
Hey, boys,
I’m Sophia.


Hey, boys,
I'm Sophia.
You are from our
class, right?


Hey, boys,
I'm Sophia.
You are from our
class, right?
You are talking "generics".
Can we join you?


Oh, yea! The more the merrier


Oh, yea! The more the merrier
I'm Tommy,
nice to meet you!


pepsi


I'm Ricky
pepsi


I'm Ricky
We were discussing
about that Pepsi truck


I'm Ricky
We were discussing
about that Pepsi truck
And what options there are to encode
it properly with generics


In the lecture Pupon discussed inheritance


In the lecture Pupon discussed inheritance
... using trucks as an example, right?


There they are,
like in the lecture!


There they are, like in the lecture!
A cabover truck parked near a bonneted truck!


Here they are!


Here they are!
I’m Luna,
by the way

```
abstract class Truck<T>{



}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){



  }




}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }



}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private Cargo<List<T>> cargo;
  private Point location;
  Truck(Point p, Cargo<List<T>> c){
    location=p; cargo=c;
  }
}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private Cargo<List<T>> cargo;
  private Point location;
  Truck(Point p, Cargo<List<T>> c){
    location=p; cargo=c;
  }
}
```

```
class Bonneted<T> extends Truck<T>{



}
```

```
class CabOver<T> extends Truck<T>{



}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private Cargo<List<T>> cargo;
  private Point location;
  Truck(Point p, Cargo<List<T>> c){
    location=p; cargo=c;
  }
}
```

```
class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, Cargo<List<T>> c){
    super(p,c);
  }
  public int aerodynamics(){ ... }
}
```

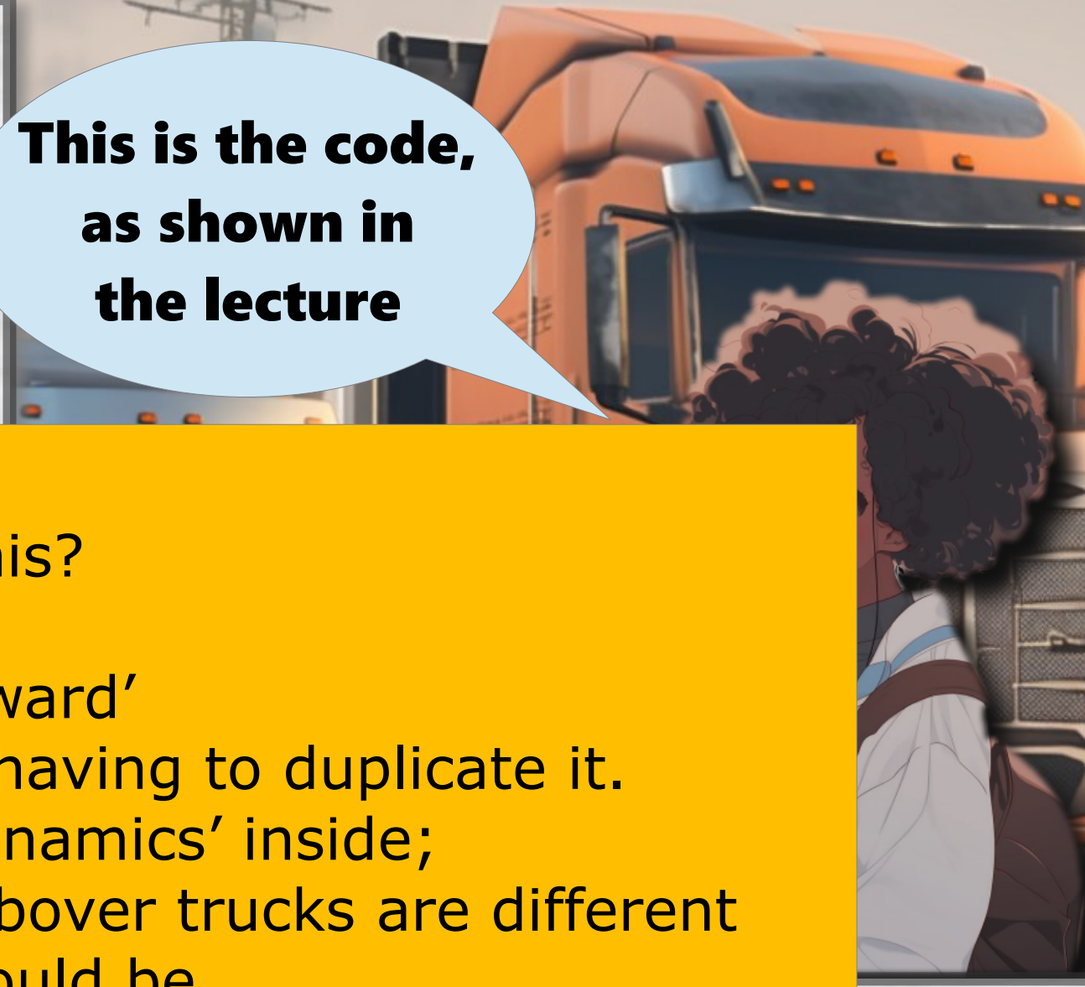
```
class CabOver<T> extends Truck<T>{
  CabOver(Point p, Cargo<List<T>> c){
    super(p,c);
  }
  public int aerodynamics(){ ... }
}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private Cargo<List<T>> cargo;
  private Point location;
  Truck(Point p, Cargo<List<T>> c){
    location=p; cargo=c;
  }
}
```


```
class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, Cargo<List<T>> c){
    super(p,c);
  }
  public int aerodynamics(){ ... }
}
```

```
class CabOver<T> extends Truck<T>{
  CabOver(Point p, Cargo<List<T>> c){
    super(p,c);
  }
  public int aerodynamics(){ ... }
}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Po
    lo
  }
  priv
  priv
  Truc
    lo
  }
}
```



Can you understand this code?
Do you think it is the best way to handle this?

The main objective is to reuse the ‘driveToward’ method in Bonneted and CabOver without having to duplicate it.
But, ‘driveToward’ may have to use ‘aerodynamics’ inside; and the aerodynamics of bonnetted and cabover trucks are different
‘Aerodynamics’ is just an example, there could be more complex differences

```
class
  Bonn
    super(p,c);
  }
  public int aerodynamics(){ ... }
}
```

```
    super(p,c);
  }
  public int aerodynamics(){ ... }
}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private Cargo<List<T>> cargo;
  private Point location;
  Truck(Point p, Cargo<List<T>> c){
    location=p; cargo=c;
  }
}
```


```
class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, Cargo<List<T>> c){
    super(p,c);
  }
  public int aerodynamics(){ ... }
}
```

```
class CabOver<T> extends Truck<T>{
  CabOver(Point p, Cargo<List<T>> c){
    super(p,c);
  }
  public int aerodynamics(){ ... }
}
```


Wow, that is exactly
Pupon code


Wow, that is exactly Pupon code
You must be very fast at taking notes!


Er... no,
I can't write
that fast.


Er... no,
I can't write
that fast.
You see, Luna has
photographic memory.


Er... no,
I can't write
that fast.
You see, Luna has
photographic memory.
She wrote it down
-


Er... no,
I can't write
that fast.
You see, Luna has
photographic memory.
She wrote it down
right now
just for our sake.


You are Dany right?


You are Dany right?
The guy that squashed
Pupon the first day.


You are Dany right?
The guy that squashed
Pupon the first day.
Surely you must have
some tricks up
your sleeves too.


You are Dany right?
The guy that squashed
Pupon the first day.
Surely you must have
some tricks up
your sleeves too.
I'm Emma.


You are Dany right?
The guy that squashed
Pupon the first day.
Surely you must have
some tricks up
your sleeves too.
I'm Emma.
If you are planning to
get in trouble again,
do NOT involve me


Talking about trouble...


Talking about trouble...
Miss Pumpkin also had a similar example


Talking about trouble...
Miss Pumpkin also had a similar example
But the code looks completely different!


But.. how?


But.. how?
How can you access Pumpkin’s notes?!


But.. how?
How can you access Pumpkin's notes?!
We are Pupon students,


But.. how?
How can you access Pumpkin’s notes?!
We are Pupon students, so UPU rules prevents us to follow the lectures


But.. how?
How can you access Pumpkin's notes?!
We are Pupon students, so UPU rules prevents us to follow the lectures of Pumpkin


But.. how?
How can you access Pumpkin's notes?!
We are Pupon students, so UPU rules prevents us to follow the lectures of Pumpkin
Oh, I just snatched them!


You did what?


You did what?
This is very serious!


You did what?
This is very serious!


You did what?
This is very serious!
Again? After all the mess when we stole Hanton notes?


Wait...
Wow...


Wait...
Wow...
This is not even
the first time?


Wait...
Wow...
This is not even the first time?


Wait...
Wow...
This is not even the first time?
So, you want to see them or not?


Hey Tommy,
how did you snatch
them this time?


Hey Tommy,
how did you snatch
them this time?
I have a man
inside the other class


Wow, you sound like a mafia boss


Wow, you sound like a mafia boss
But you are just saying that your friend in the other class


Wow, you sound like a mafia boss
But you are just saying that your friend in the other class shared their notes with you


Those notes are going to be precious!
The final exam is coming soon


Those notes are going to be precious!
The final exam is coming soon
It will be written half and half,
by both Pupon and Pumpkin


Come on,
just show us the
notes already!


Here they are!
Get ready for
the surprise!

```
interface Truck<Self,T>{



}
```

```
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){



  }
}
```

```
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```

```
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}



record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{


}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{


}
```

```
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```


```
record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }

}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }

}
```

```
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```


```
record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

```
abstract class Truck<T>{

}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){



  }



}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }


}
```

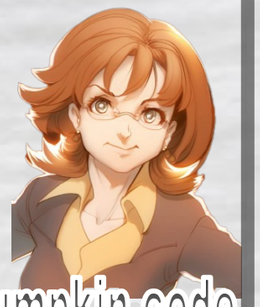
```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private Cargo<List<T>> cargo;
  private Point location;
  Truck(Point p, Cargo<List<T>> c){ location=p; cargo=c; }
}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private Cargo<List<T>> cargo;
  private Point location;
  Truck(Point p, Cargo<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{


}

class CabOver<T> extends Truck<T>{


}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private Cargo<List<T>> cargo;
  private Point location;
  Truck(Point p, Cargo<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, Cargo<List<T>> c){ super(p,c); }

}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, Cargo<List<T>> c){ super(p,c); }

}
```

```
abstract class Truck<T>{
  public abstract int aerodynamics();
  public Cargo<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private Cargo<List<T>> cargo;
  private Point location;
  Truck(Point p, Cargo<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, Cargo<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, Cargo<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```

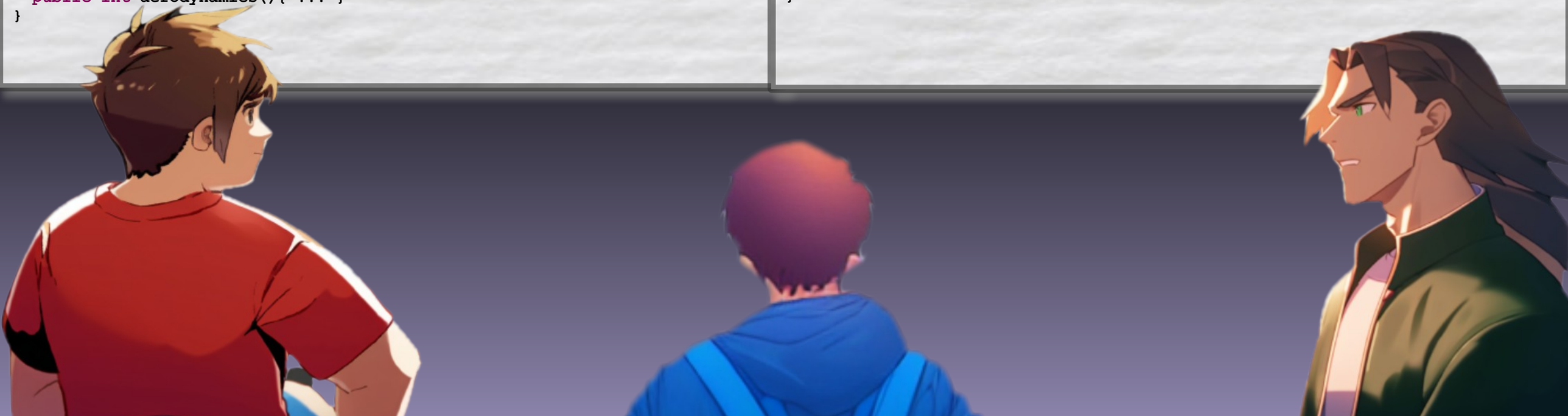


Pumpkin code

```java
record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code



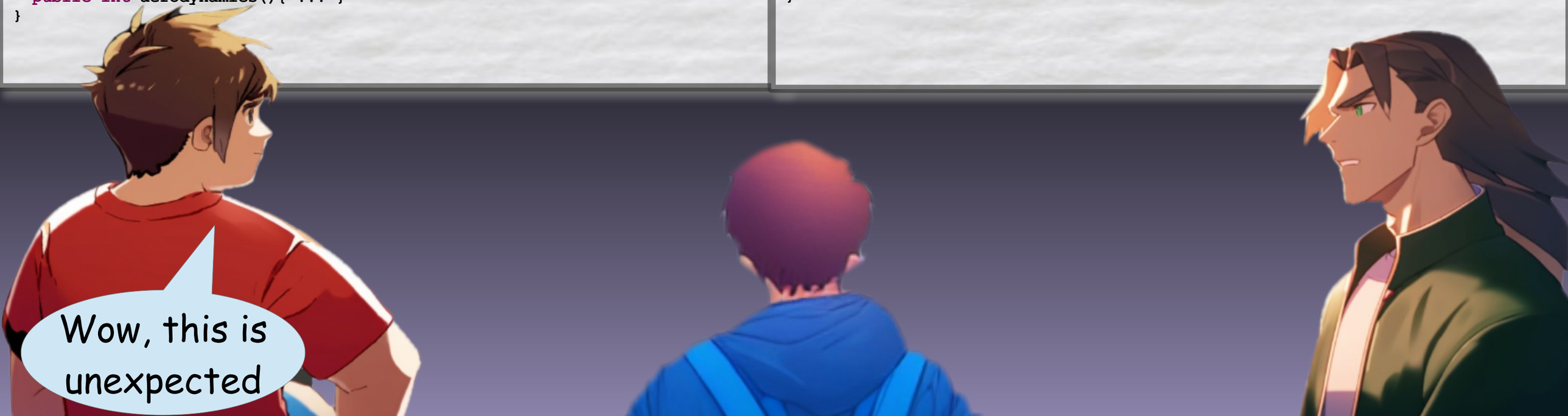
```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```

Pumpkin code

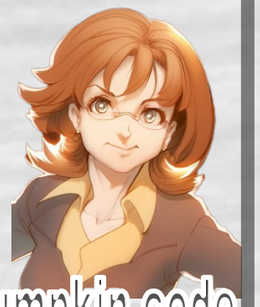
```java
record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code



```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```

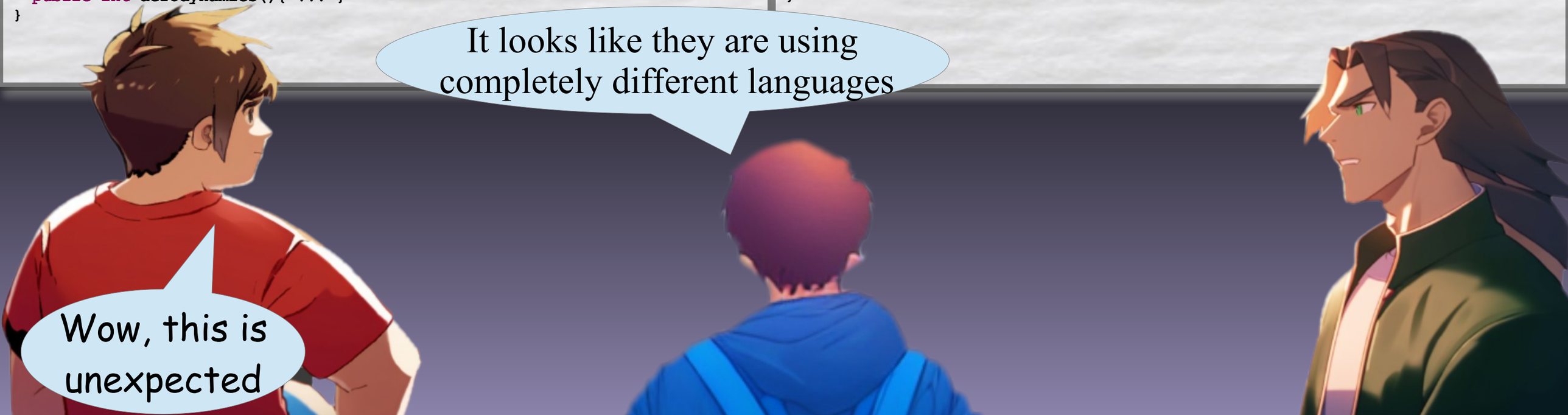


Pumpkin code

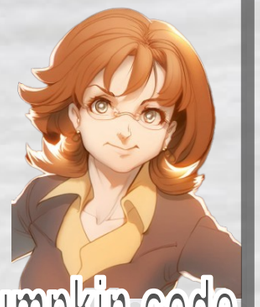
```java
record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code



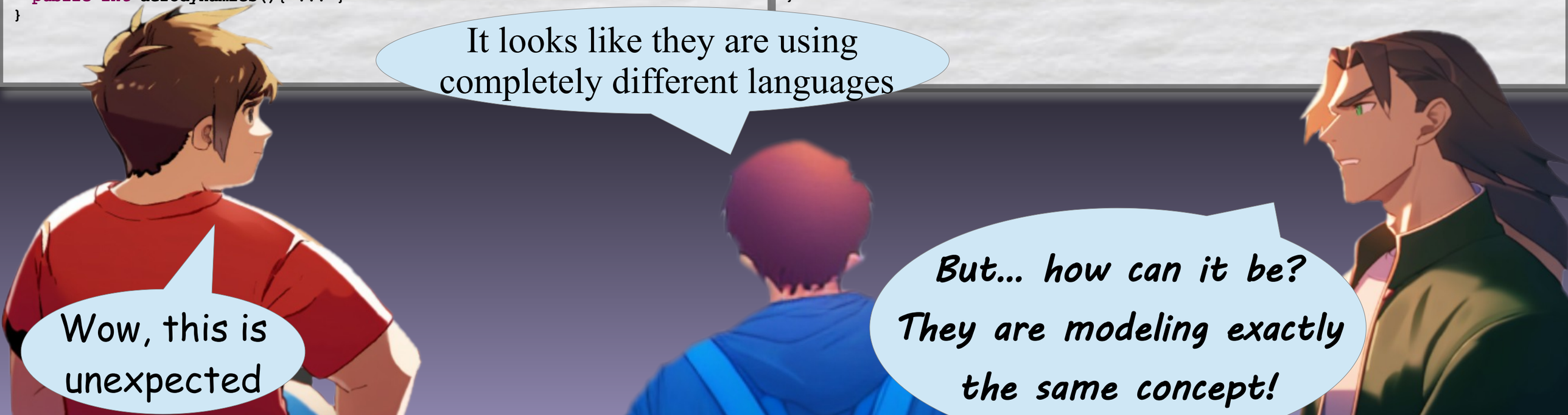
```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}



record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```



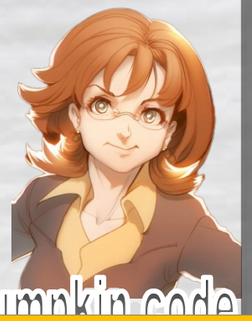
Pumpkin code

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code



```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```



Pumpkin code

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
```

Puppon code

```java
record Bonne
  (Point loc
  public Bon
  public int
}
record CabOv
 (Point loca
  public Cab
  public int
}
```

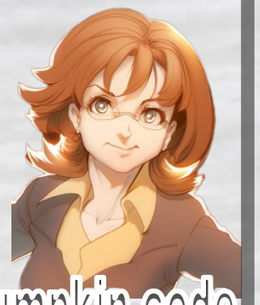
Can you understand the difference in these two approaches?
Do you think one is better then the other?

One of these two is not as usable as it should.
Which one? Why?

Wow, this is unexpected

But... how can it be? They are modeling exactly the same concept!

```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```



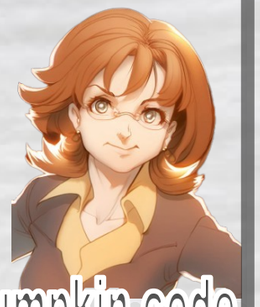
Pumpkin code

```java
record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code

```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```



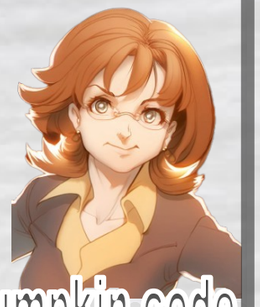
Pumpkin code

```java
record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code

```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```



Pumpkin code

```java
record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code

We are lost for words too.

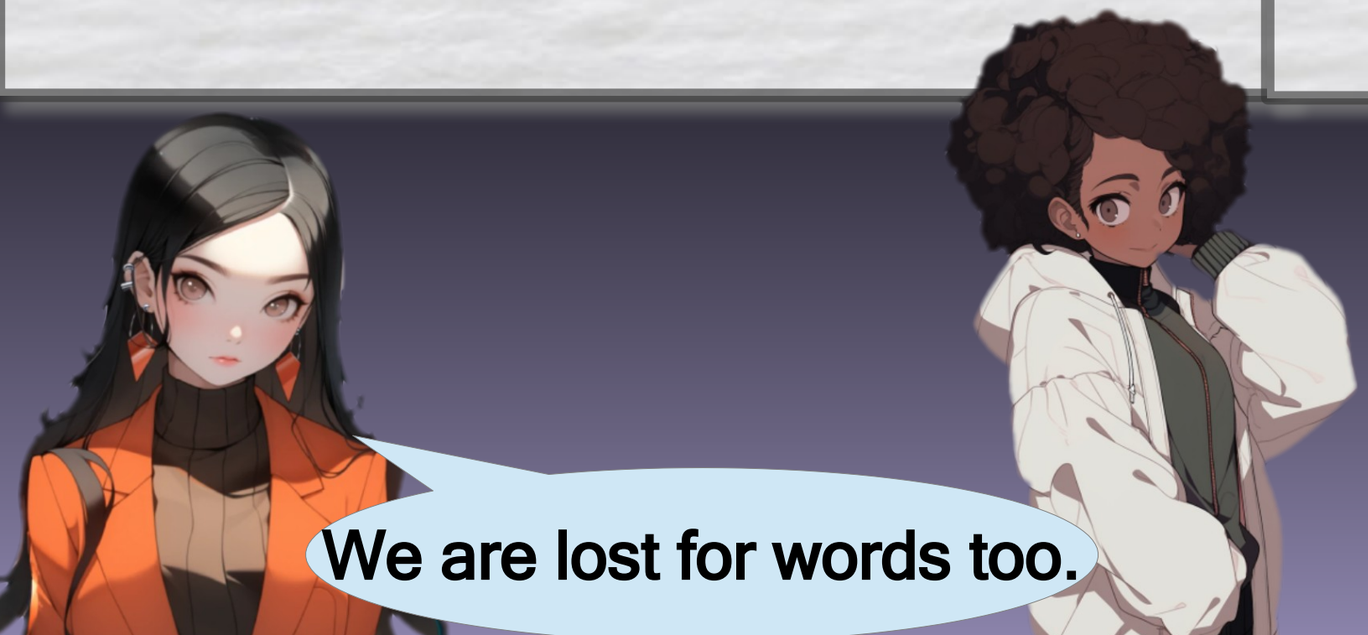
```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}
```

Pumpkin code

```java
record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code



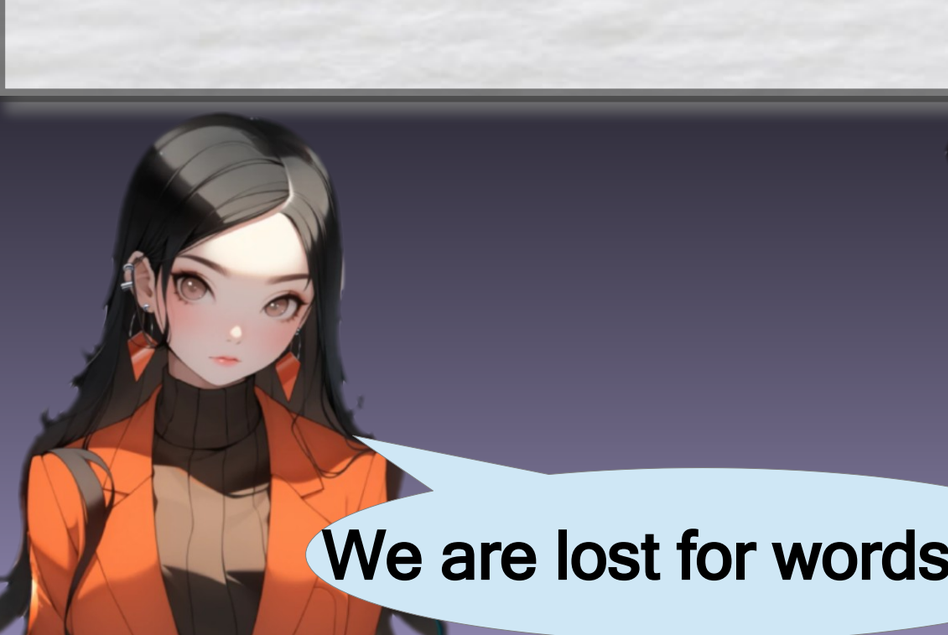
We are lost for words too.

```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}



record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

Pumpkin code

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code



We are lost for words too.

I've now memorized both versions but I can't understand why they are so different!

```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}



record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

Pumpkin code

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code

We are lost for words too.

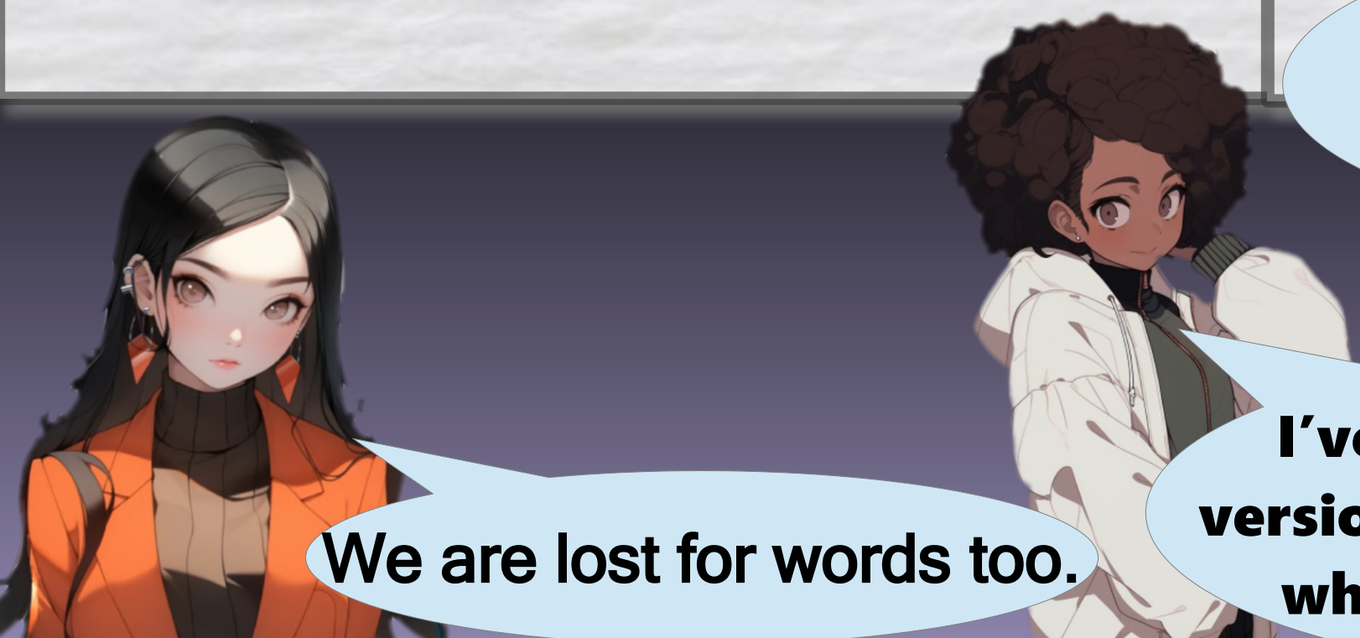
I've now memorized both versions but I can't understand why they are so different!

```java
interface Truck<Self,T>{
  int aerodynamics();
  CargoC<List<T>> cargo();
  Point location();
  Self withLocation(Point p);

  default Self driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    return withLocation(wayPoint);
  }
}



record Bonneted<T>
  (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
  public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
  public int aerodynamics(){ ... }
}
record CabOver<T>
 (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
  public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
  public int aerodynamics(){ ... }
}
```

Pumpkin code

```java
abstract class Truck<T>{
  public abstract int aerodynamics();
  public CargoC<List<T>> cargo(){ return cargo; }
  public Point location(){ return location; }
  public void location(Point p){ location=p; }

  void driveToward(Point destination, int duration){
    ...
    Point wayPoint=...;
    location(wayPoint);
  }
  private CargoC<List<T>> cargo;
  private Point location;
  Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
  Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
  CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
  public int aerodynamics(){ ... }
}
```

Pupon code



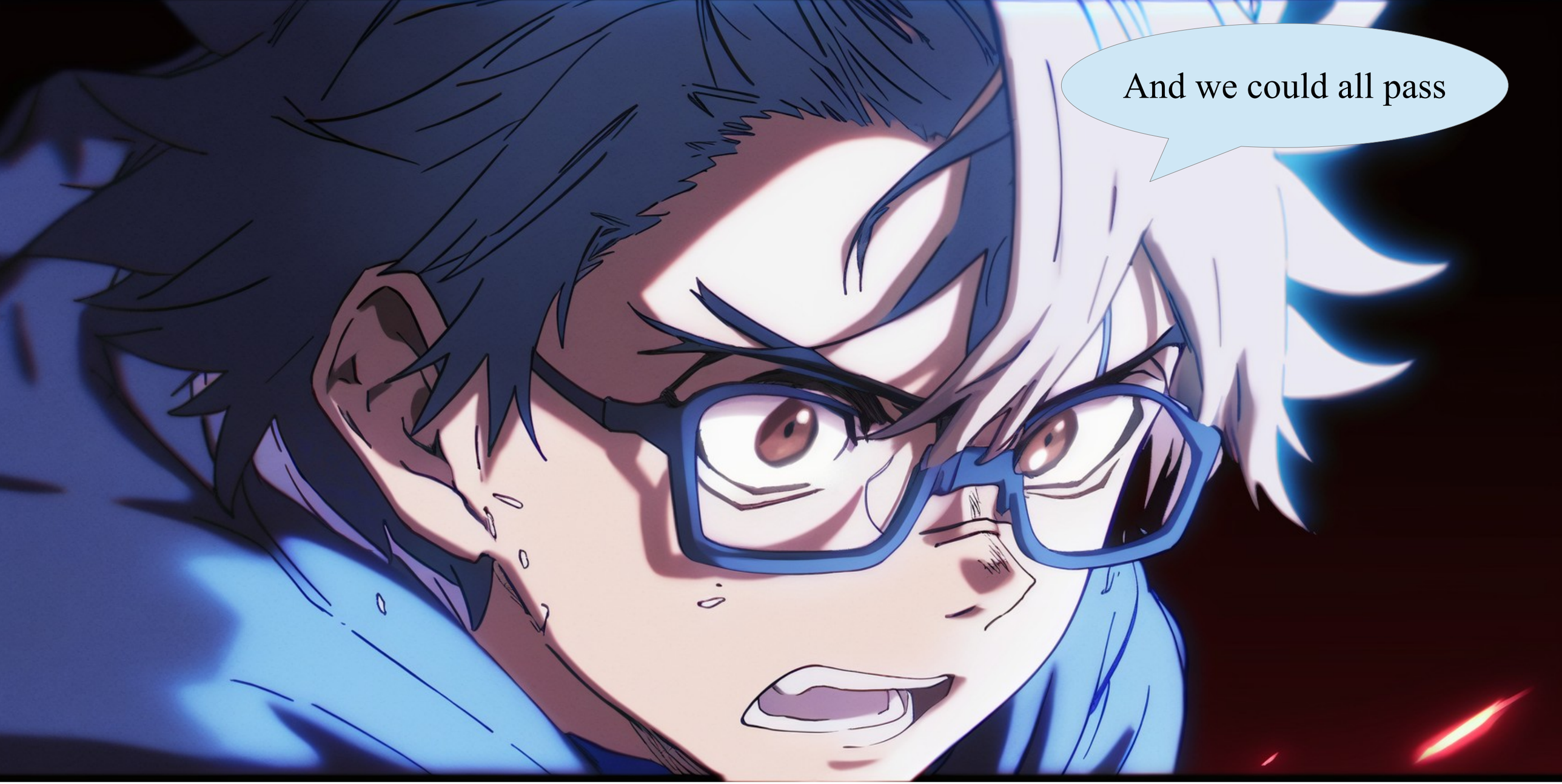
We are lost for words too.

If the six of us are to pass, we need to learn this stuff now!

I've now memorized both versions but I can't understand why they are so different!


This is it.


If I use my power,
I can understand that code
and teach them


And we could all pass


But if all of us pass


Pupon would likely win the challenge


And if he wins,
he will just …


Fail me!


However, I cannot just …


make other people fail
so that I can pass


I will do it!


Power of
Angelic explanations


Activate


Activate


It is do or die time!