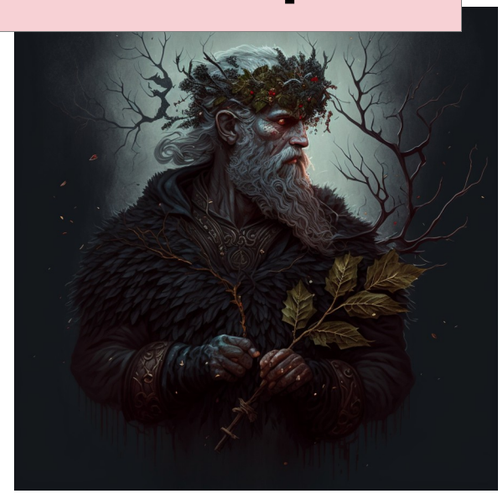
Unchecked

Unchecked

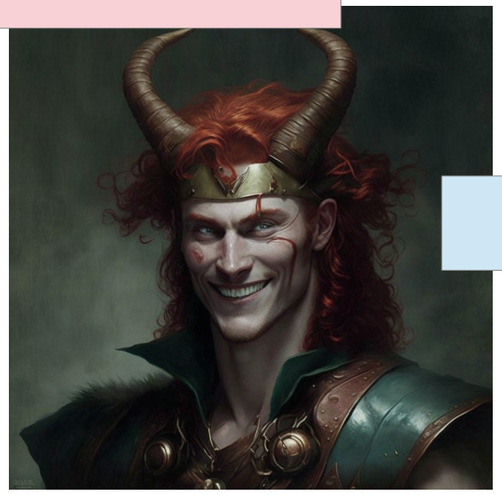
Throwable

Exception

RuntimeException



Error



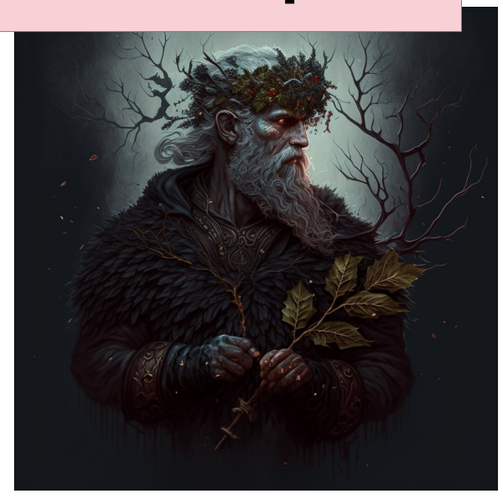
An Error is a subclass of Throwable
that indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.

AssertionError, AWTError, CoderMalfunctionError, IOError,
LinkageError, ThreadDeath, VirtualMachineError, ...

Throwable

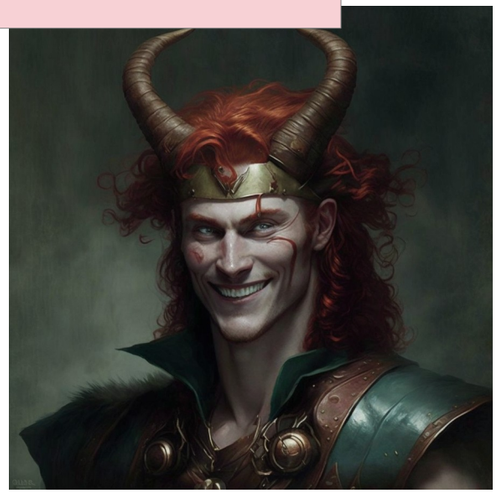
Exception

RuntimeException



The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.

Error



The class Exception and any subclasses that are not also subclasses of RuntimeException are checked exceptions. Checked exceptions need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the boundary.

Tons of kids:

ExecutionException, FontFormatException, GeneralSecurityException, InterruptedException, IOException, MarshalException, MidiUnavailableException, MimeTypeParseException, ParseException, ReflectiveOperationException, SQLException, TimeoutException, TooManyListenersException, XMLParseException, ...

Throwable

Exception

RuntimeException

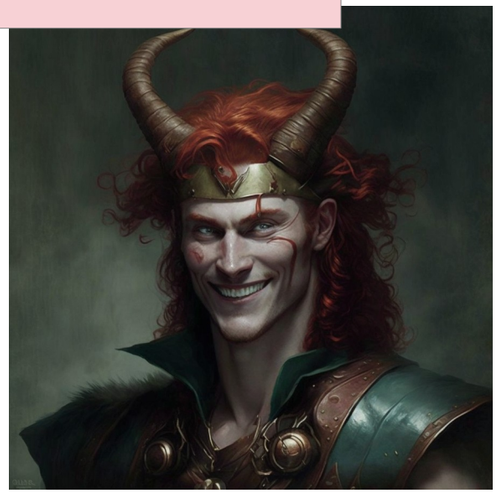
RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

RuntimeException and its subclasses are unchecked exceptions. Unchecked exceptions do not need to be declared in a method or constructor's throws clause

Tons of kids. Those are actually useful/relevant

ArithmeticException, ArrayStoreException,ClassCastException, CompletionException, ConcurrentModificationException, EnumConstantNotPresentException, IllegalArgumentException, IndexOutOfBoundsException, NegativeArraySizeException, NoSuchElementException, NullPointerException, SecurityException, UnmodifiableSetException, UnsupportedOperationException,

Error



More metaphors: catch something

- If the hero is too weak, the monster will win and the hero will be defeated.
- However, with the ‘finally’ block, the hero can say their last words!
- **No matter if they win or lose**, they will say those words before the stack unwinds away from them, sending them back into oblivion.


I started to even add
some music and performance

Automated testing -- Why is good

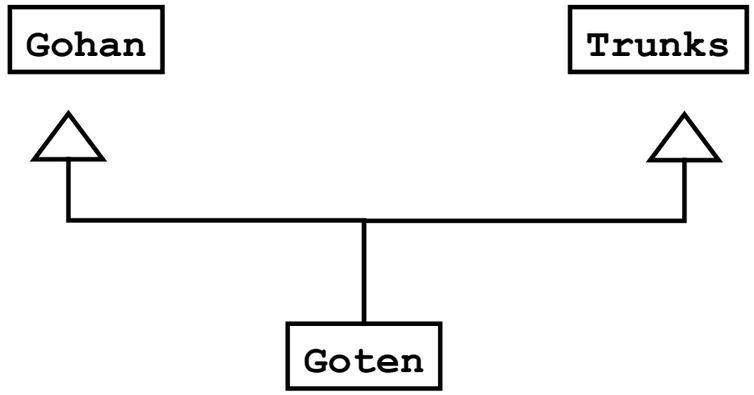
- Don Knuth: this is the story of my life

[...]
Birds don't just fly
They fall down and get up
Nobody learns without getting it won
[...]
I wanna try everything
I wanna try even though I could **fail**
I'll keep on making those **new** mistakes
I'll keep on making them **every day**
Those **new mistakes**
Oh oh, try everything

Shakira - Try Everything
https://www.youtube.com/watch?v=c6rP-YP4c5I

Multiple inheritance

https://www.youtube.com/watch?v=8cpyejx506o



- From Java8, this is now possible!
  - A class cannot have more than one superclass
    - Other languages (e.g. C++) support this
  - But, a class can implement **more than one interface**
  - Interfaces can now have reusable implementation!

```
class Goten extends Gohan, Trunks { … }
```

```
class Goten implements Gohan, Trunks { … }
```

- Risk of multiple inheritance: we may end up with too much stuff,
- A class that is too fat.
  (violating the single
  responsibility principle)

- To avoid this, we need to make the interfaces used to modularize our behaviour as slim as possible

- To avoid this, we need to make the interfaces used to modularize our behaviour as slim as possible

- We can have many different interfaces, sometimes with just a single default method (and the private methods that it uses internally)

- The other methods will be abstract and defined in other interfaces.

Too many images can be distracting,
Sometimes a single small image
can have a better impact

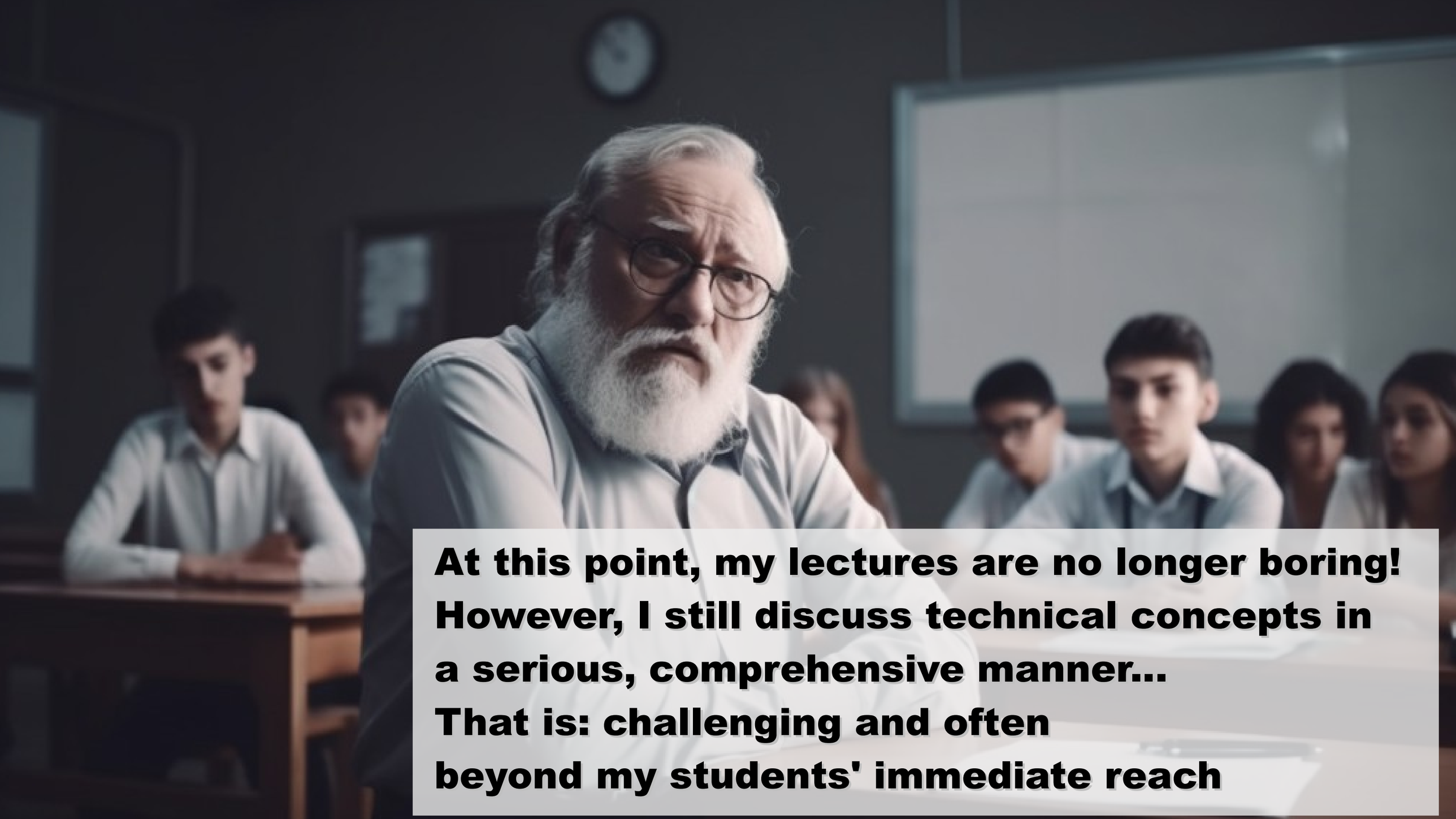
Property testing

Instead of just testing that an operation on a specific input does exactly what we expect, we can generate some input and check for general properties, for example with minIndex we could generate random sequences of numbers and test that the number at the resulting input is indeed smaller than all the other ones.

Watch: https://www.youtube.com/watch?v=IYzDFHx6QPY

“The lazy programmer's guide to writing thousands of tests”
- Scott Wlaschin

This talk is also known as “The enterprise developer from hell”


At this point, my lectures are no longer boring!
However, I still discuss technical concepts in
a serious, comprehensive manner...
That is: challenging and often
beyond my students' immediate reach

Zone of Proximal Development (ZPD) and Constructivism

- In VUW, we attend teaching seminars.
- Research result:
  The most effective learning is often facilitated by more experienced peers, those just 'beyond the limit'. Much of our learning occurs through social interaction, where students interact with peers who are engaged in the same or slightly more advanced stages of learning. These classmates can offer unique insights and explanations, particularly accessible to other students.
- Hearing these theories, and recognizing them as true based on my experience, I felt redundant as a teacher. If students learn better from other students, what is my role?

Idea! I pretend to be another student when I teach!

"*These classmates can offer unique insights and explanations, particularly accessible to other students*"

- Why are they more accessible? Because they are FALSE! These explanations contain enough truth to allow students to complete exercises and activities, but they are not always perfectly correct. They are simplified, and often full of misunderstandings that, in the long run, might prove counterproductive. However, in the short term, these 'imperfect' explanations work.
- If one is willing to refine and correct their knowledge as they go along, then learning the reality through successive approximations might not be a bad idea. Perhaps?


I just.. can not do it.
I can not lie.
I will not lie.
Even if I have to fail
all my students,
(and be fired for it)
I will not lie!

Ultimate Programming University

So, I stated writing a comicbook!

- When something is easy, I explain it directly.
- When it’s difficult, I use the comic book.
- The story evolves so that the concept becomes relevant.
- My characters discover the concept and don’t fully understand it. They help each other, discuss, do exercises, take tests, etc.

- My characters don’t lie either; they simply say things that are reasonable and ‘in character’ according to the level of competence they have at that point in the story.
- This way, I can use imperfect explanations without lying.