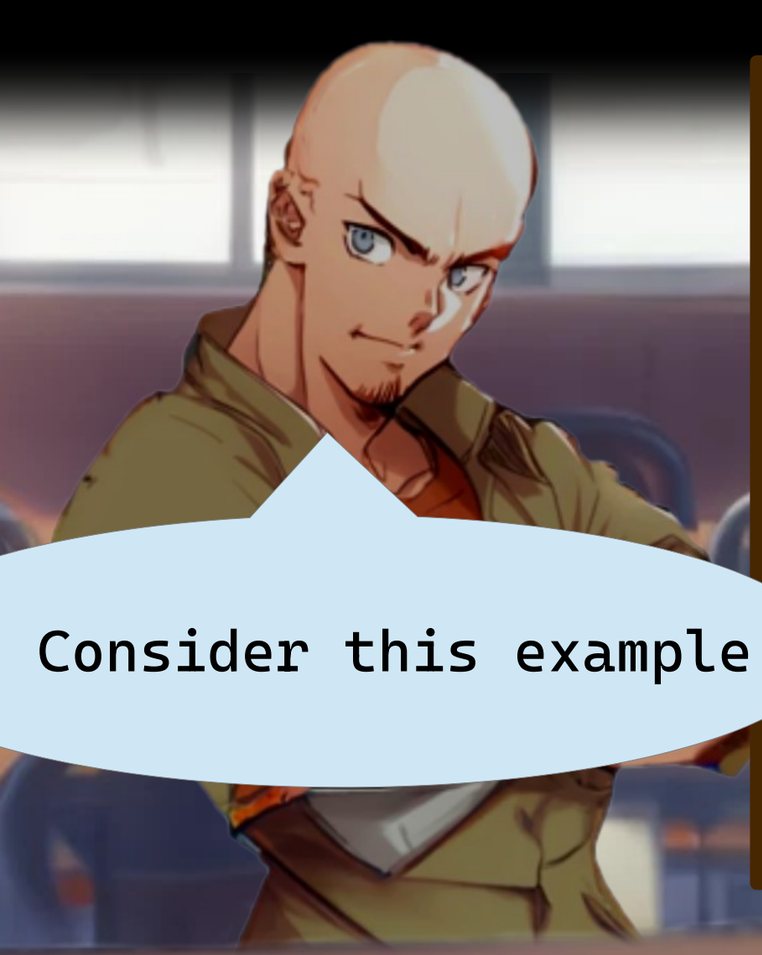
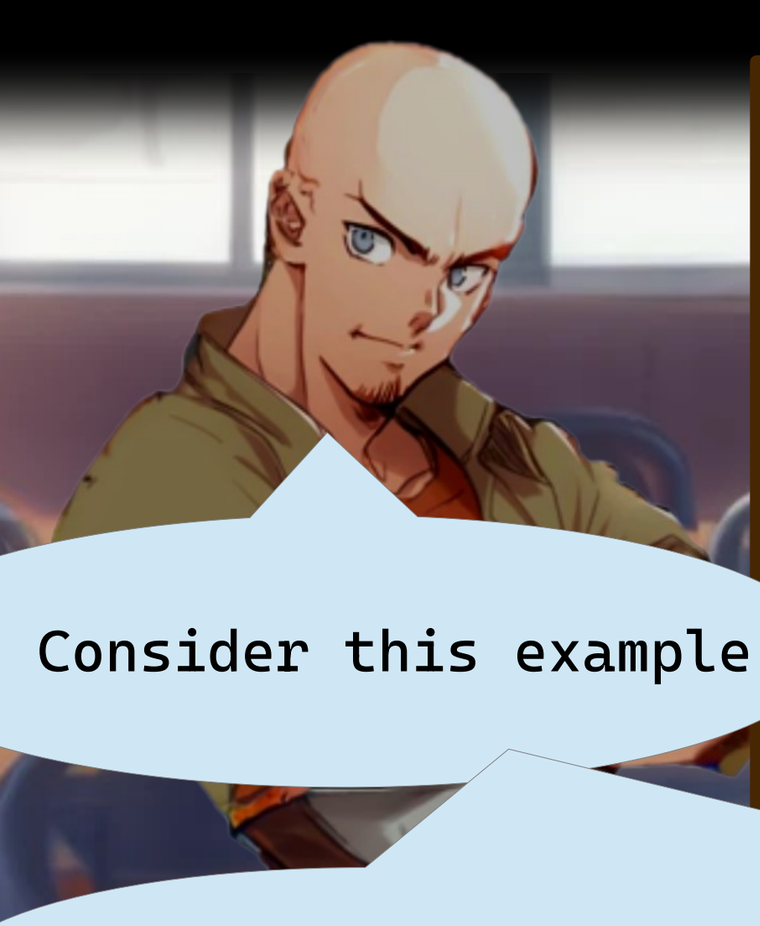
```java
//Given a String count how many digits and
//how many spaces in a single pass
final class StringInfo{
  public int digits = 0;
  public int spaces = 0;
  public StringInfo(String s){
     s.chars().forEach(c->{
     if(Character.isDigit(c)){ digits+=1; }
     if(Character.isWhitespace(c)){ spaces+=1; }
     });
   }
}
```



Consider this example

```
//Given a String count how many digits and
//how many spaces in a single pass
final class StringInfo{
  public int digits = 0;
  public int spaces = 0;
  public StringInfo(String s){
     s.chars().forEach(c->{
     if(Character.isDigit(c)){ digits+=1; }
     if(Character.isWhitespace(c)){ spaces+=1; }
     });
   }
}
```



Consider this example

```
//Given a String count how many digits and
//how many spaces in a single pass
final class StringInfo{
  public int digits = 0;
  public int spaces = 0;
  public StringInfo(String s){
    s.chars().forEach(c->{
    if(Character.isDigit(c)){ digits+=1; }
    if(Character.isWhitespace(c)){ spaces+=1; }
    });
  }
```

Here all fields have a default value.
If they were local variables,
  we would not be able to update them inside the lambda.
Moreover, in this way the class StringInfo also works
  as a tuple type, storing the two results we want.
If the logic to update those fields was more complex,
  we could easily split it into multiple methods of StringInfo

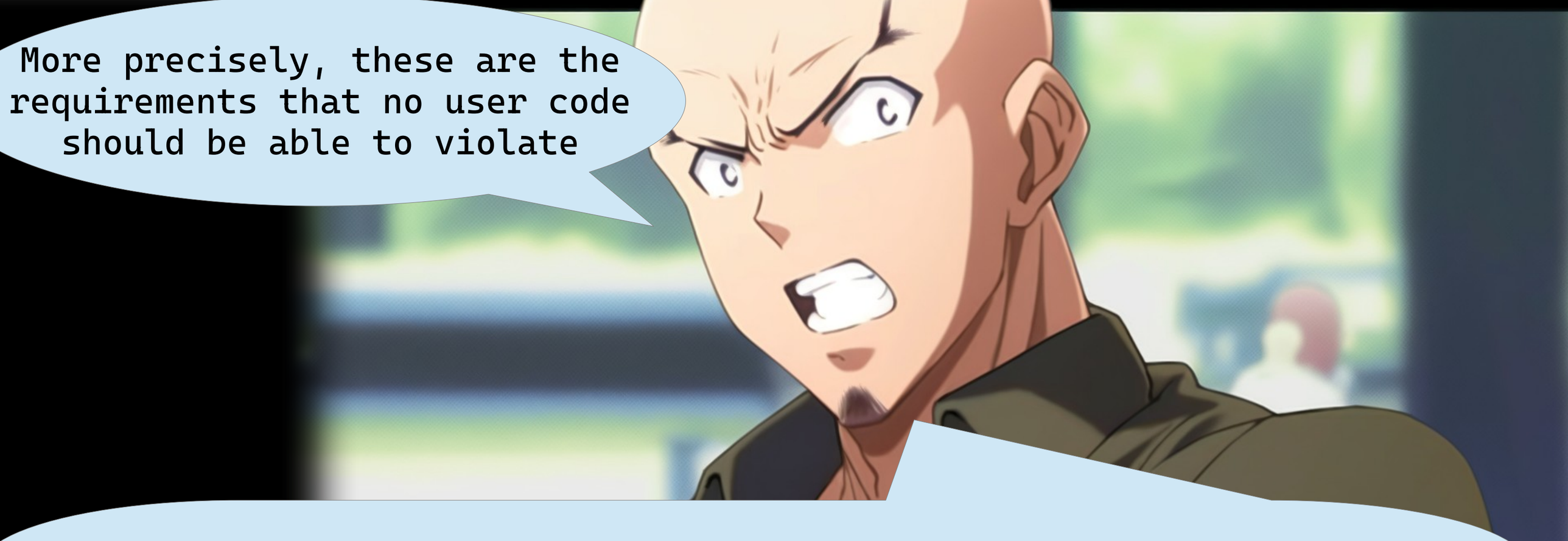
A challenging problem

A challenging problem

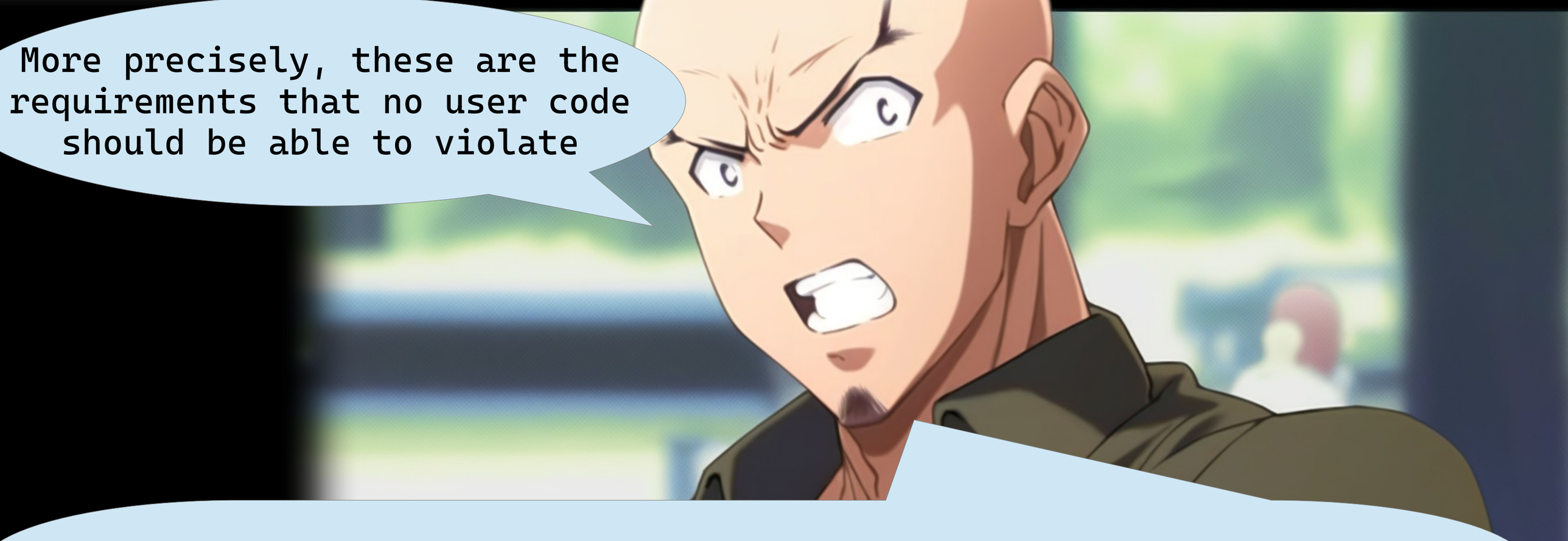
A challenging problem

A challenging problem

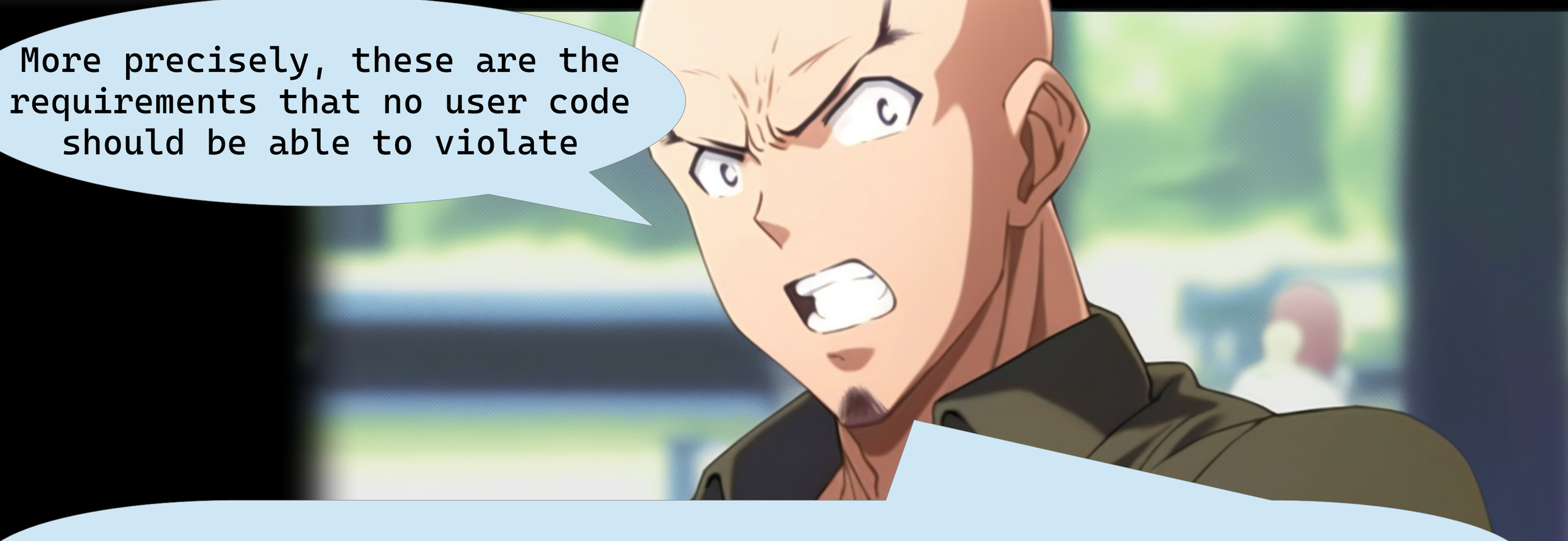

More precisely, these are the requirements that no user code should be able to violate



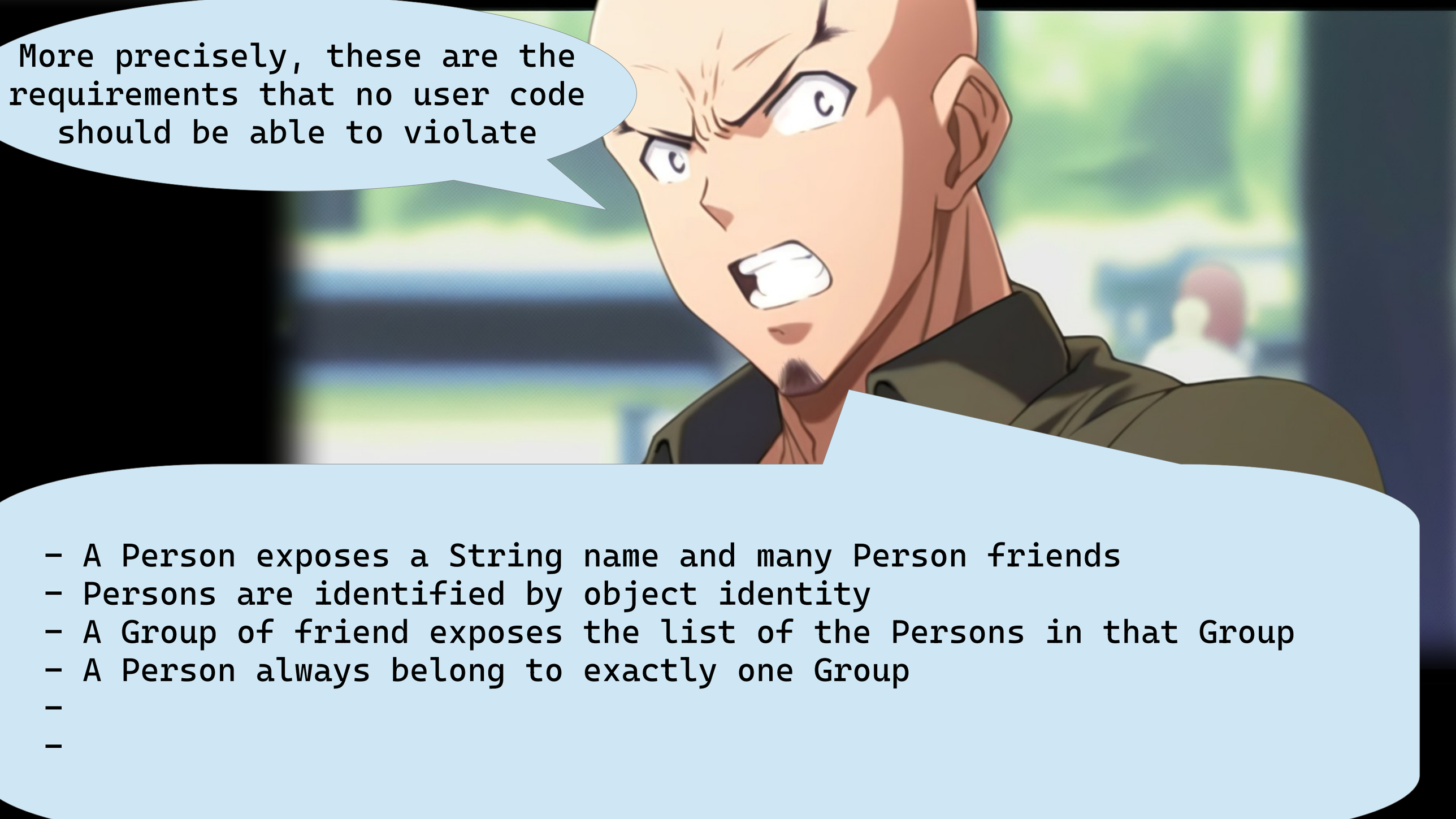
- A Person exposes a String name and many Person friends
-
-
-
-
-



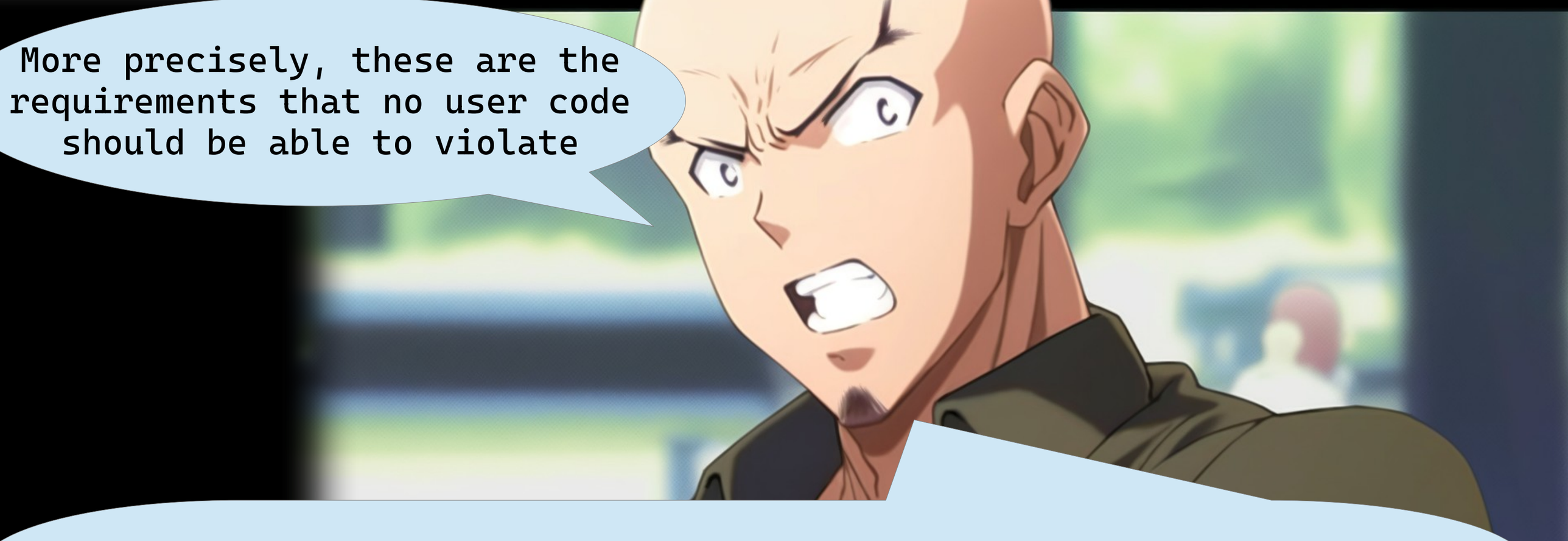
- A Person exposes a String name and many Person friends
- Persons are identified by object identity
-
-
-
-



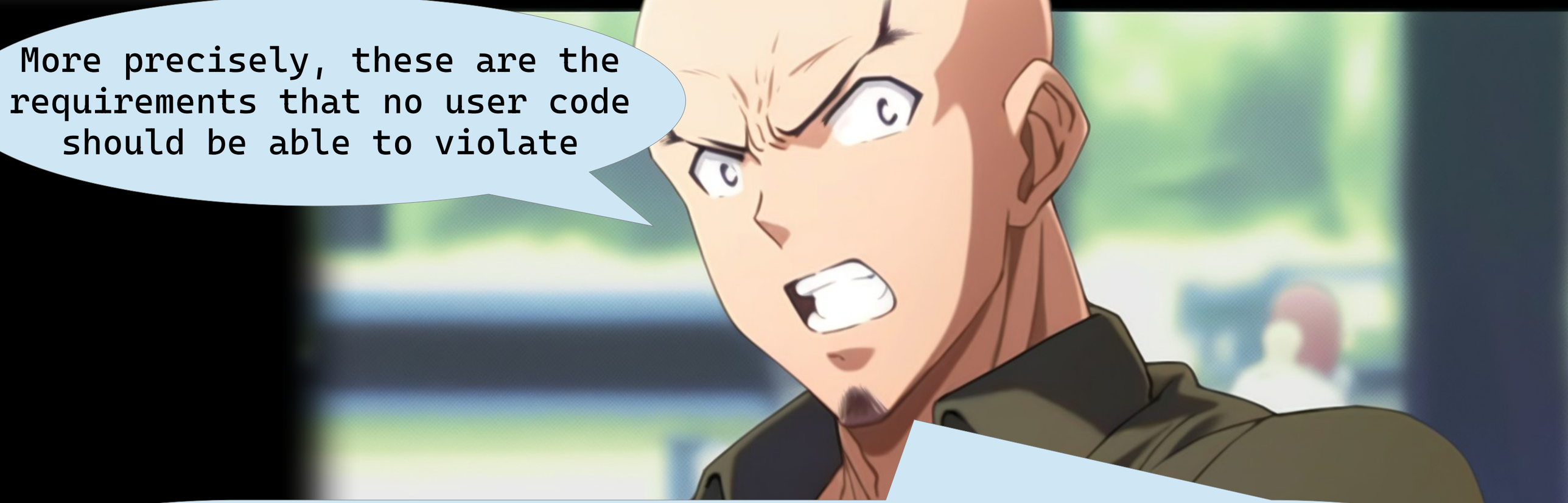
- A Person exposes a String name and many Person friends
- Persons are identified by object identity
- A Group of friend exposes the list of the Persons in that Group
-
-
-



- A Person exposes a String name and many Person friends
- Persons are identified by object identity
- A Group of friend exposes the list of the Persons in that Group
- A Person always belong to exactly one Group
-
-

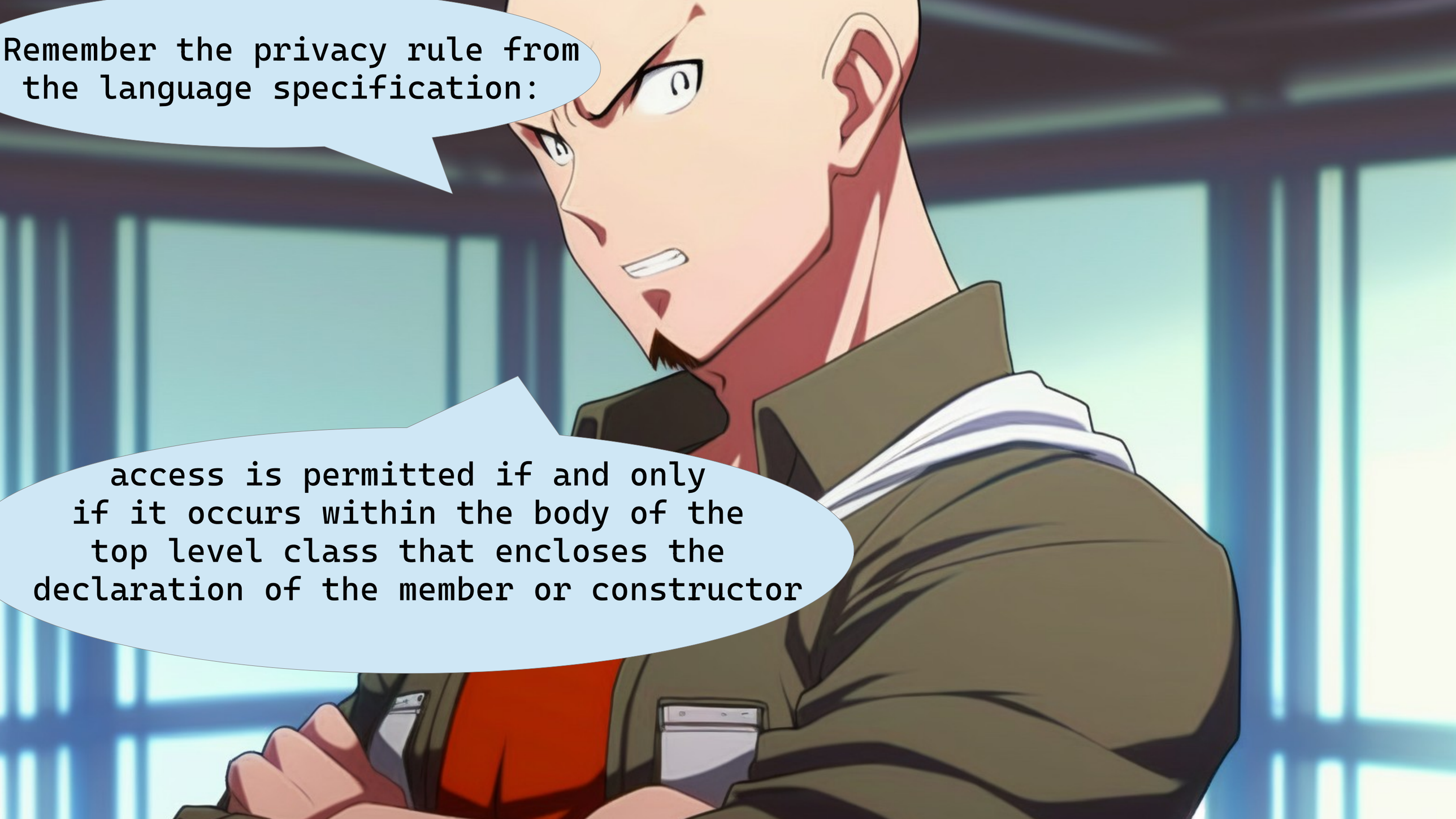


- A Person exposes a String name and many Person friends
- Persons are identified by object identity
- A Group of friend exposes the list of the Persons in that Group
- A Person always belong to exactly one Group
- All the friends of a Person belong to the same Group
-

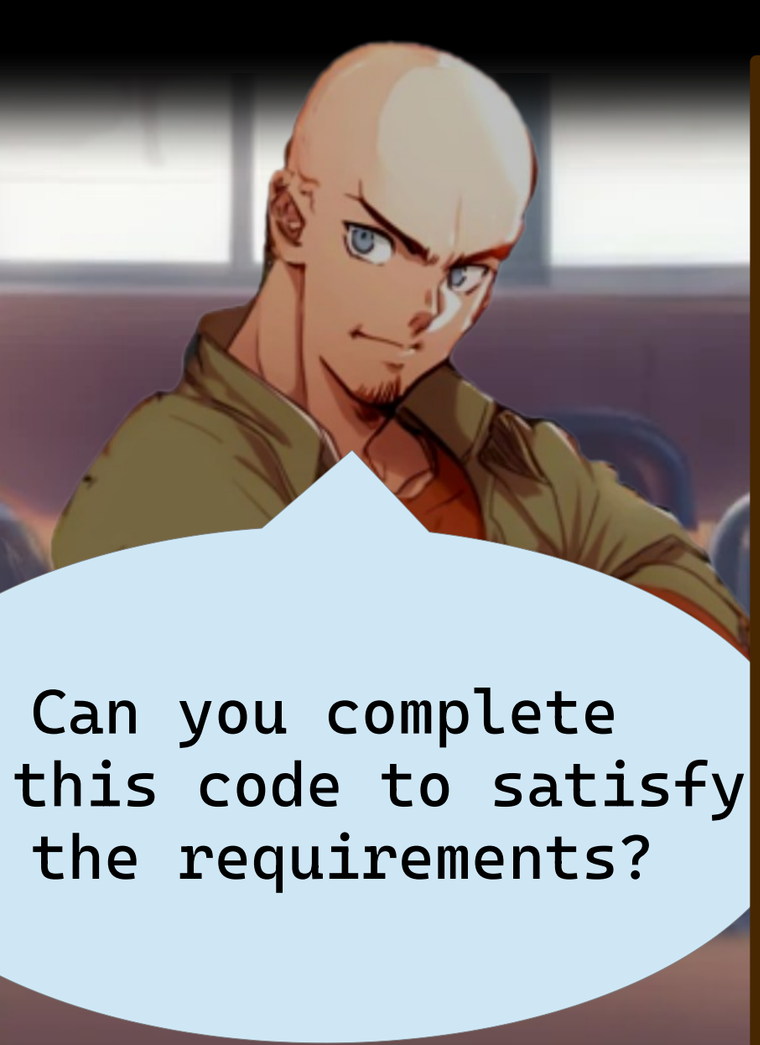


- A Person exposes a String name and many Person friends
- Persons are identified by object identity
- A Group of friend exposes the list of the Persons in that Group
- A Person always belong to exactly one Group
- All the friends of a Person belong to the same Group
- Friendship is symmetrical


Remember the privacy rule from the language specification:

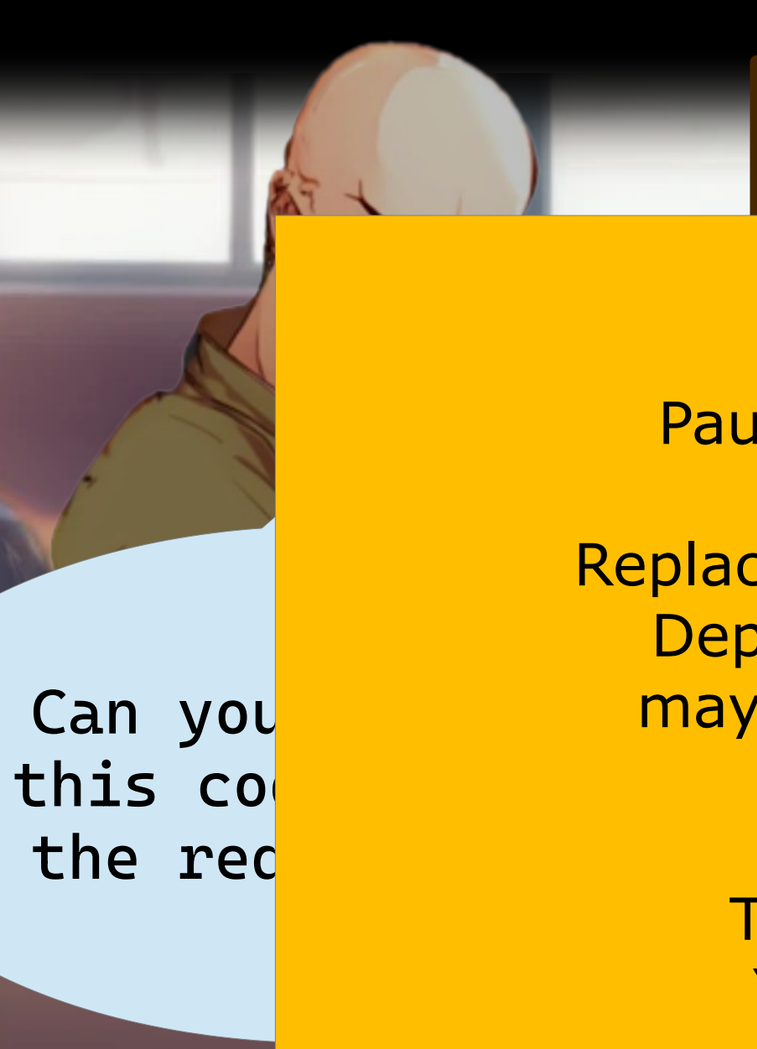

Remember the privacy rule from the language specification:
access is permitted if and only if it occurs within the body of the top level class that encloses the declaration of the member or constructor

```
public class Group{//Number of required lines may vary
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    /*here*/
  }
  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```



Can you complete this code to satisfy the requirements?

```
public class Group{//Number of required lines may vary
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    /*here*/
  }
  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```

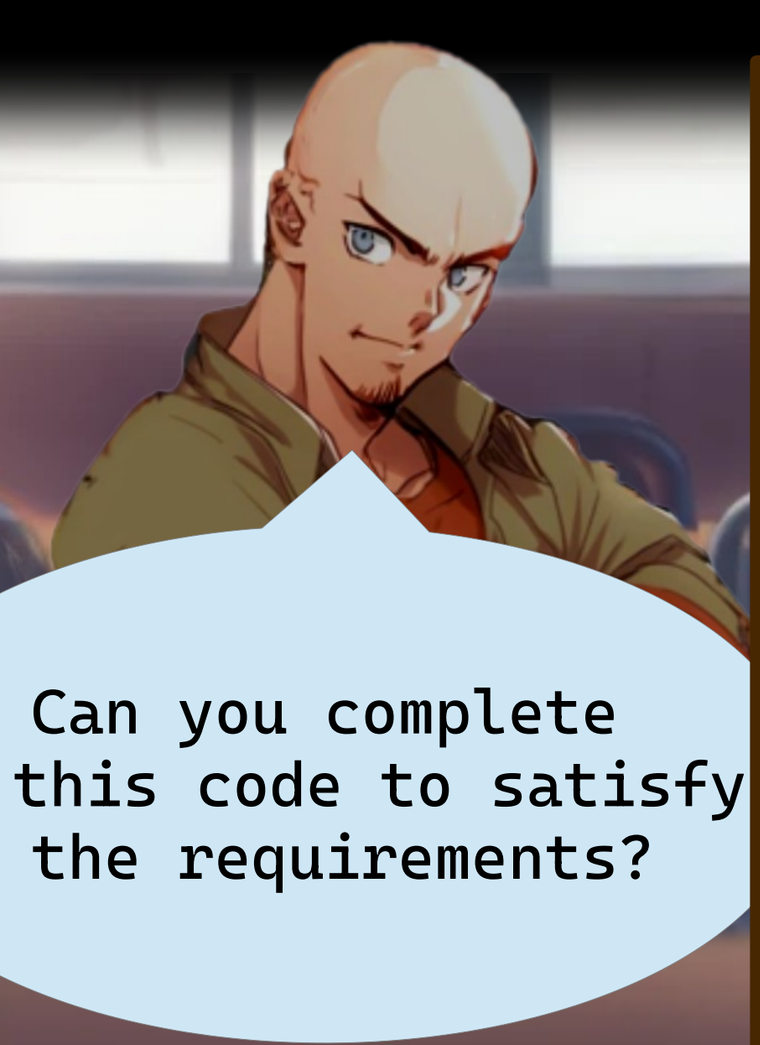



```
public class Group{//Number of required lines may vary
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
      }
   }
}
```

Pause the video. Can you solve this one?

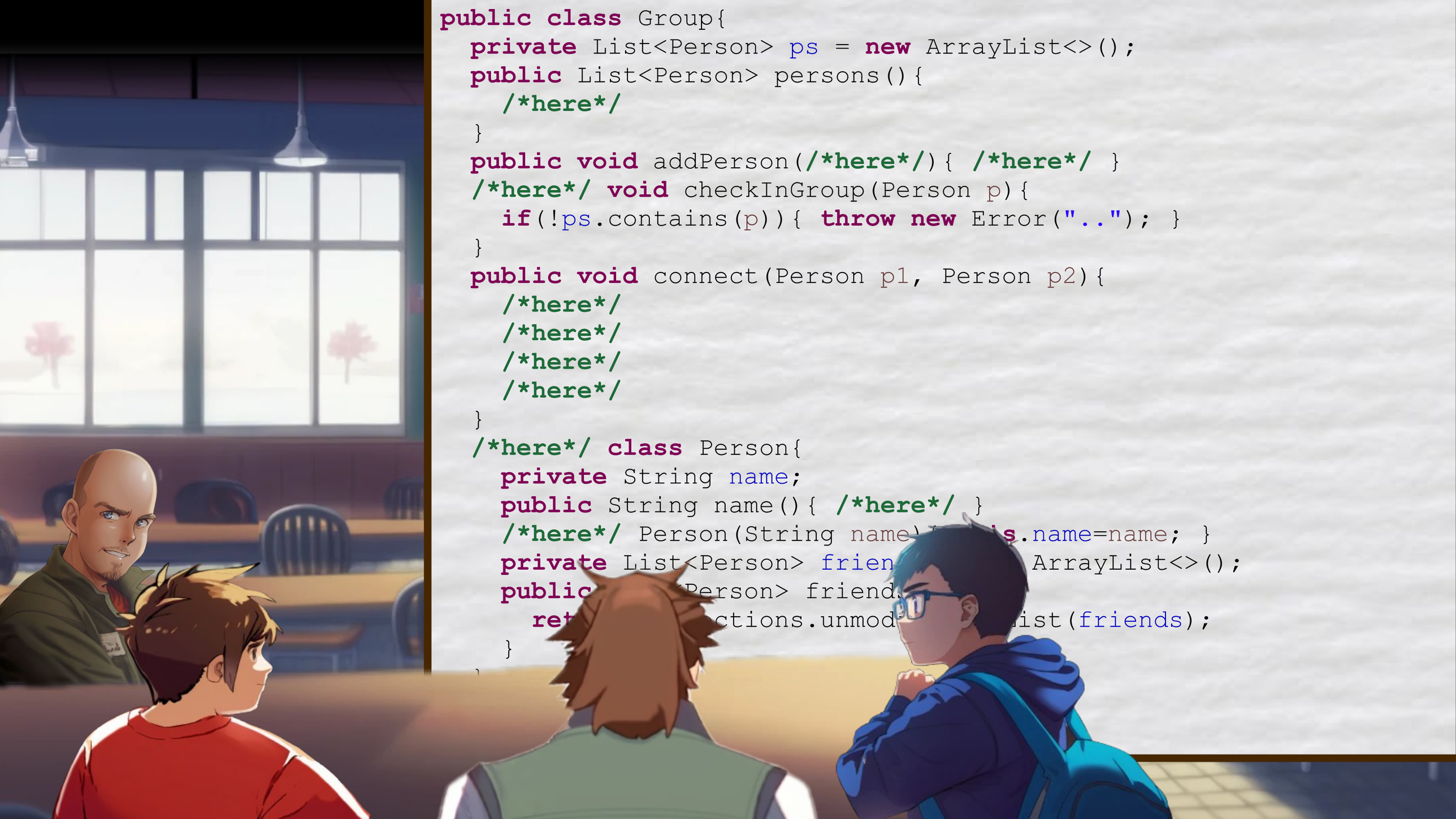
Replace every /*here*/ with appropriate code.
Depending on your indentation style, you
may or may not need more or fewer lines.

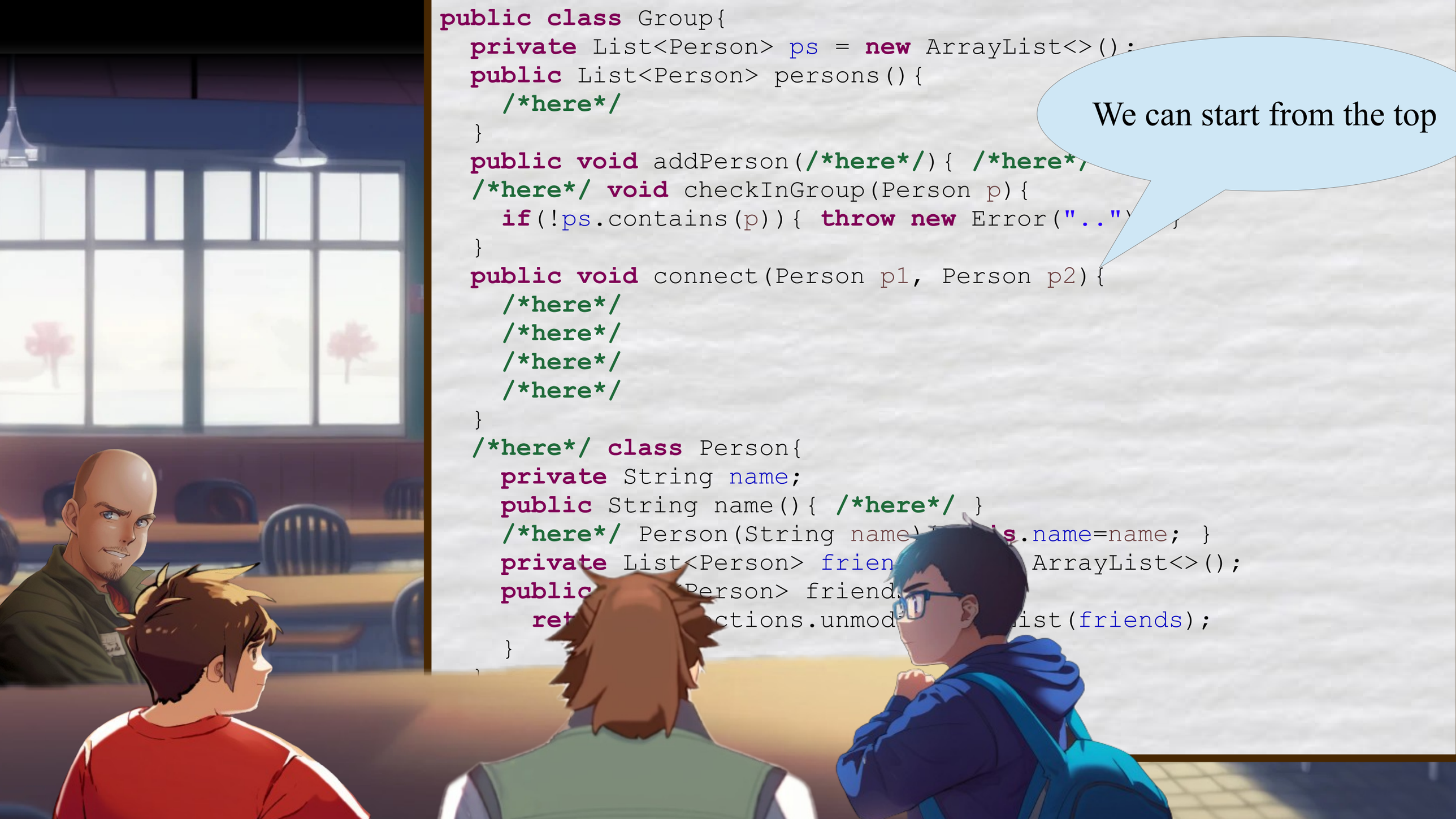
This message will disappear shortly.
You can pause the video after that.

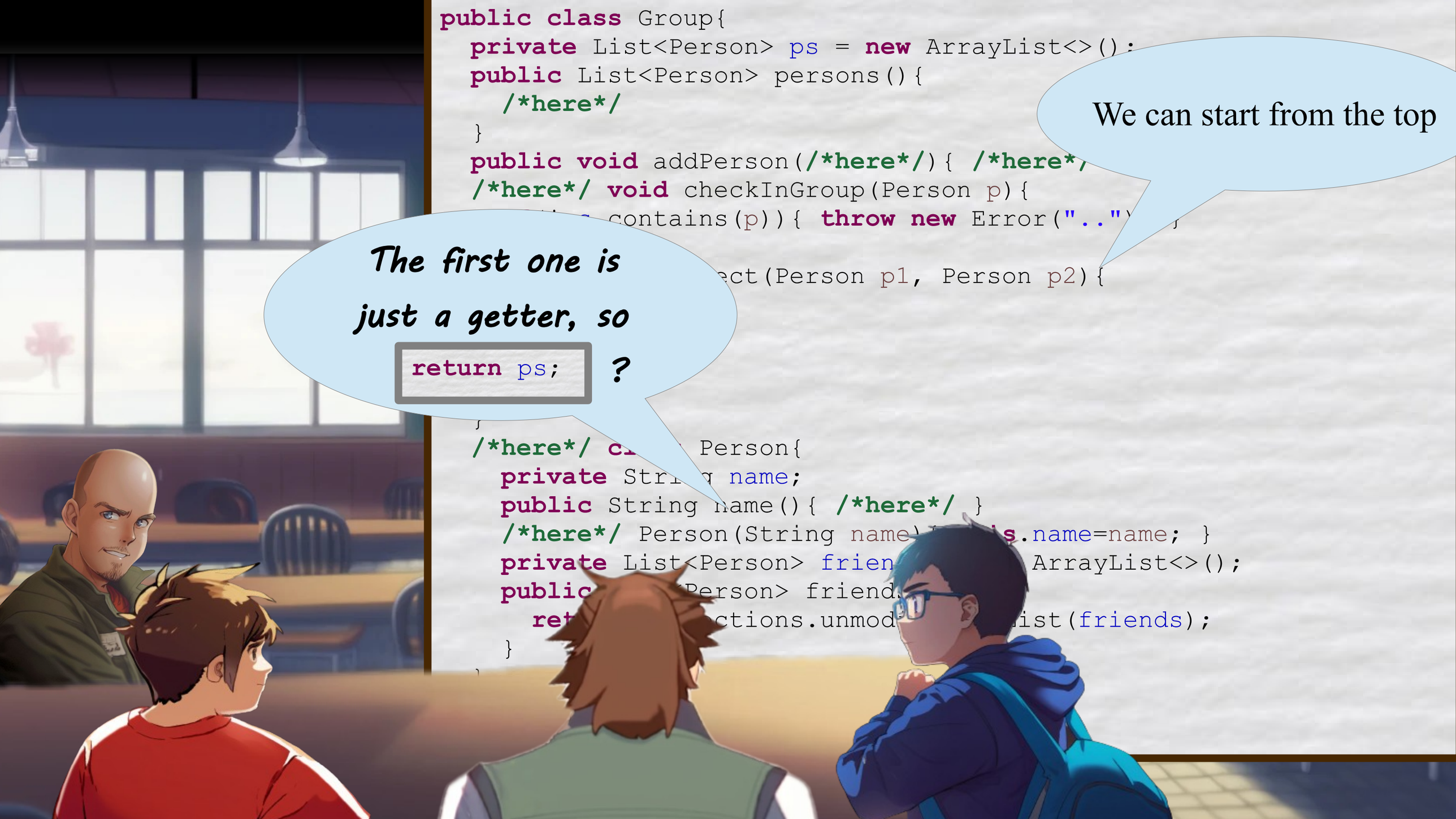


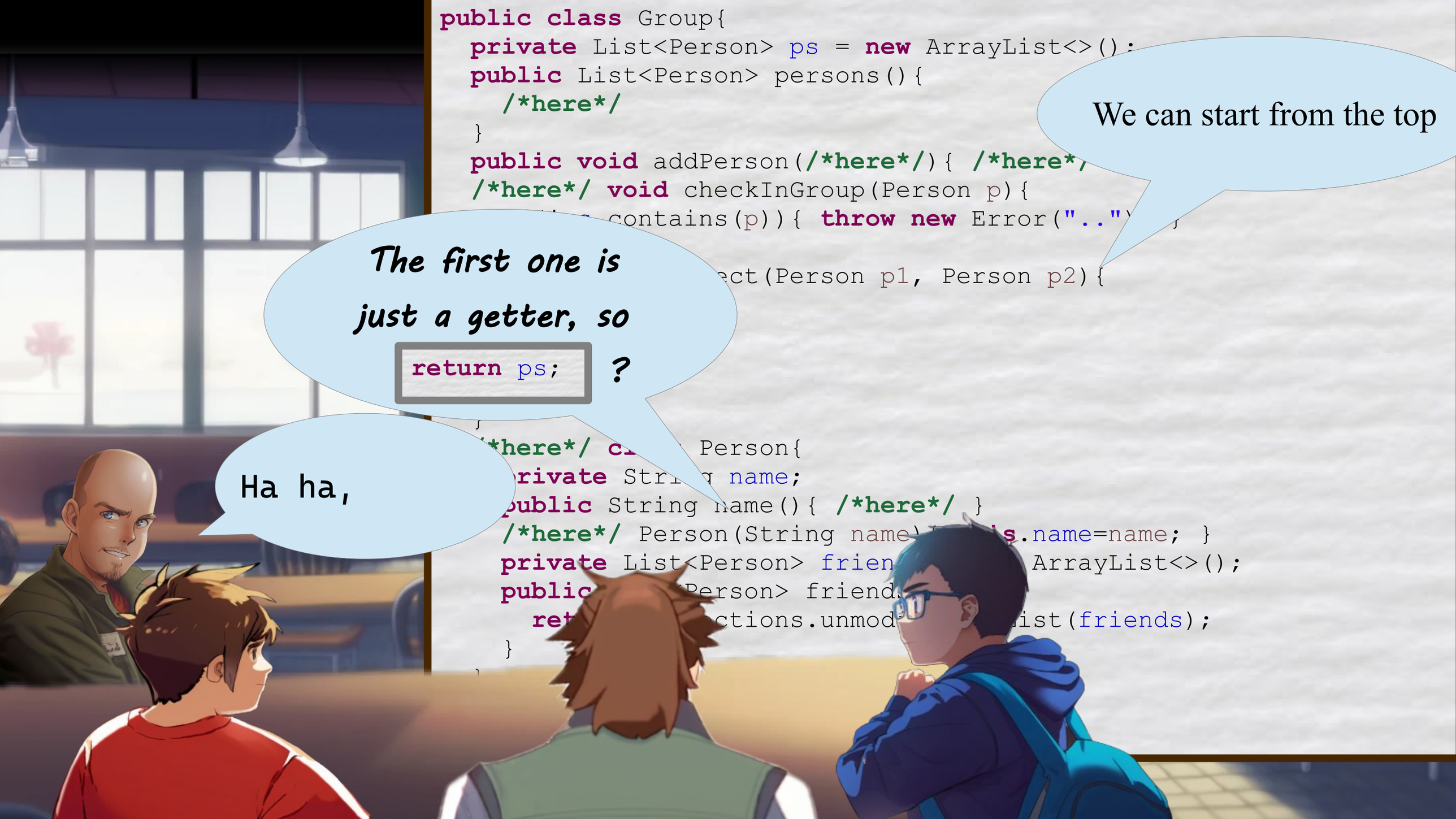
Can you complete this code to satisfy the requirements?

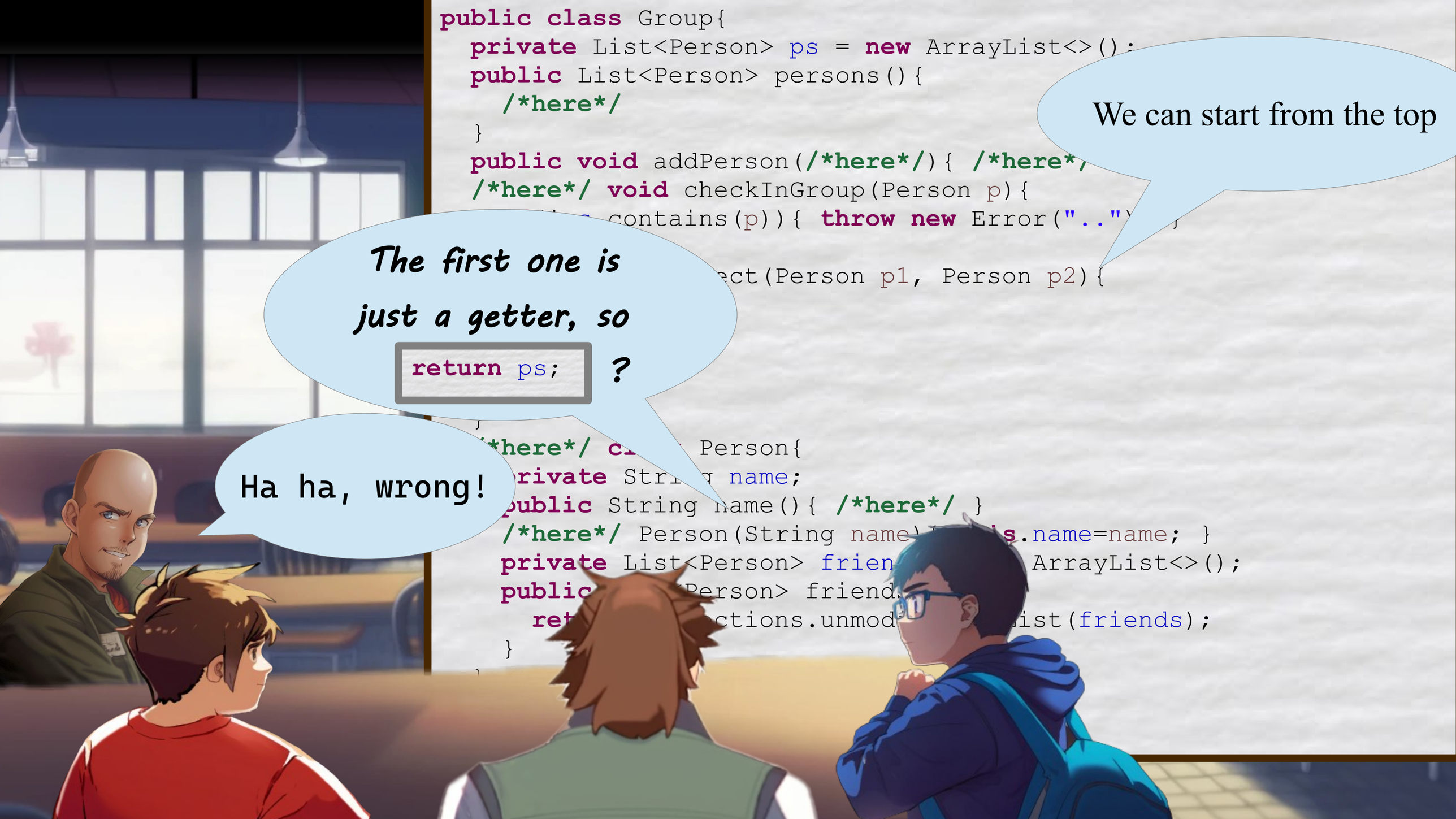
```
public class Group{//Number of required lines may vary
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    /*here*/
  }
  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
/*here*/
}
public void addPerson(/*here*/){ /*here*/ }
/*here*/ void checkInGroup(Person p){
if(!ps.contains(p)){ throw new Error(..); }
}
public void connect(Person p1, Person p2){
/*here*/
/*here*/
/*here*/
/*here*/
}
/*here*/ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name)
private List<Person>
public
ret
}

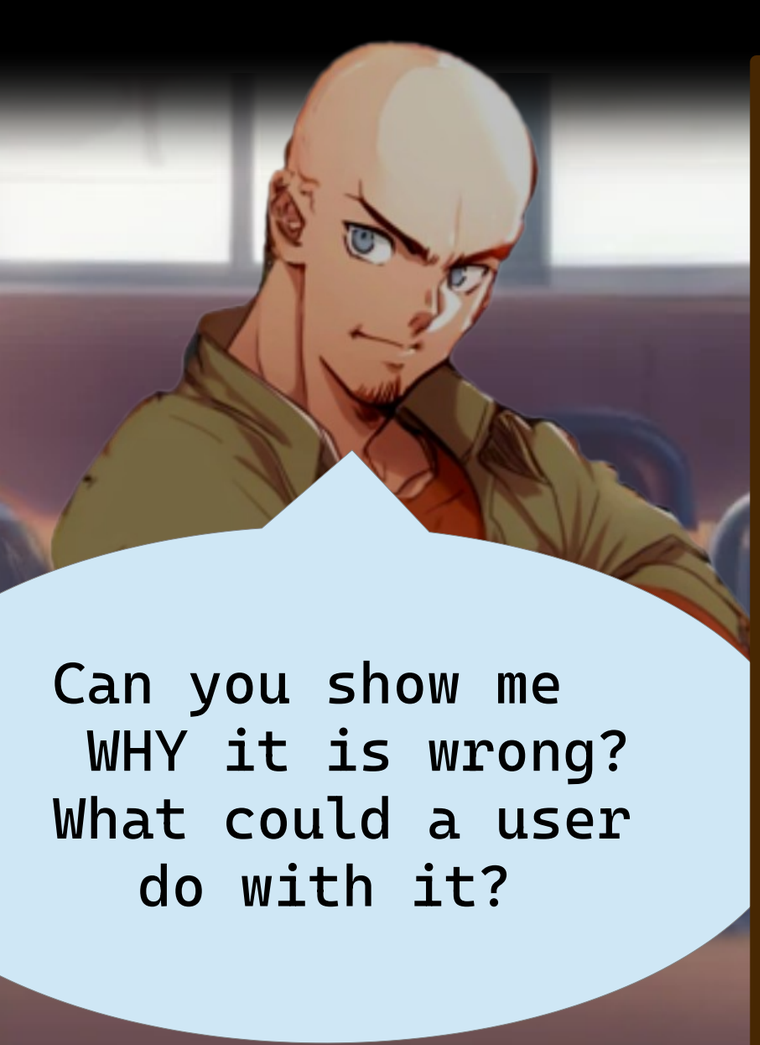

public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
/*here*/
}
public void addPerson(/*here*/){ /*here*/
/*here*/ void checkInGroup(Person p){
if(!ps.contains(p)){ throw new Error("..")
}
public void connect(Person p1, Person p2){
/*here*/
/*here*/
/*here*/
/*here*/
}
/*here*/ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name)
private List<Person> frien
public
ret
}
We can start from the top


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
/*here*/
}
public void addPerson(/*here*/){ /*here*/
/*here*/ void checkInGroup(Person p){
contains(p)){ throw new Error("..")
ect(Person p1, Person p2){
Person{
private Str g name;
public String name(){ /*here*/ }
/*here*/ Person(String name) s.name=name; }
private List<Person> frien ArrayList<>();
public Person> friend
ret ctions.unmod ist(friends);
}
The first one is just a getter, so
return ps;
?
We can start from the top


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
/*here*/
}
public void addPerson(/*here*/){ /*here*/
/*here*/ void checkInGroup(Person p){
contains(p)){ throw new Error("..")
ect(Person p1, Person p2){
here/ c Person{
rivate Str g name;
public String name(){ /*here*/ }
/*here*/ Person(String name) s.name=name; }
private List<Person> frien ArrayList<>();
public Person> friend
re ctions.unmod ist(friends);
}
The first one is just a getter, so
return ps;
?
Ha ha,
We can start from the top


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
/*here*/
}
public void addPerson(/*here*/){ /*here*/
/*here*/ void checkInGroup(Person p){
contains(p)){ throw new Error("..")
ect(Person p1, Person p2){
here/ c Person{
rivate Str g name;
public String name(){ /*here*/ }
/*here*/ Person(String name) s.name=name; }
private List<Person> frien ArrayList<>();
public Person> friend
re ctions.unmod ist(friends);
}
We can start from the top
The first one is just a getter, so
return ps;
?
Ha ha, wrong!

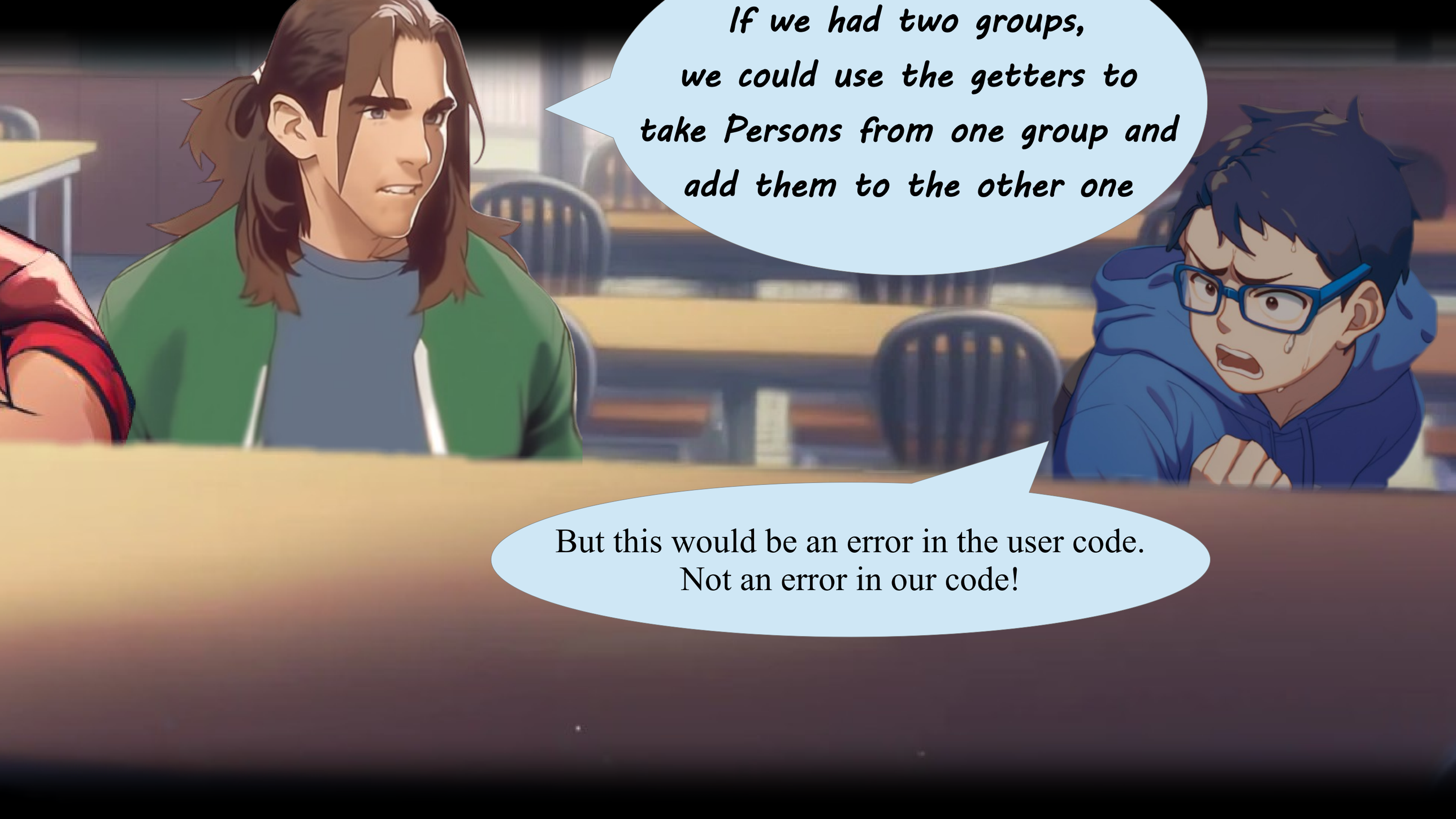
```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
      return ps;
  }
  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```



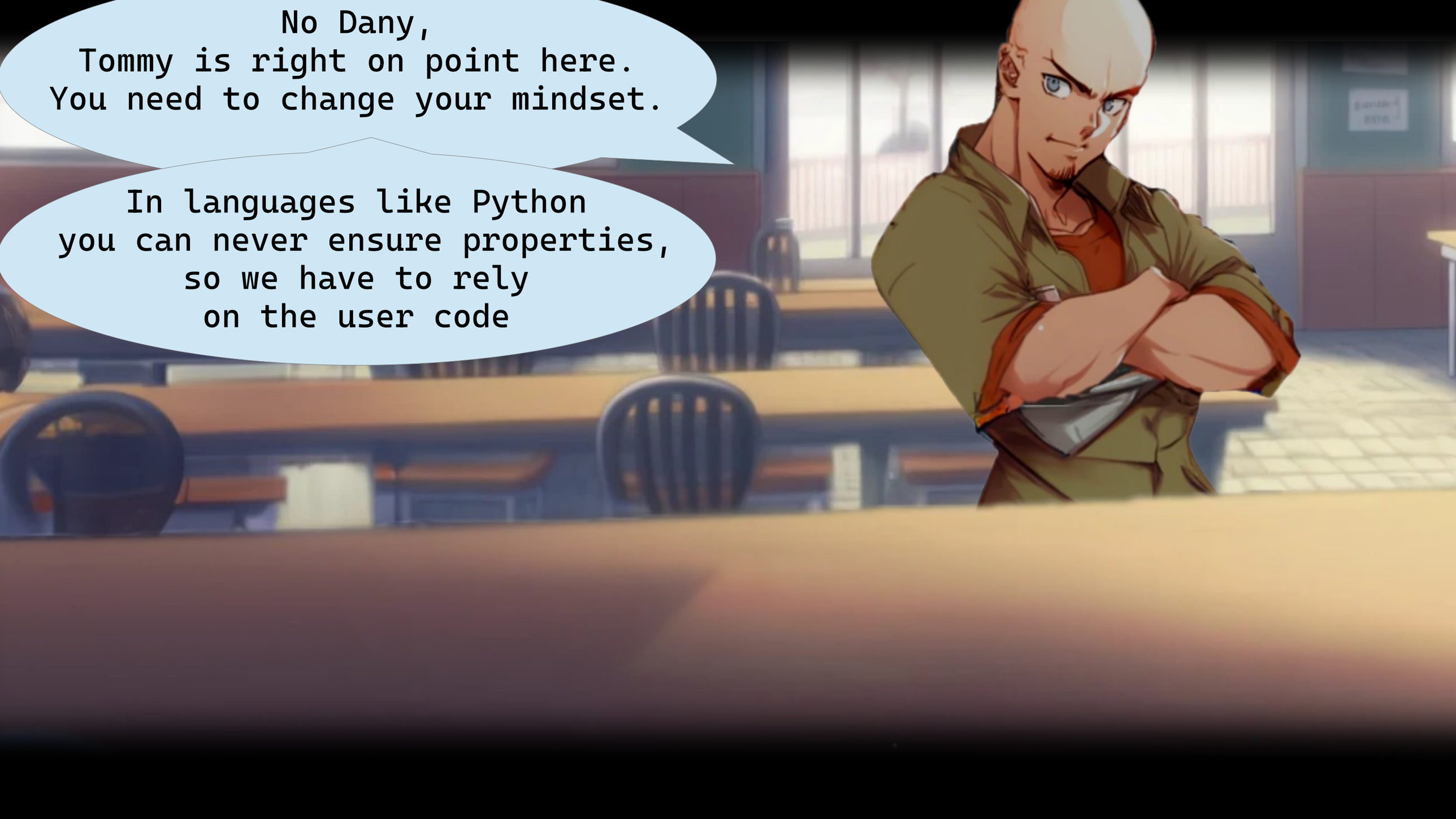
Can you show me
WHY it is wrong?
What could a user
do with it?

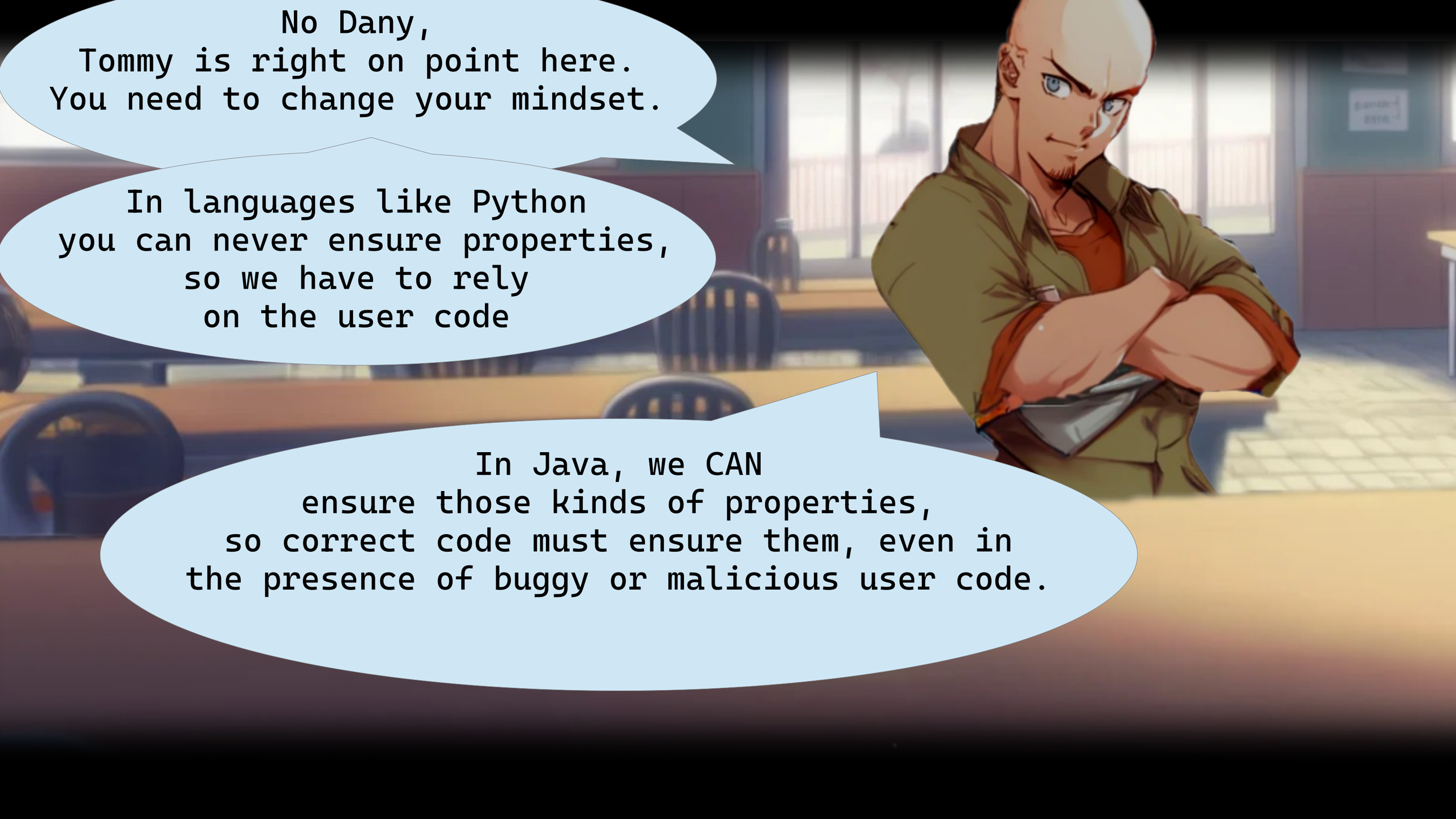
```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return ps;
  }
  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```


If we had two groups,
we could use the getters to
take Persons from one group and
add them to the other one


If we had two groups,
we could use the getters to
take Persons from one group and
add them to the other one
But this would be an error in the user code.
Not an error in our code!


No Dany,
Tommy is right on point here.
You need to change your mindset.


No Dany,
Tommy is right on point here.
You need to change your mindset.
In languages like Python
you can never ensure properties,
so we have to rely
on the user code


No Dany,
Tommy is right on point here.
You need to change your mindset.
In languages like Python
you can never ensure properties,
so we have to rely
on the user code
In Java, we CAN
ensure those kinds of properties,
so correct code must ensure them, even in
the presence of buggy or malicious user code.


So, …
‘correct code’
is not an
absolute concept


So, …
‘correct code’
is not an
absolute concept
It depends on the
language you are using.

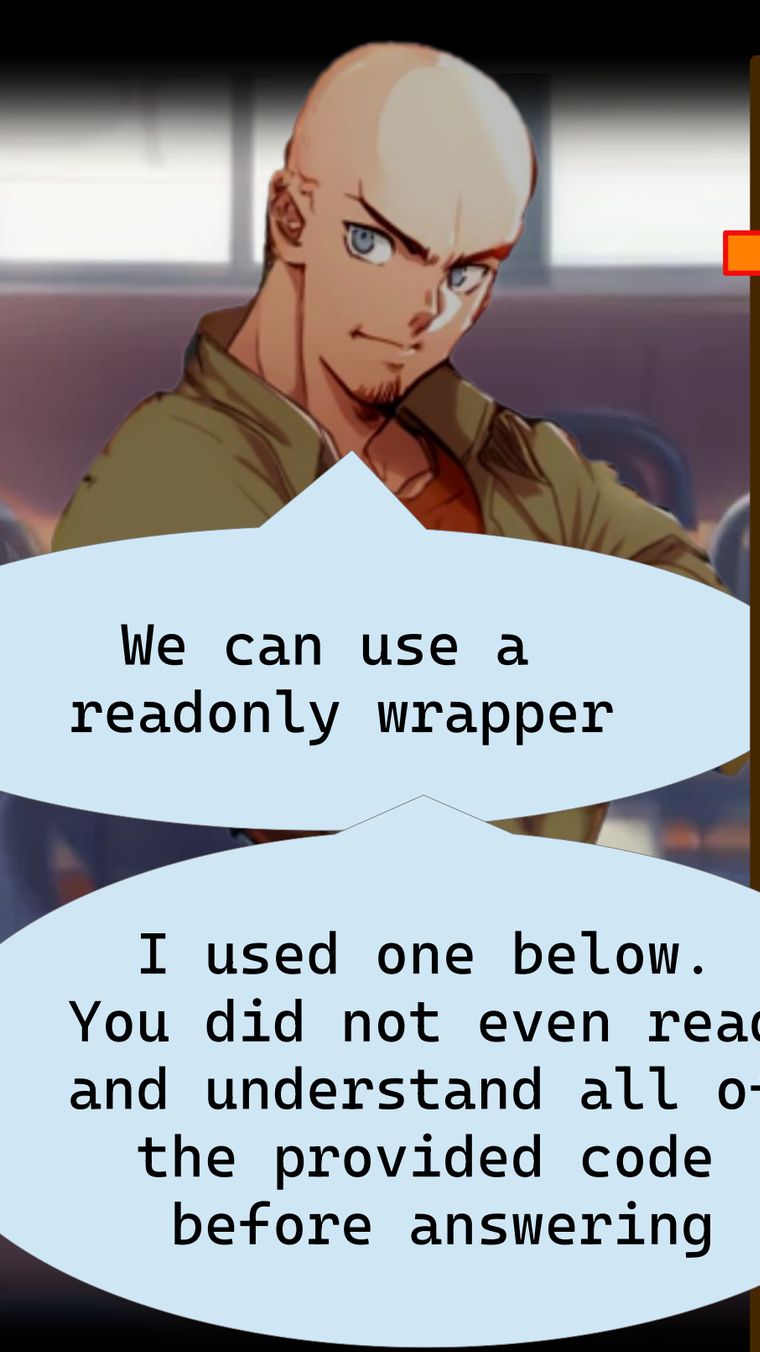

So... what should we do?


So… what should we do?
Copy the list into a new one every time someone calls the getter?


So... what should we do?
Copy the list into a new one every time someone calls the getter?
That would be ridiculously slow.

```java
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```


```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```



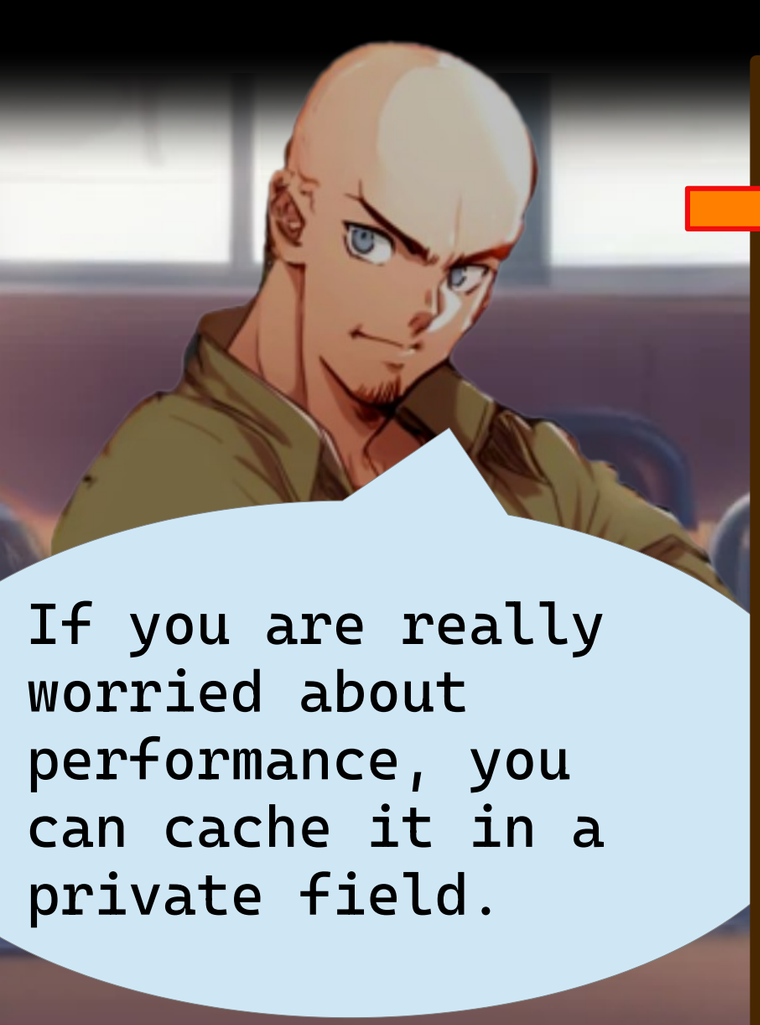
We can use a readonly wrapper

I used one below. You did not even read and understand all of the provided code before answering

```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```

```
public class Group{
  private List<Person> ps = new ArrayList<>();

  public List<Person> persons(){ return    ;}

  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```

```
public class Group{
  private List<Person> ps = new ArrayList<>();
  private List<Person> psW = Collections.unmodifiableList(ps);
  public List<Person> persons(){ return psW;}

  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```



If you are really worried about performance, you can cache it in a private field.

```
public class Group{
  private List<Person> ps = new ArrayList<>();
  private List<Person> psW = Collections.unmodifiableList(ps);
  public List<Person> persons(){ return psW;}

  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```

If you are really worried about performance, you can cache it in a private field.

But I think it is a bad idea. Java is highly optimized for small, short-lived objects

```
public class Group{
  private List<Person> ps = new ArrayList<>();
  private List<Person> psW = Collections.unmodifiableList(ps);
  public List<Person> persons(){ return psW;}

  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```

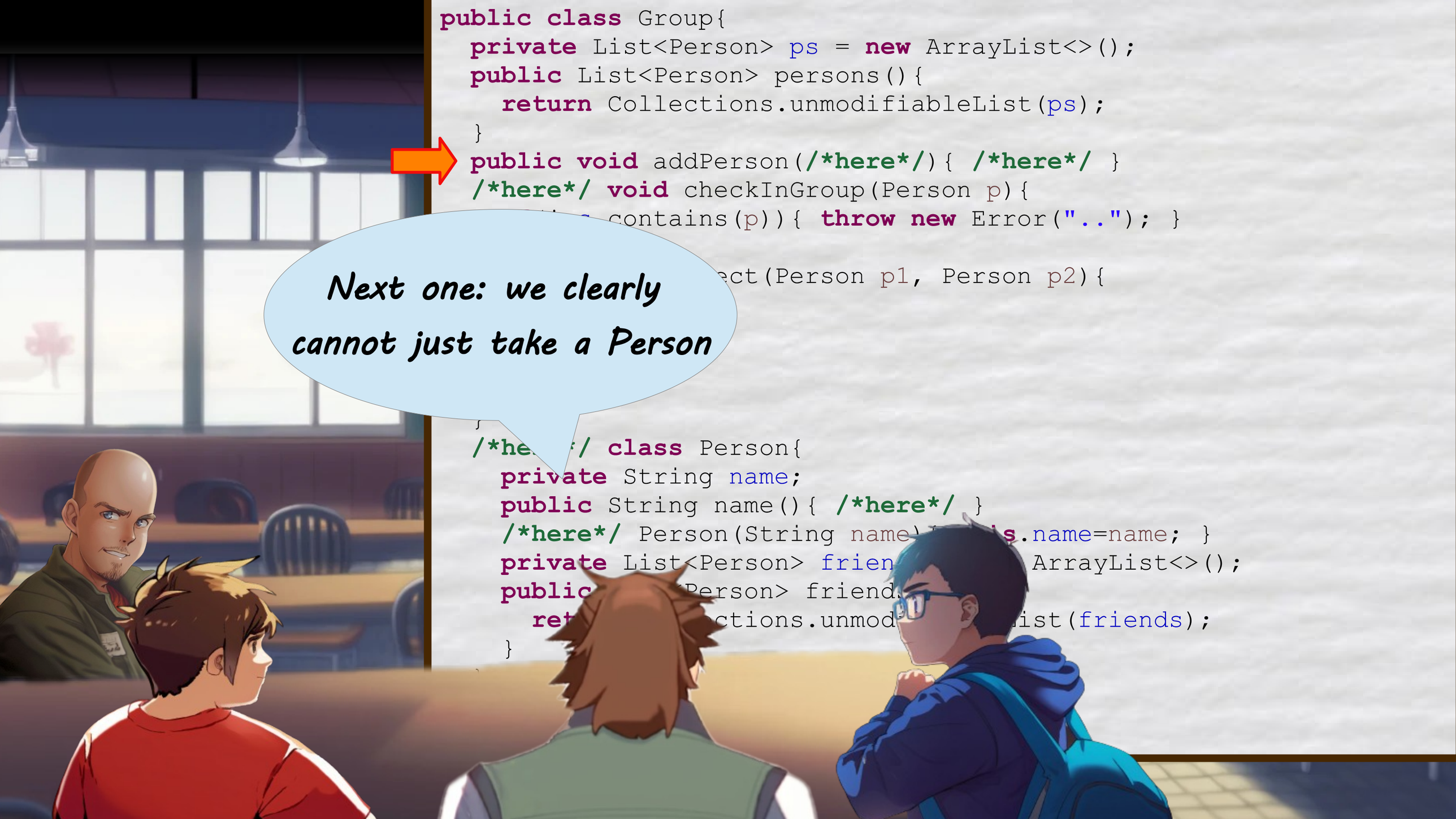
If you are really worried about performance, you can cache it in a private field.

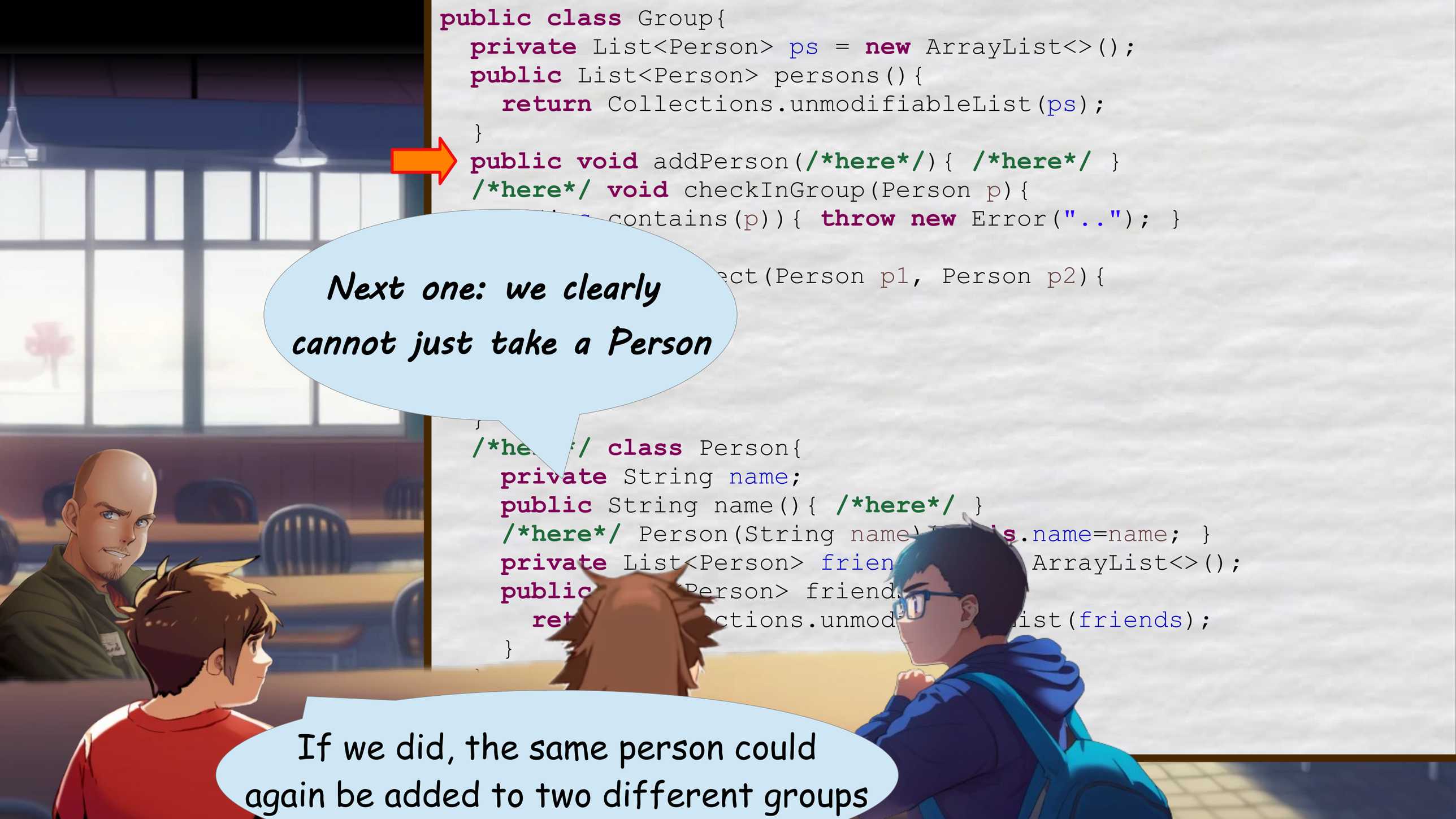
But I think it is a bad idea. Java is highly optimized for small, short-lived objects

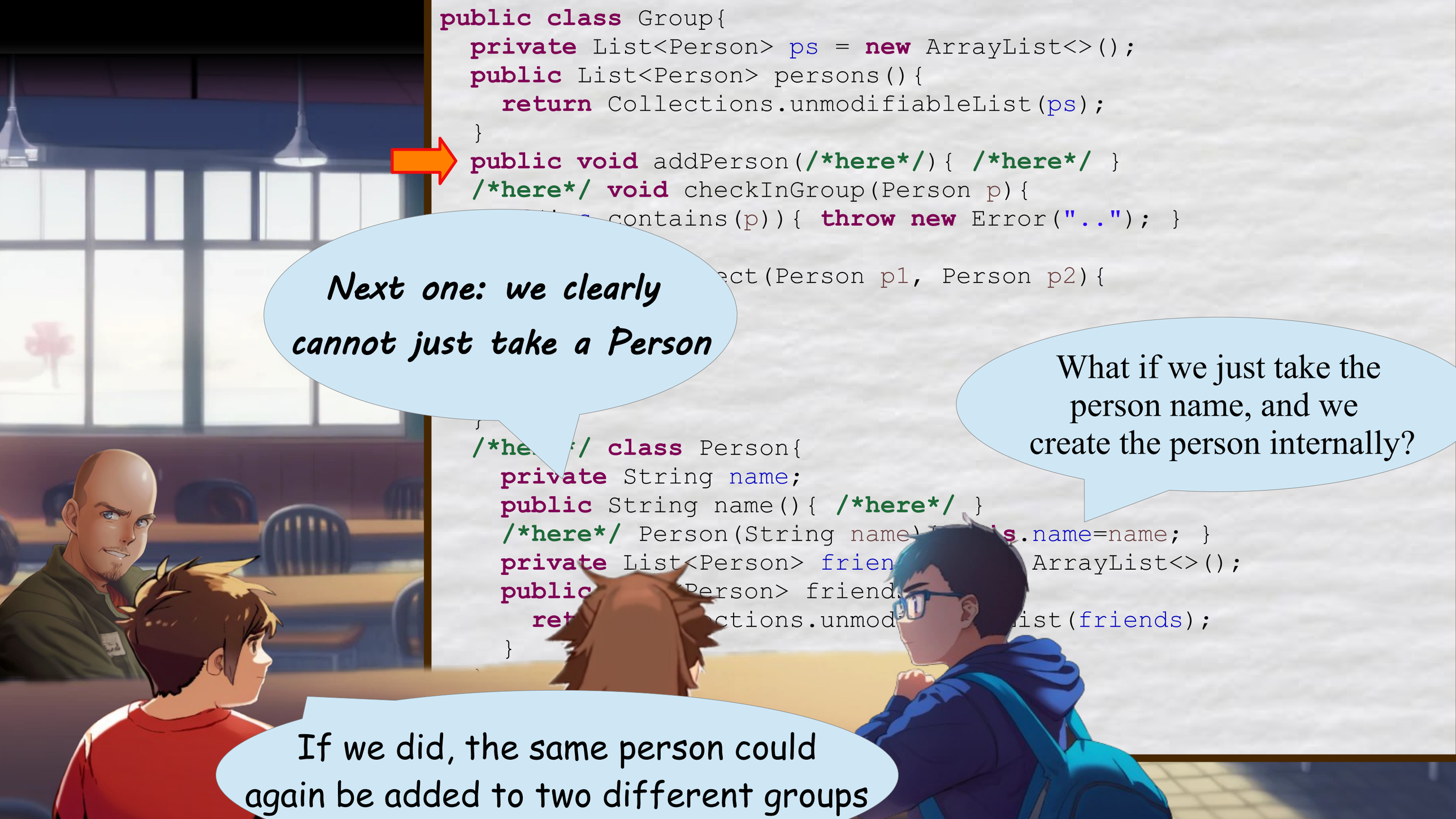
```
public class Group{
  private List<Person> ps = new ArrayList<>();
  private List<Person> psW = Collections.unmodifiableList(ps);
  public List<Person> persons(){ return psW;}

  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
```

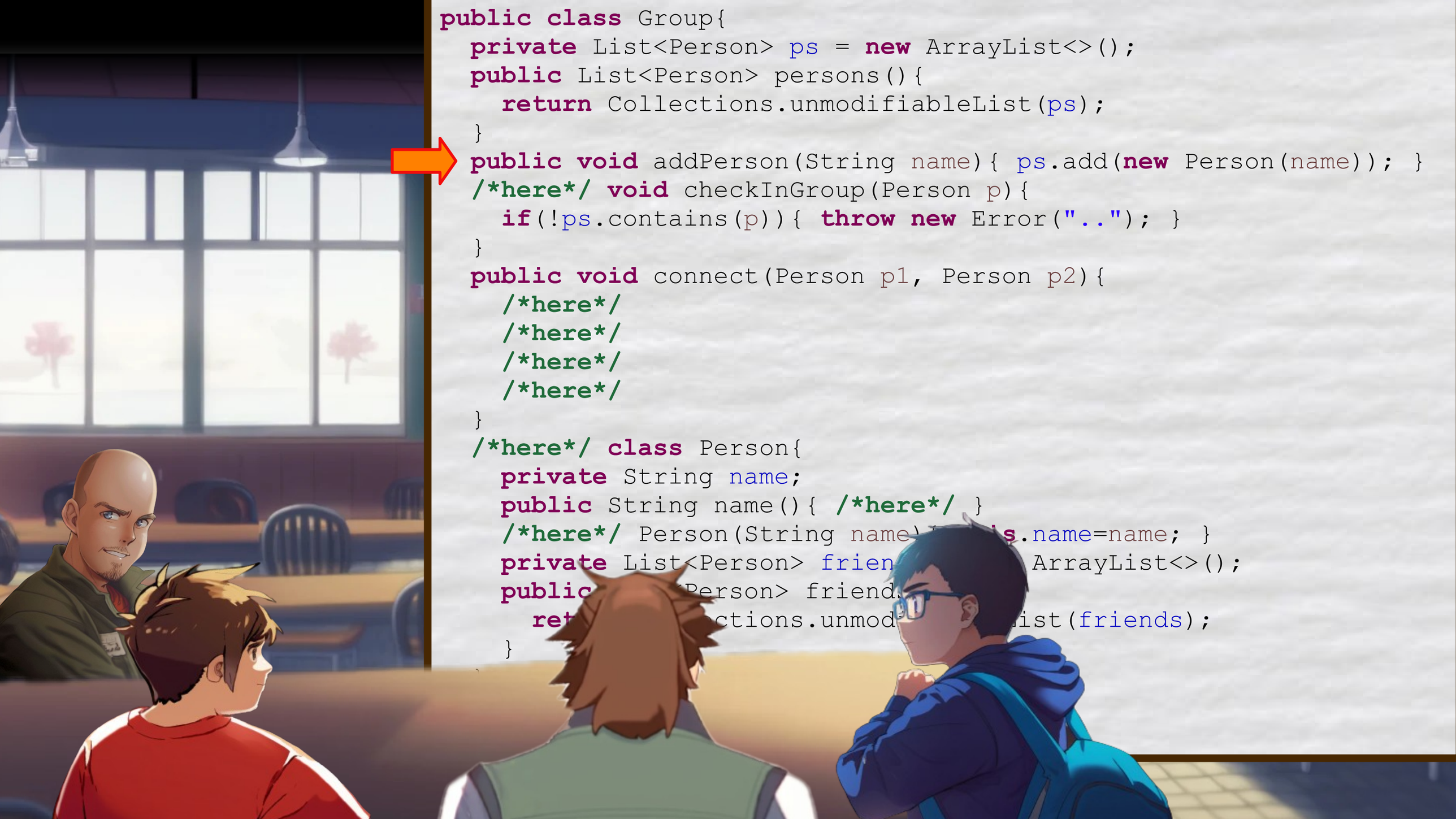
Calling Collections.unmodifiableList could be as fast as an ‘if’

```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(/*here*/){ /*here*/ }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name)
    private List<Person> frien
    public
      ret
    }
```

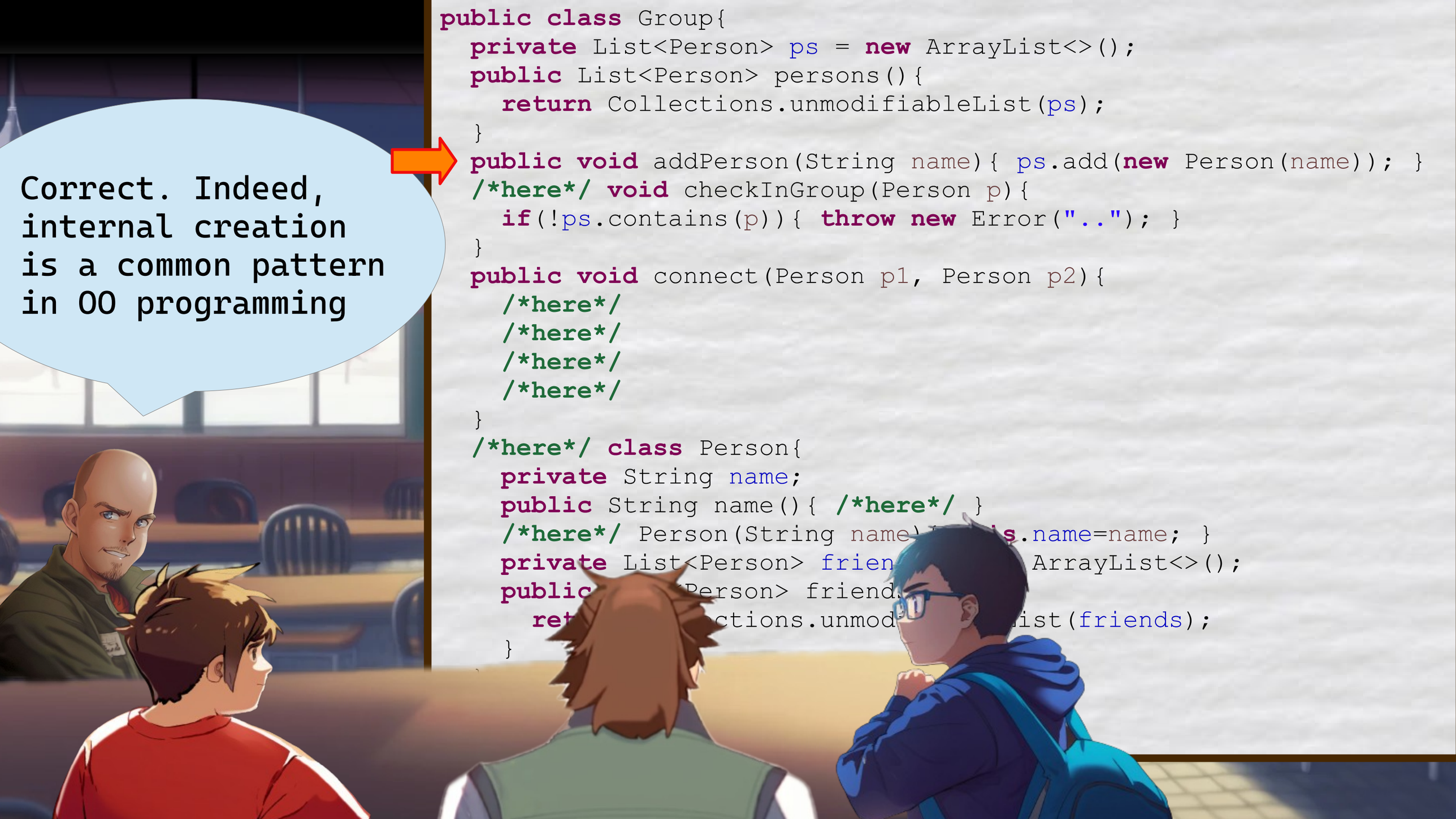

Next one: we clearly cannot just take a Person
public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
return Collections.unmodifiableList(ps);
}
public void addPerson(/*here*/){ /*here*/ }
/*here*/ void checkInGroup(Person p){
contains(p)){ throw new Error(..); }
ect(Person p1, Person p2){
/*he */ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name) { s.name=name; }
private List<Person> frien ArrayList<>();
public Person> friend
ret ctions.unmod ist(friends);
}


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
return Collections.unmodifiableList(ps);
}
public void addPerson(/*here*/){ /*here*/ }
/*here*/ void checkInGroup(Person p){
contains(p)){ throw new Error(..."); }
ect(Person p1, Person p2){
/*he */ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name) s.name=name; }
private List<Person> frien ArrayList<>();
public Person> friend
ret ctions.unmod ist(friends);
}
Next one: we clearly cannot just take a Person
If we did, the same person could again be added to two different groups


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
return Collections.unmodifiableList(ps);
}
public void addPerson(/*here*/){ /*here*/ }
/*here*/ void checkInGroup(Person p){
contains(p)){ throw new Error(".."); }
ect(Person p1, Person p2){
/*he */ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name) s.name=name; }
private List<Person> frien ArrayList<>();
public Person> friend
ret ctions.unmod ist(friends);
}
Next one: we clearly cannot just take a Person
What if we just take the person name, and we create the person internally?
If we did, the same person could again be added to two different groups



```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```




```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name)
    private List<Person> frien
    public
      ret
    }
```

```java
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```

Is this asking for the modifier of the checkInGroup method?

```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/

    ere*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
  }
}
```

Is this asking for the modifier of the checkInGroup method?

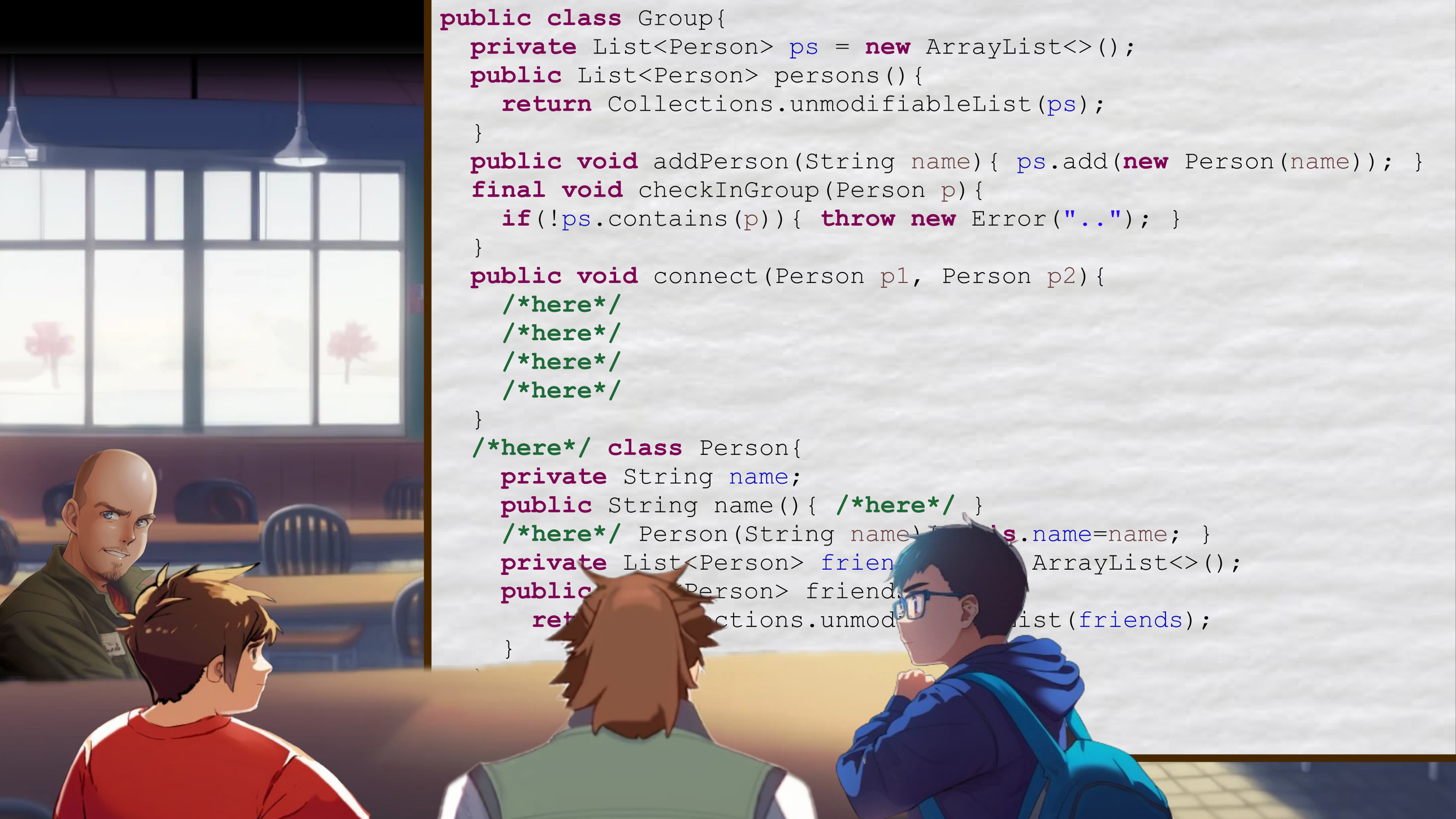
I've learned the trick here! It must be final, otherwise the user may override what it does...

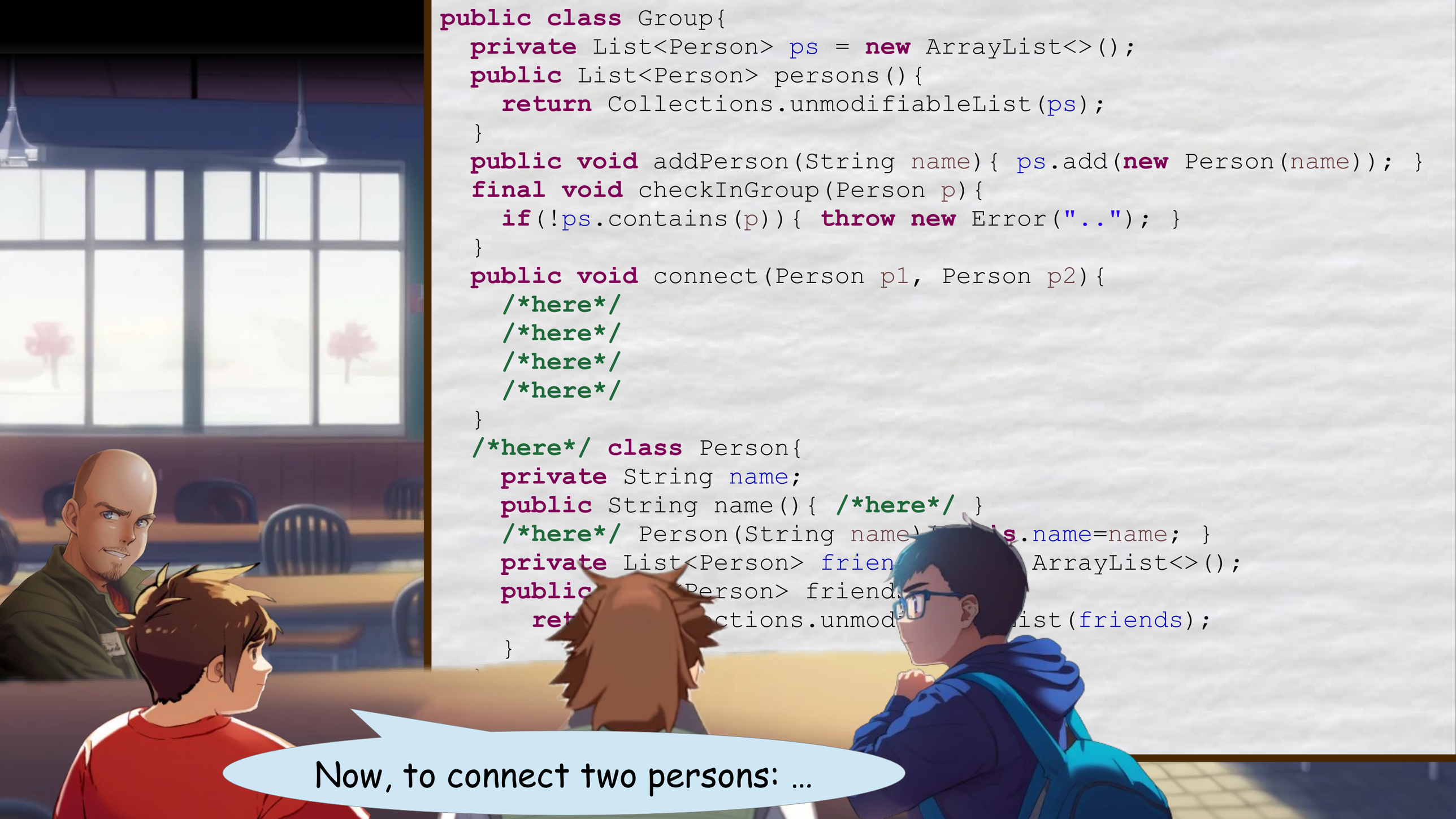
```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  /*here*/ void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    /*here*/
    /*here*/
    /*here*/
    /*here*/

    ere*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
      vate List<Person> friends = new ArrayList<>();
        List<Person> friends(){
          Collections.unmodifiableList(friends);
```


Very good!
You are starting to think defensively!


Very good!
You are starting to think defensively!
The Group class is not final.


Very good!
You are starting to think defensively!
The Group class is not final.
So, that method must be either final or private


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
return Collections.unmodifiableList(ps);
}
public void addPerson(String name){ ps.add(new Person(name)); }
final void checkInGroup(Person p){
if(!ps.contains(p)){ throw new Error(".."); }
}
public void connect(Person p1, Person p2){
/*here*/
/*here*/
/*here*/
/*here*/
}
/*here*/ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name)
private List<Person> frien
public
ret
}


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
return Collections.unmodifiableList(ps);
}
public void addPerson(String name){ ps.add(new Person(name)); }
final void checkInGroup(Person p){
if(!ps.contains(p)){ throw new Error(".."); }
}
public void connect(Person p1, Person p2){
/*here*/
/*here*/
/*here*/
/*here*/
}
/*here*/ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name)
private List<Person> frien
public
ret
}
Now, to connect two persons: ...


First we must check if they are in the current group

First we must check if they are in the current group

```
final void checkInGroup(Person p){
  if(!ps.contains(p)){ throw new Error(".."); }
}
public void connect(Person p1, Person p2){
  checkInGroup(p1);  checkInGroup(p2);
  /*here*/
  /*here*/
  /*here*/
}
```

First we must check if they are in the current group

```
final void checkInGroup(Person p){
  if(!ps.contains(p)){ throw new Error(".."); }
}
public void connect(Person p1, Person p2){
  checkInGroup(p1);  checkInGroup(p2);
  /*here*/
  /*here*/
  /*here*/
}
```

But I'm worried, since this would cause a dynamic error.

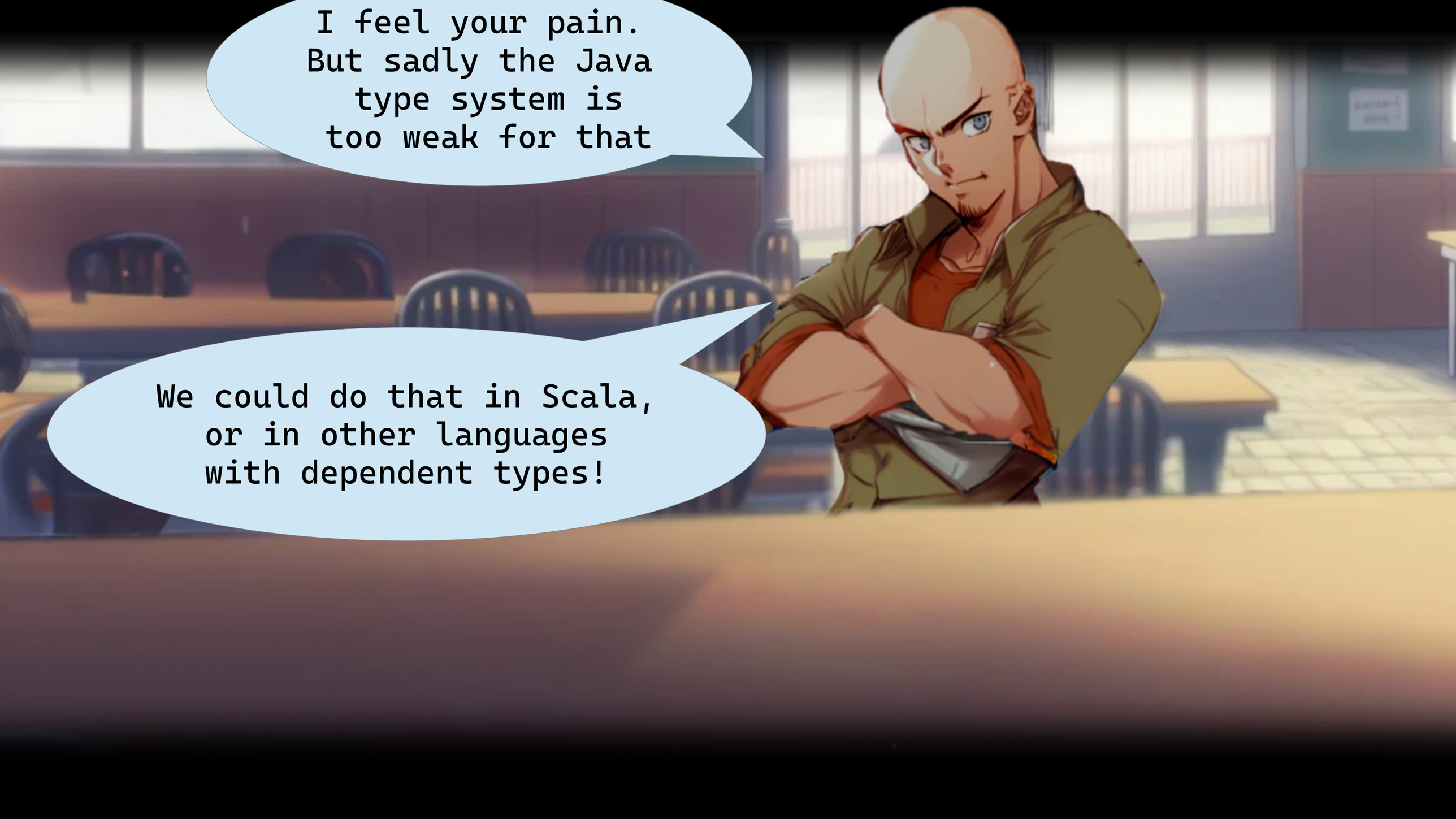
First we must check if they are in the current group

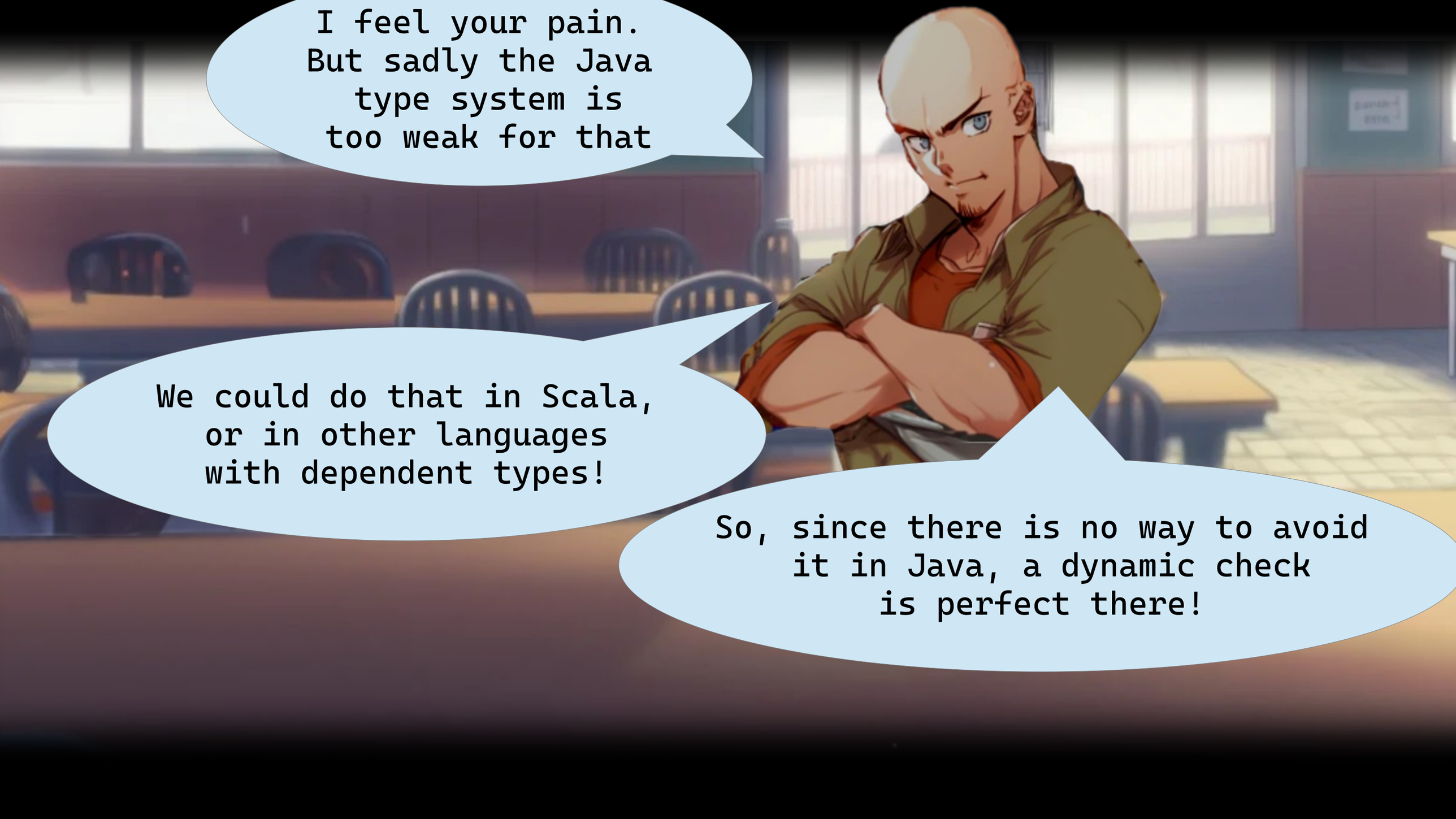
```
final void checkInGroup(Person p){
  if(!ps.contains(p)){ throw new Error(".."); }
}
public void connect(Person p1, Person p2){
  checkInGroup(p1);  checkInGroup(p2);
  /*here*/
  /*here*/
  /*here*/
}
```

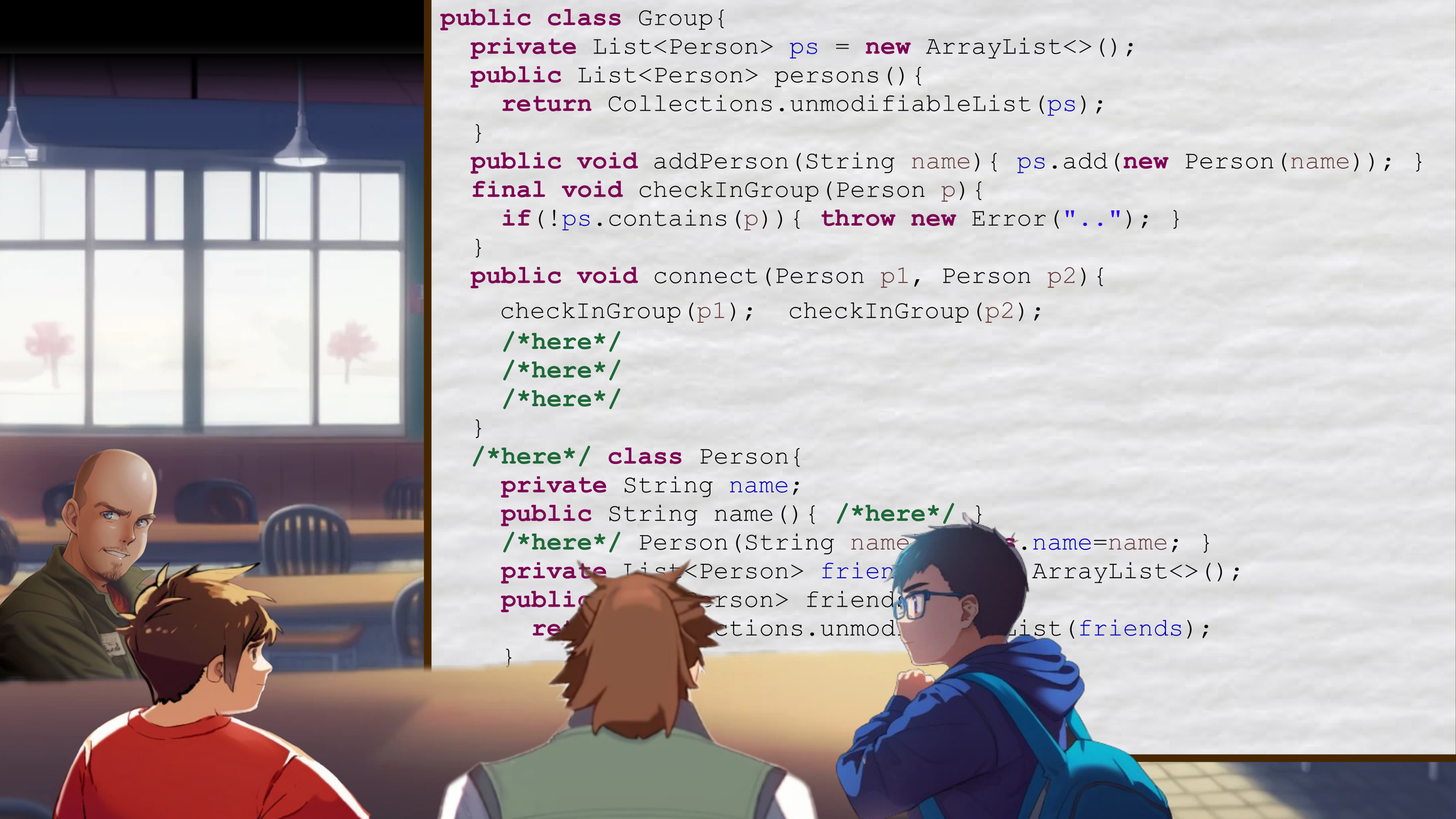
But I'm worried, since this would cause a dynamic error.

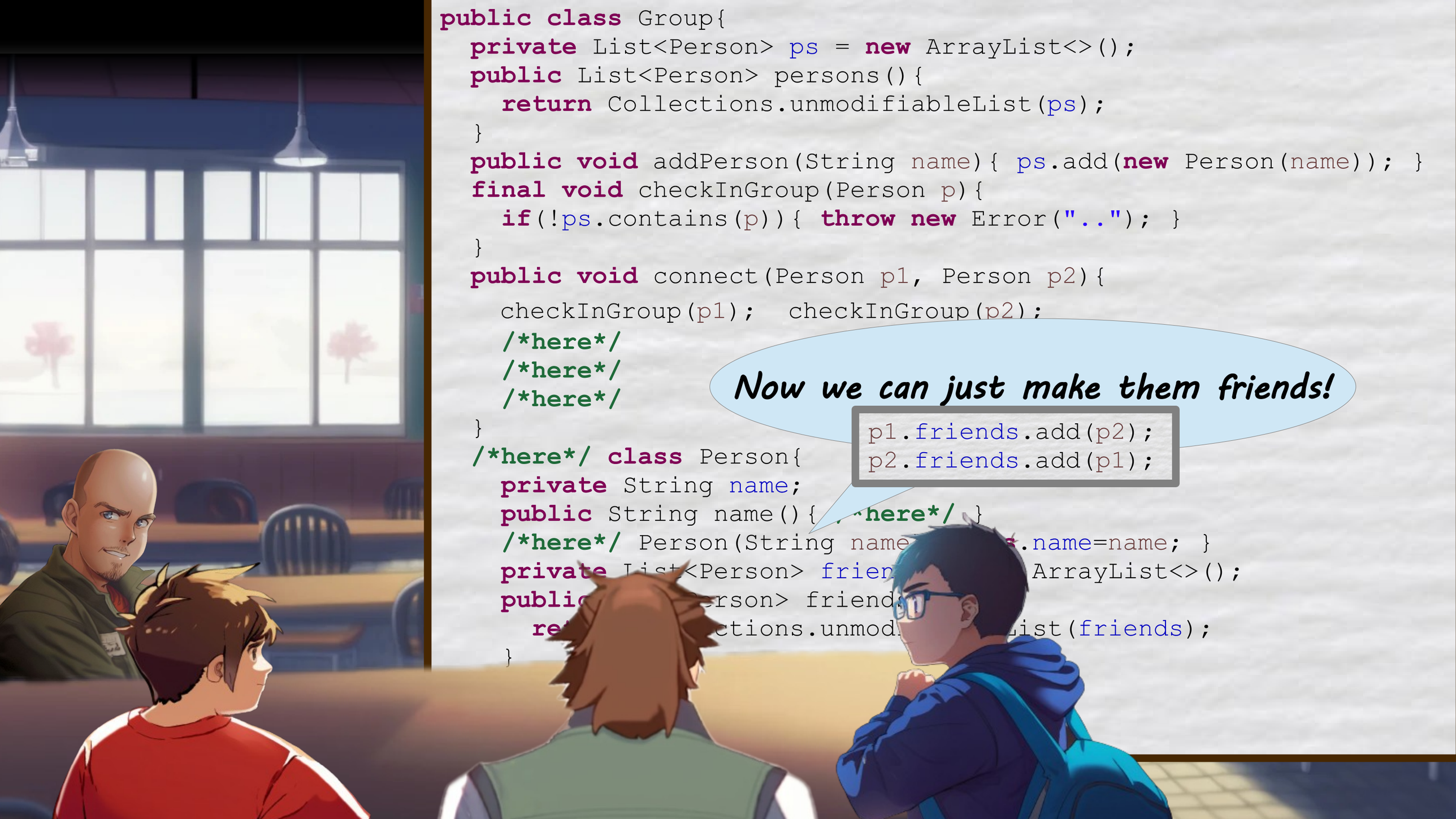
Could we catch this at compile time instead?


I feel your pain.
But sadly the Java
type system is
too weak for that


I feel your pain.
But sadly the Java
type system is
too weak for that
We could do that in Scala,
or in other languages
with dependent types!

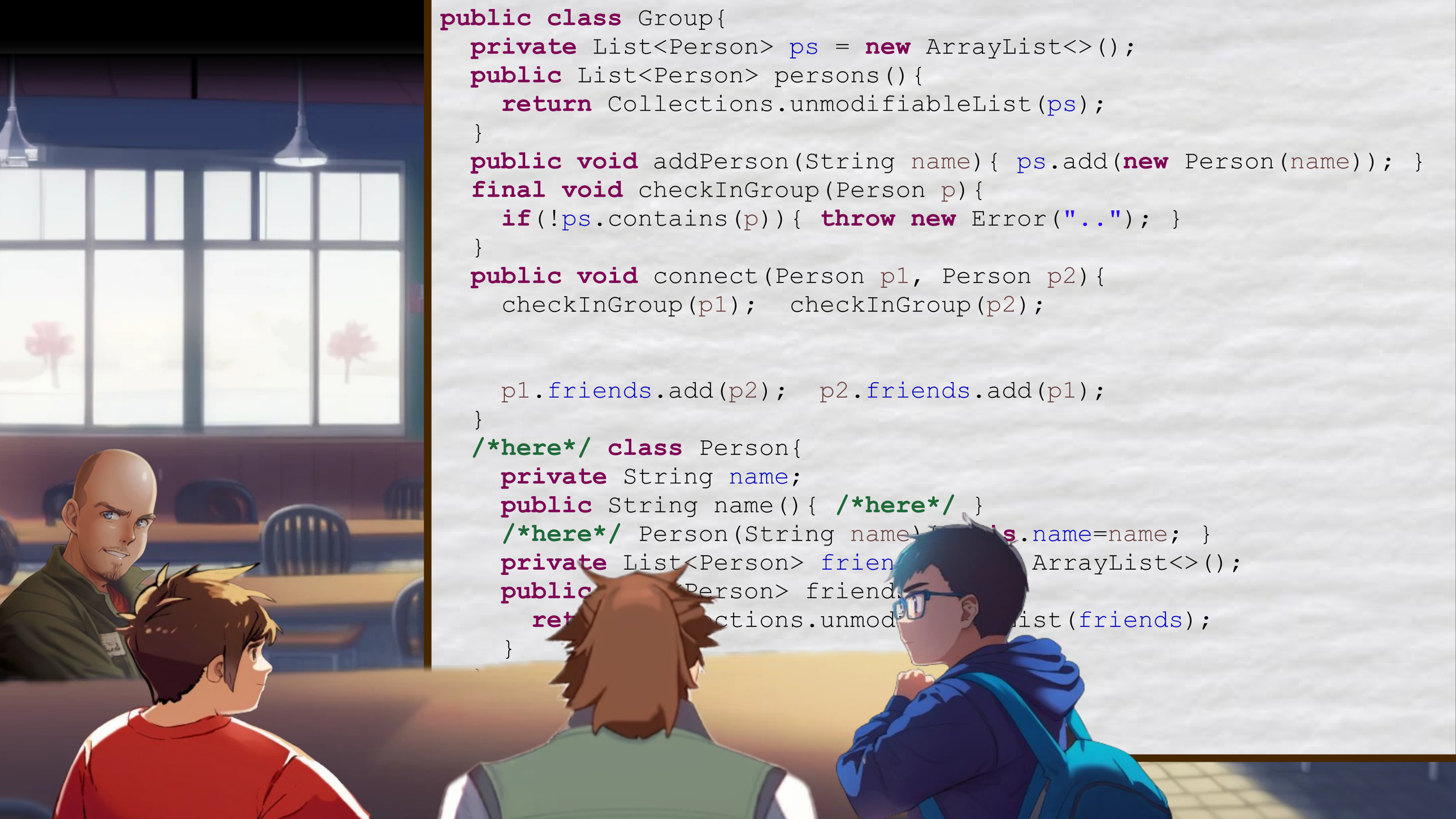

I feel your pain.
But sadly the Java
type system is
too weak for that
We could do that in Scala,
or in other languages
with dependent types!
So, since there is no way to avoid
it in Java, a dynamic check
is perfect there!


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
return Collections.unmodifiableList(ps);
}
public void addPerson(String name){ ps.add(new Person(name)); }
final void checkInGroup(Person p){
if(!ps.contains(p)){ throw new Error(".."); }
}
public void connect(Person p1, Person p2){
checkInGroup(p1); checkInGroup(p2);
/*here*/
/*here*/
/*here*/
}
/*here*/ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name
.name=name; }
private
ArrayList<>();
public
return
List(friends);
}

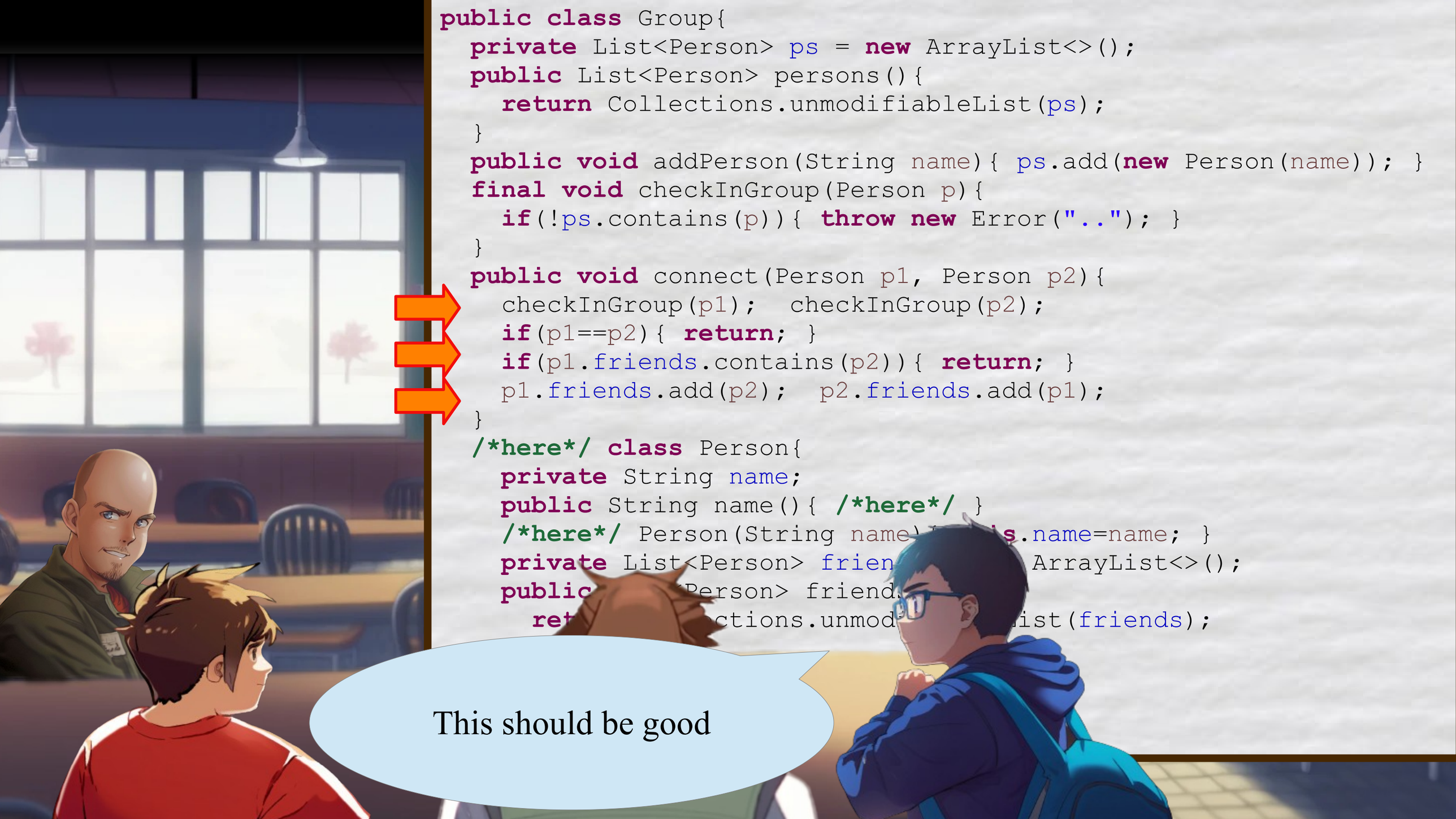

public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
return Collections.unmodifiableList(ps);
}
public void addPerson(String name){ ps.add(new Person(name)); }
final void checkInGroup(Person p){
if(!ps.contains(p)){ throw new Error(".."); }
}
public void connect(Person p1, Person p2){
checkInGroup(p1); checkInGroup(p2);
/*here*/
/*here*/
/*here*/
}
/*here*/ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name .name=name; }
private List<Person> frien ArrayList<>();
public rson> friend
re ctions.unmod List(friends);
}
Now we can just make them friends!
p1.friends.add(p2);
p2.friends.add(p1);


What if they are already friends?

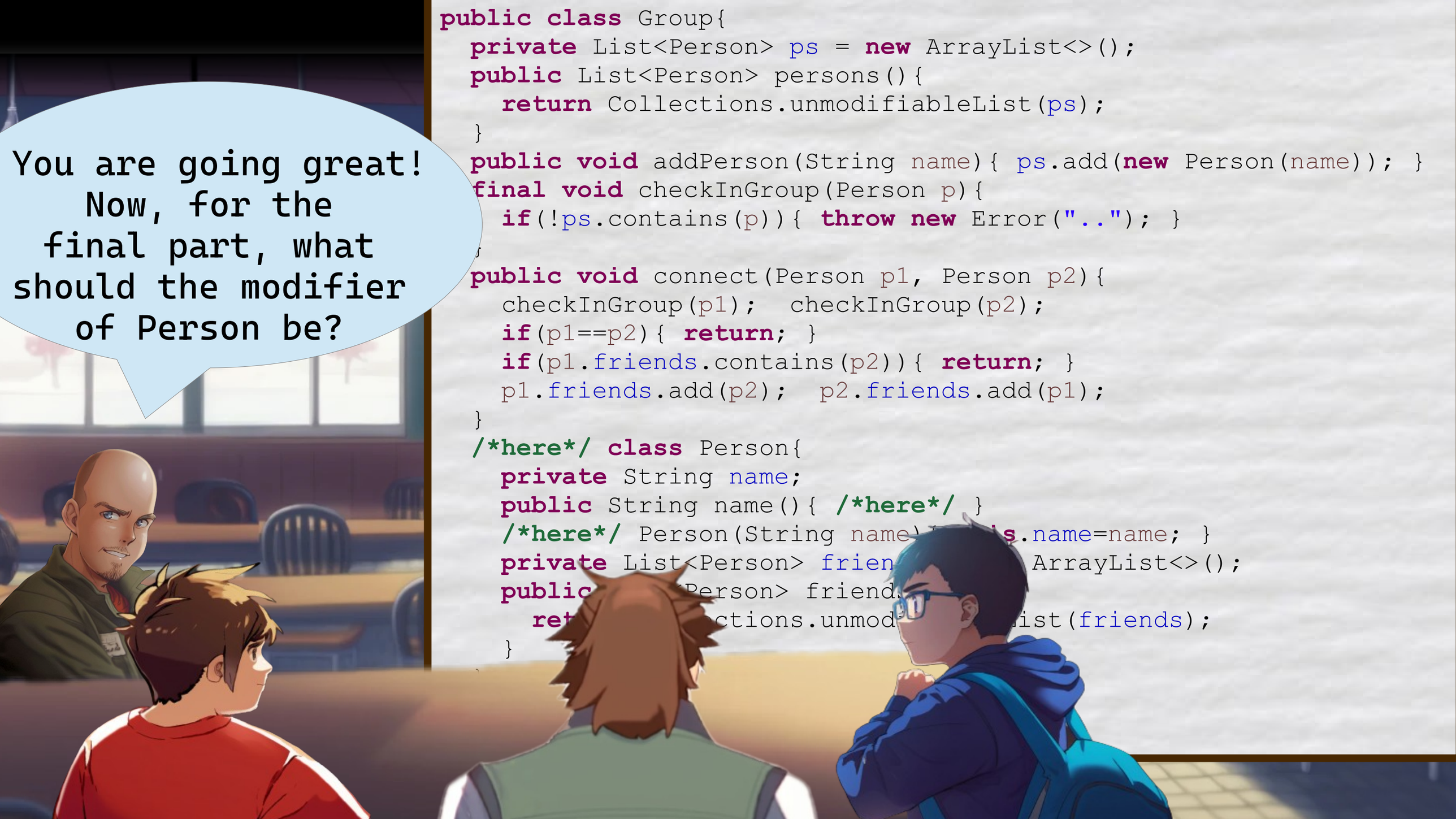

What if they are already friends?
Even worse, what if
the two persons are
actually the same person?



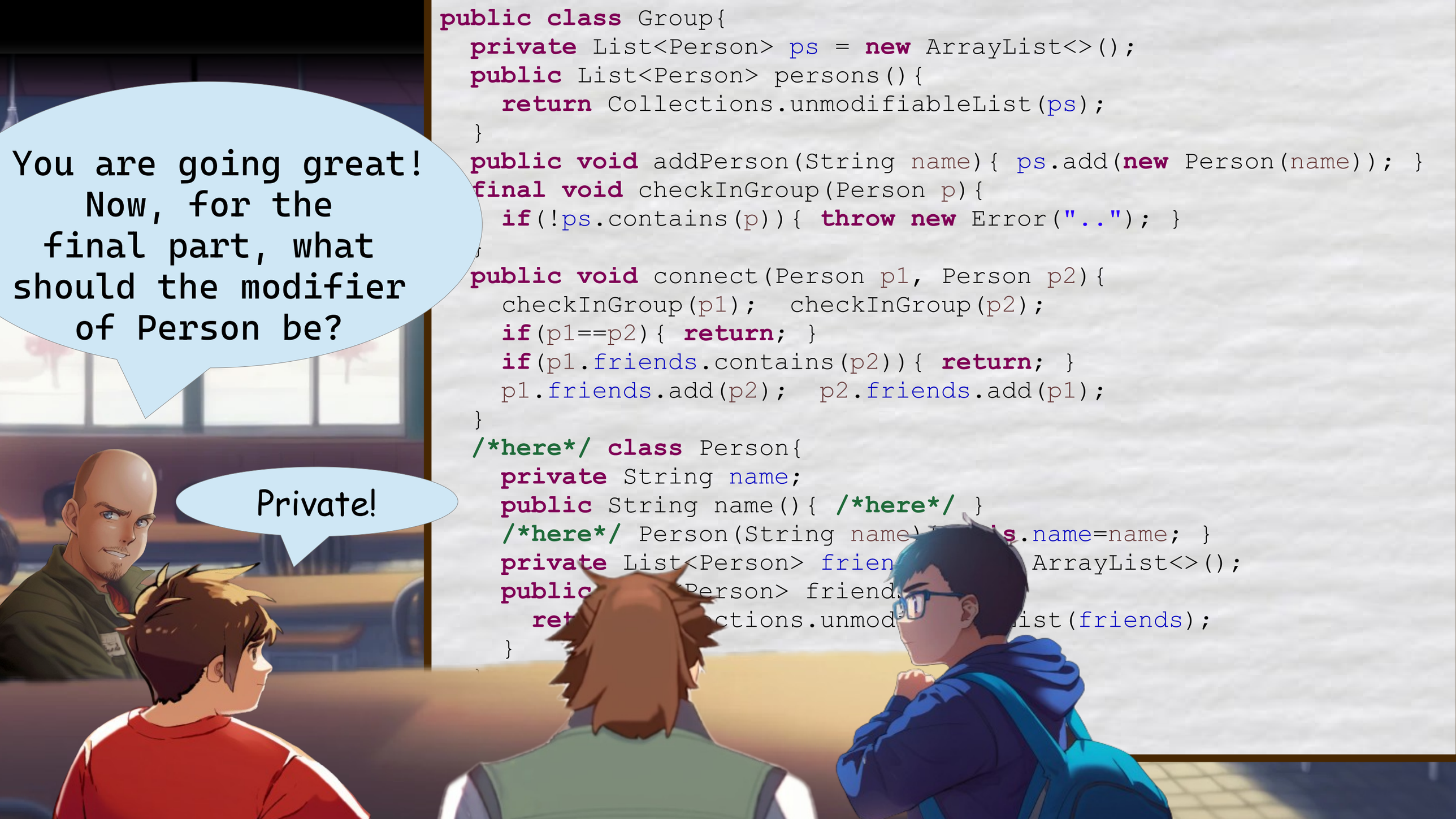
```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);


    p1.friends.add(p2);  p2.friends.add(p1);
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```


public class Group{
private List<Person> ps = new ArrayList<>();
public List<Person> persons(){
return Collections.unmodifiableList(ps);
}
public void addPerson(String name){ ps.add(new Person(name)); }
final void checkInGroup(Person p){
if(!ps.contains(p)){ throw new Error(".."); }
}
public void connect(Person p1, Person p2){
checkInGroup(p1); checkInGroup(p2);
if(p1==p2){ return; }
if(p1.friends.contains(p2)){ return; }
p1.friends.add(p2); p2.friends.add(p1);
}
/*here*/ class Person{
private String name;
public String name(){ /*here*/ }
/*here*/ Person(String name)
private List<Person> frien
public
ret
This should be good

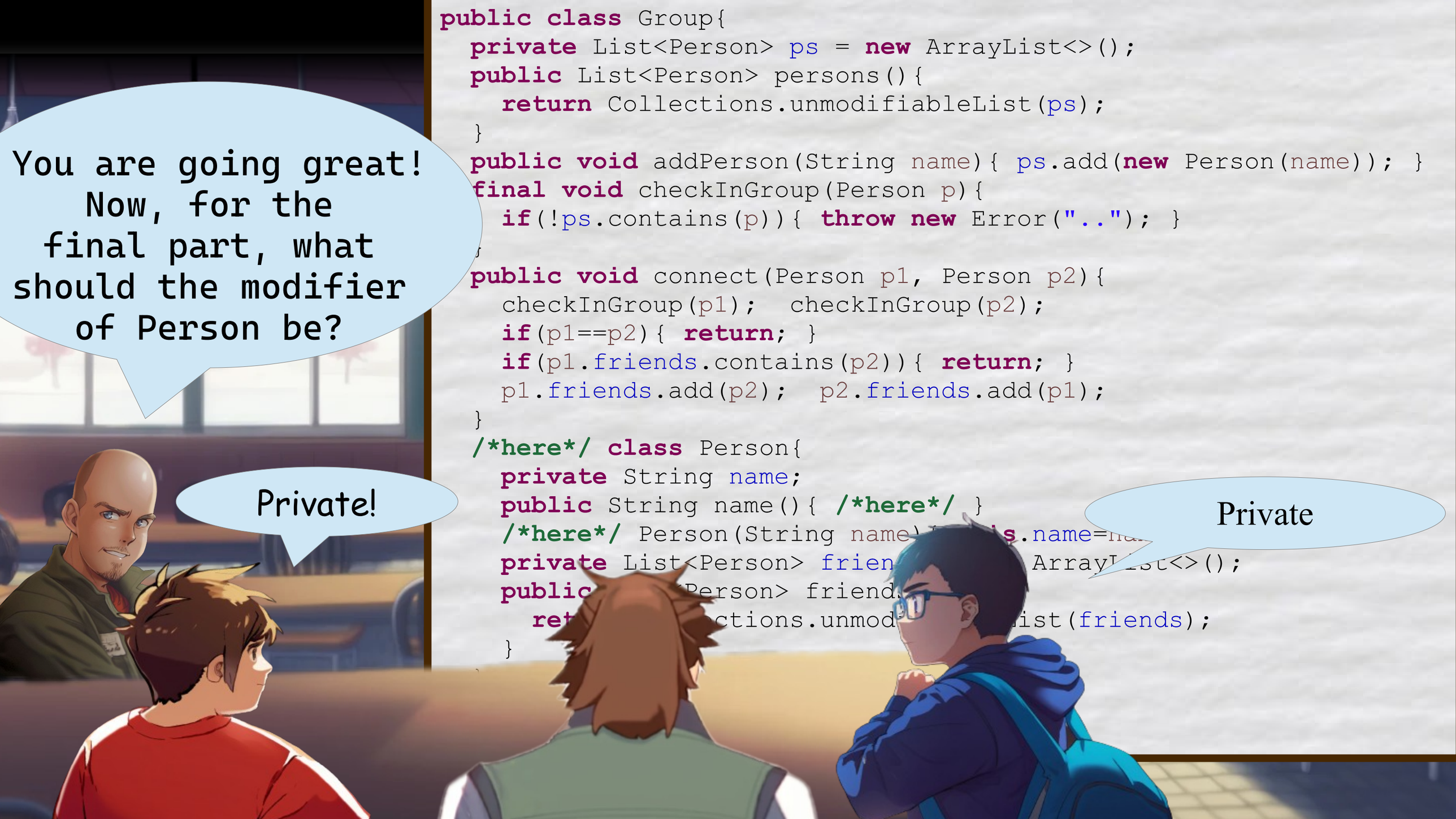
```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```




```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```




```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```

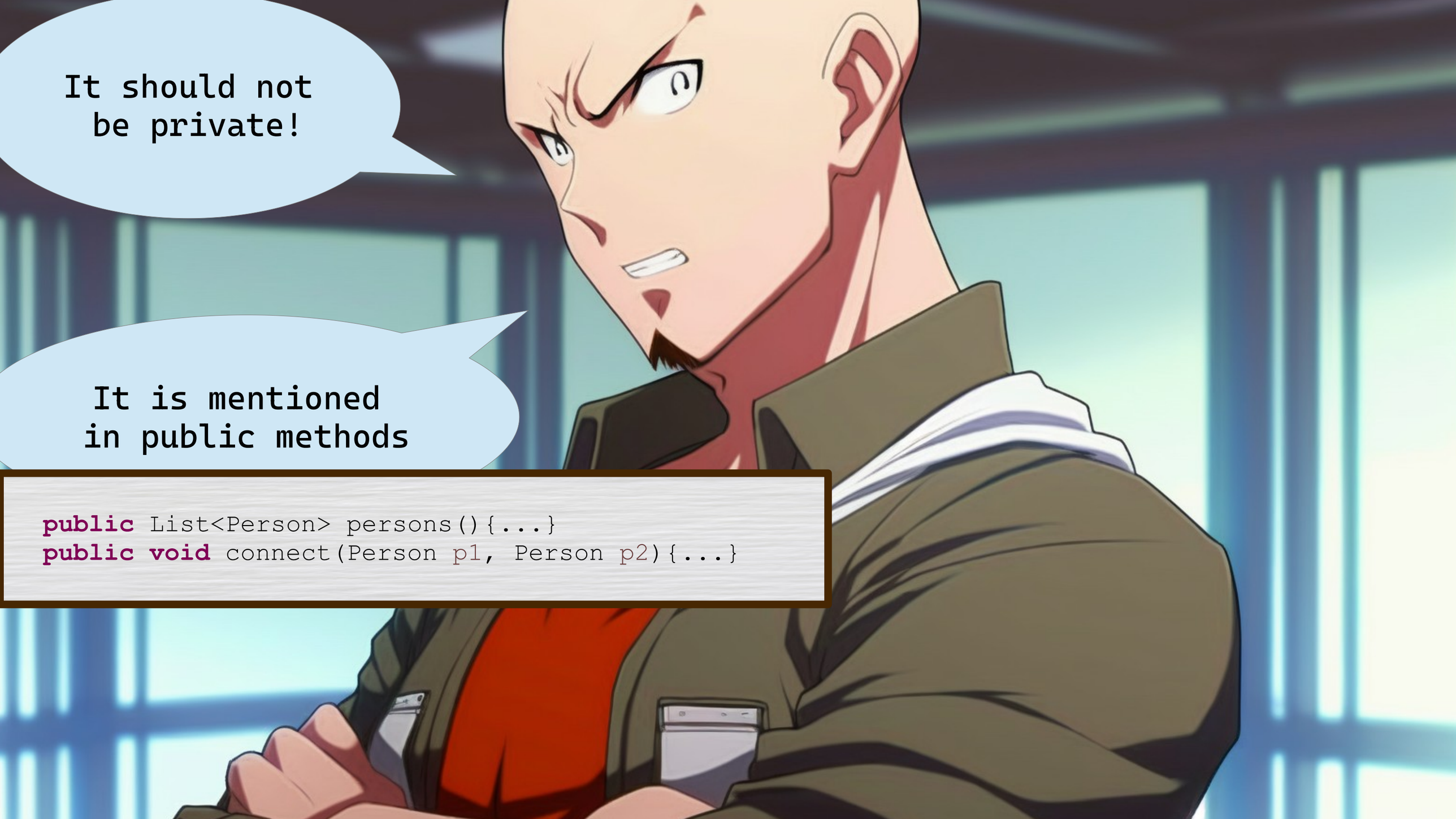


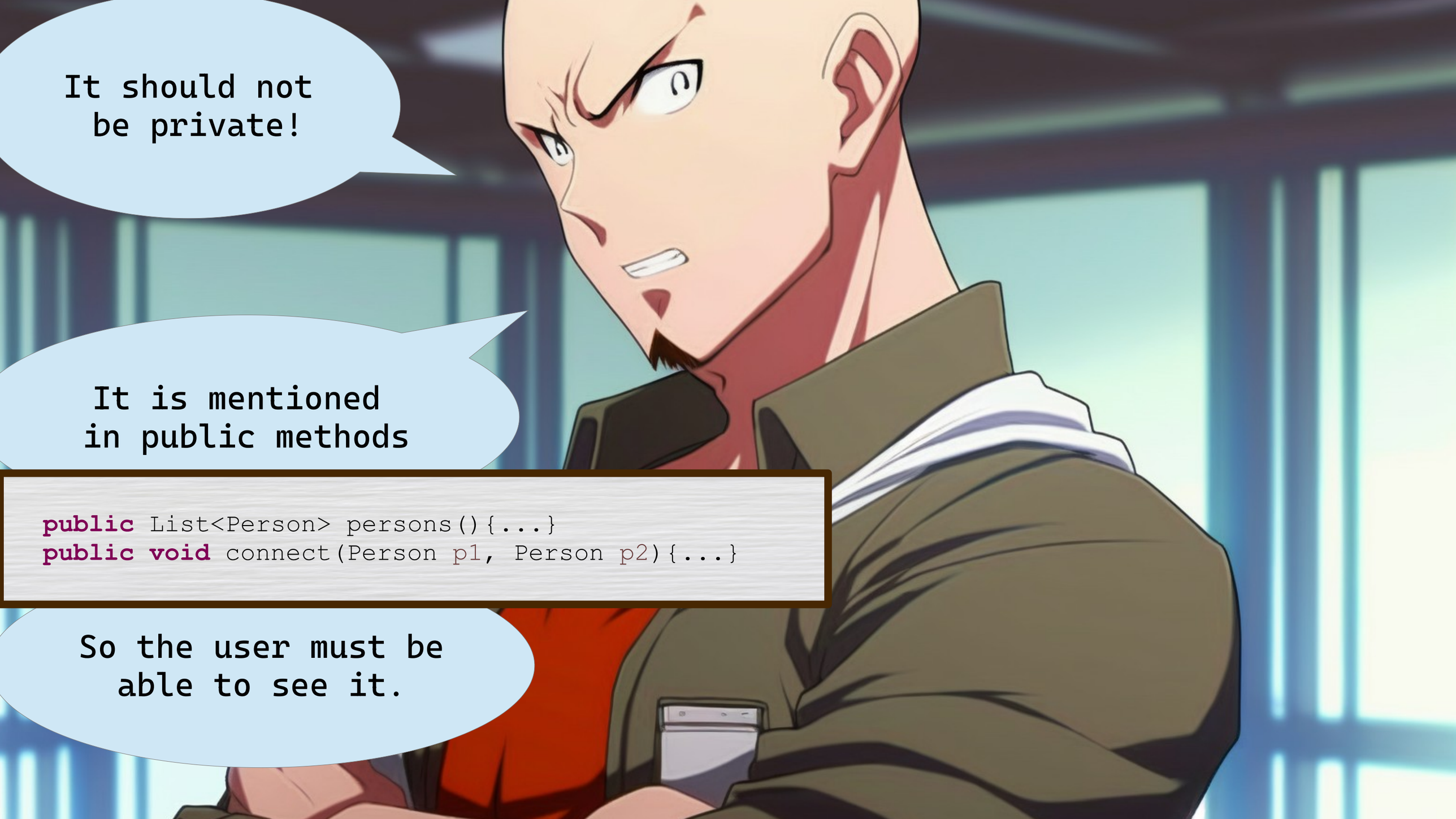

```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```


```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  /*here*/ class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```


It should not be private!


It should not be private!
It is mentioned in public methods


It should not be private!
It is mentioned in public methods
public List<Person> persons(){...}
public void connect(Person p1, Person p2){...}

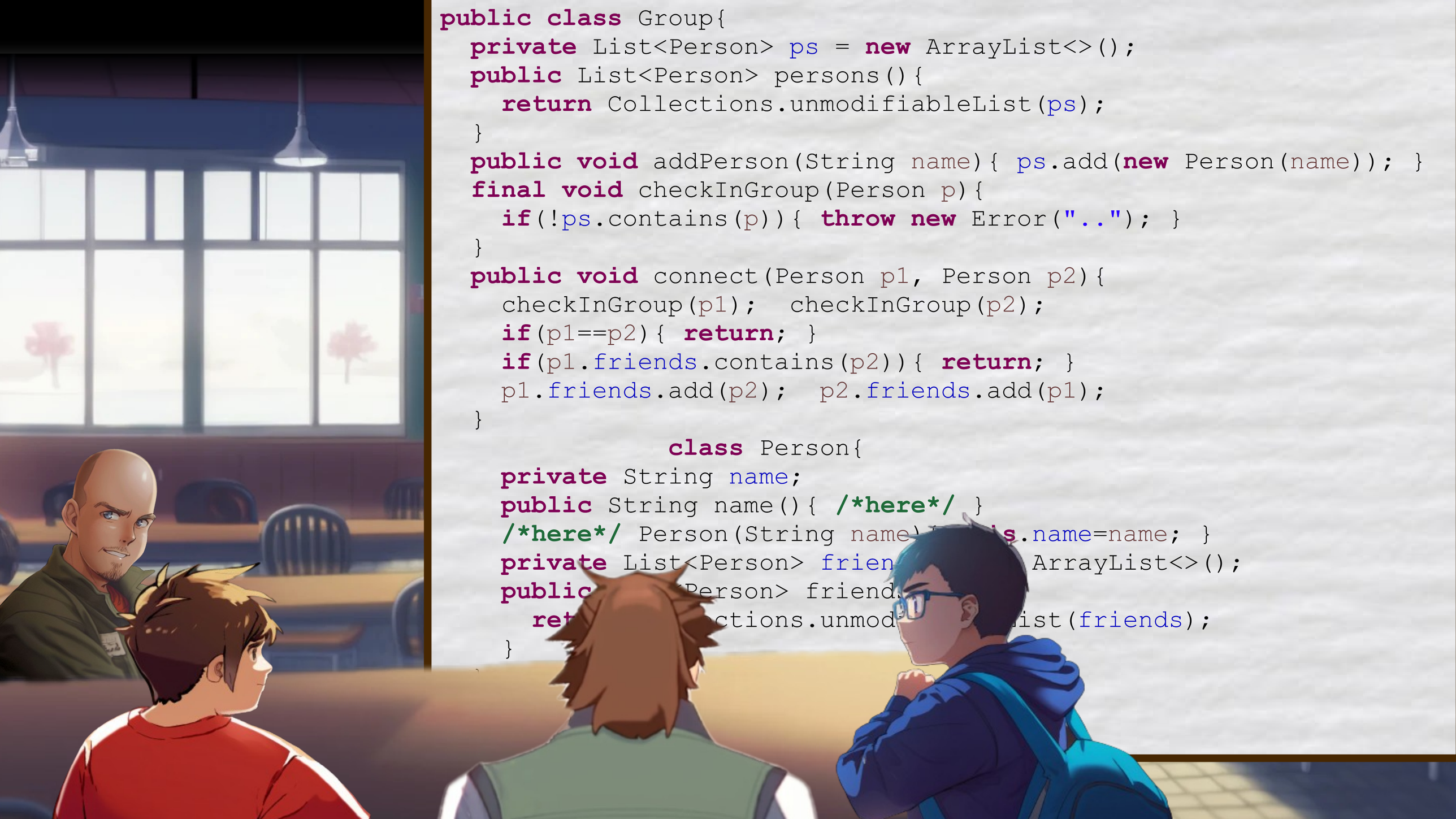

It should not be private!
It is mentioned in public methods
public List<Person> persons(){...}
public void connect(Person p1, Person p2){...}
So the user must be able to see it.


It must be public.

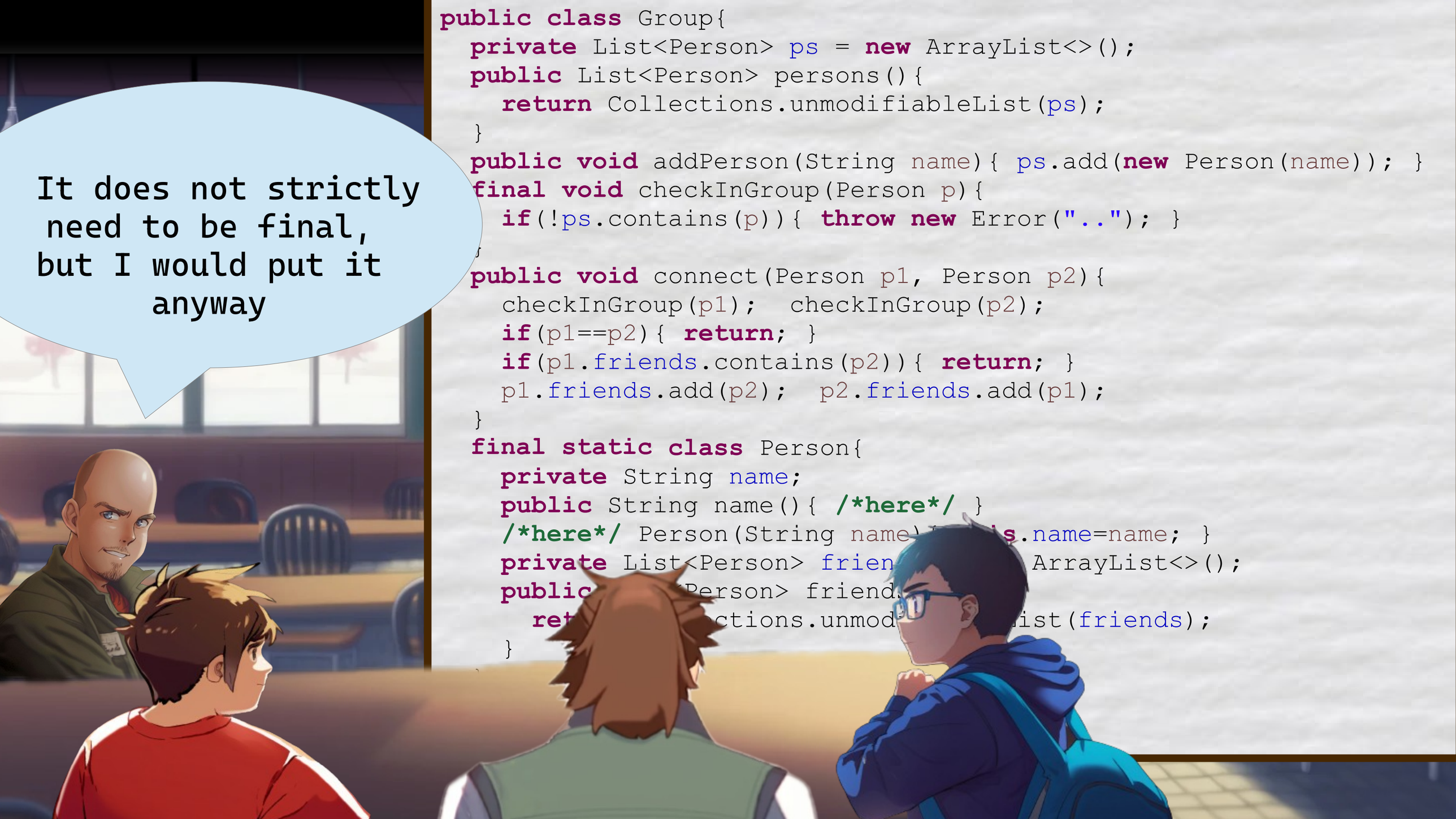

It must be public.
Since Person
is a nested class, …


It must be public.
Since Person
is a nested class, …
we also need to chose
if it is static or not.

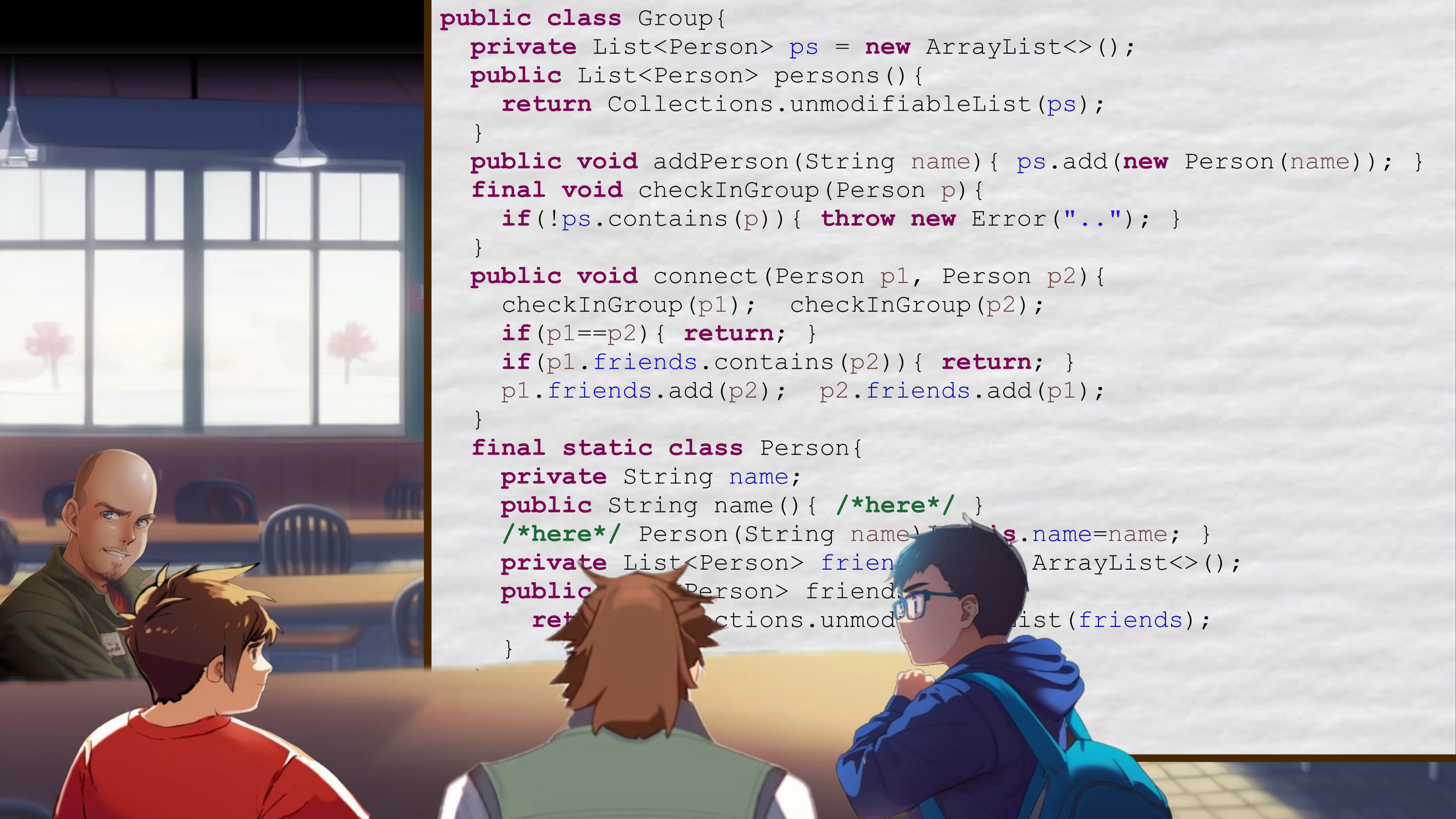

It must be public.
Since Person
is a nested class, …
we also need to chose
if it is static or not.
Since a Person does not
need to know about their Group,
Person should be static.



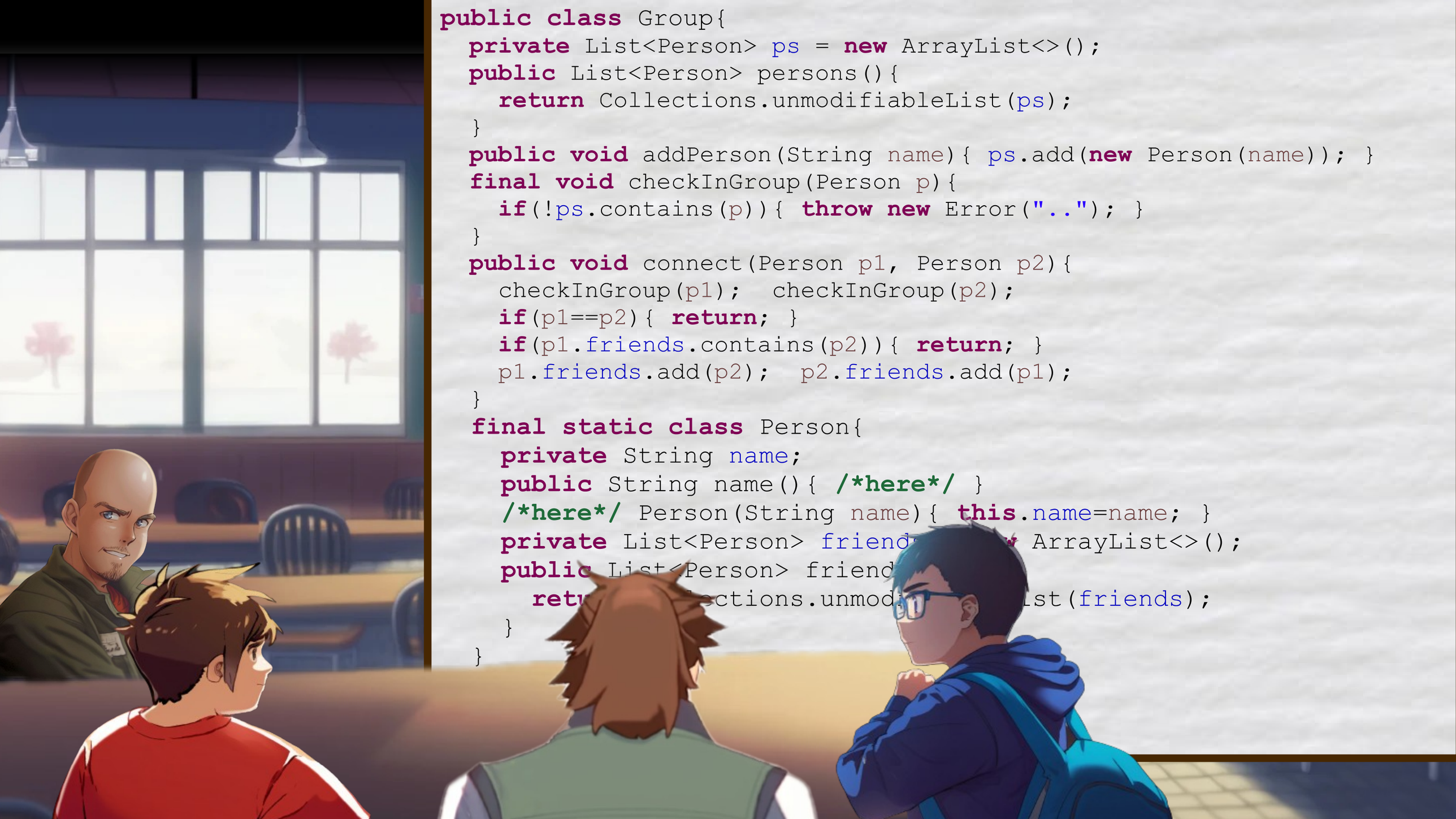
```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
                class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```




```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```



```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodifiableList(friends);
    }
```



```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friend       ArrayList<>();
    public List<Person> friend
      retu      ctions.unmod      ist(friends);
    }
  }
```

```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends
      retu lections.unmod st(friends);
    }
  }
}
```

```java
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
   final static class Person{
     private String name;
     public String name(){ /*here*/ }
     /*here*/ Person(String name){ this.name=name; }
     private List<Person> friends = new ArrayList<>();
     public List<Person> friends(){
       return Collections.unmodifiableList(friends);
     }
   }
}
```

```java
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
   final static class Person{
     private String name;
     public String name(){ /*here*/ }
     /*here*/ Person(String name){ this.name=name; }
     private List<Person> friends = new ArrayList<>();
     public List<Person> friends(){
       return Collections.unmodi[illegible]List(friends);
     }
   }
}
```

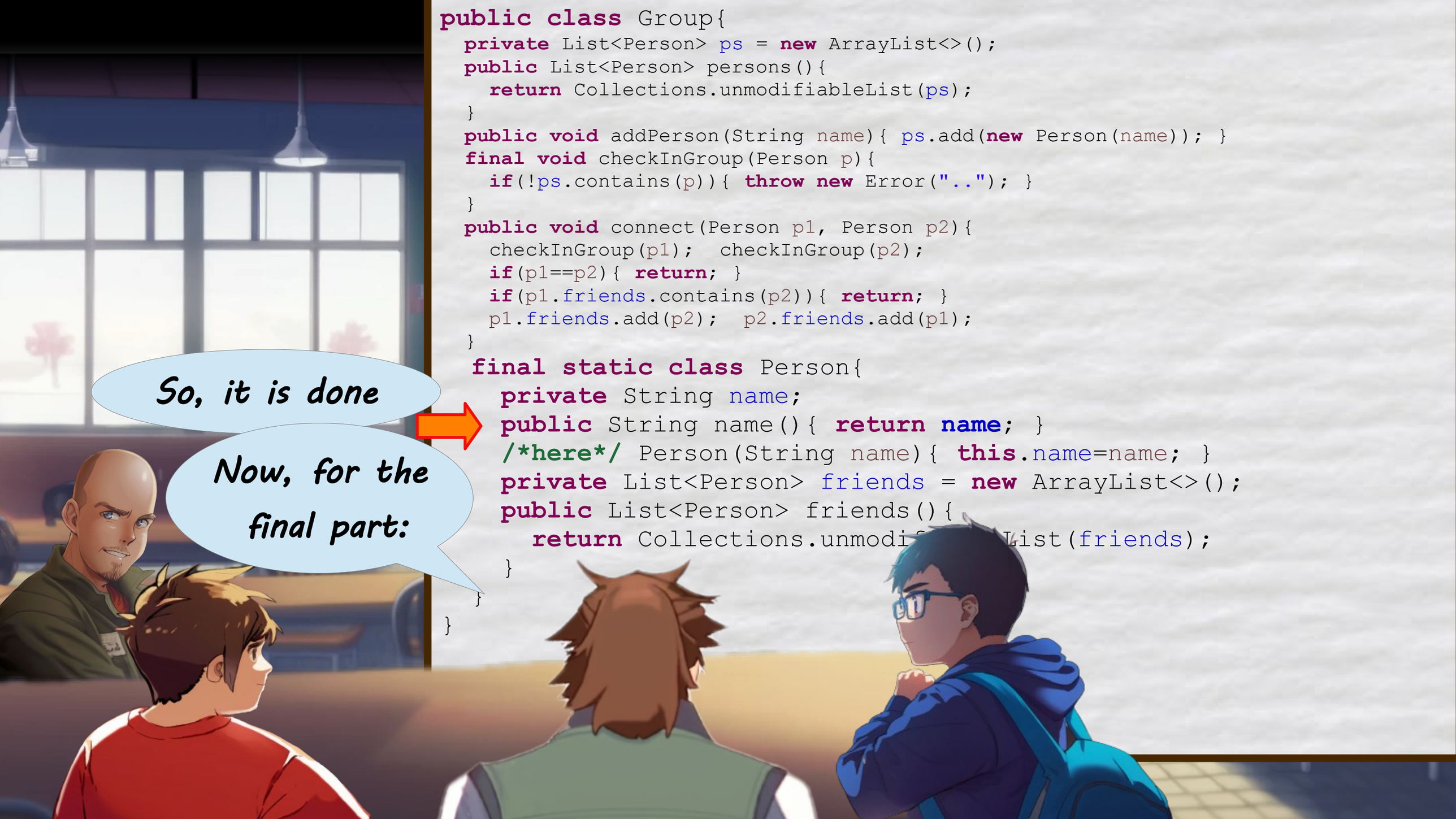

```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    private String name;
    public String name(){ /*here*/ }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodi[illegible]List(friends);
    }
  }
}
```


String are
deeply immutable
in Java


String are
deeply immutable
in Java
They do not need
wrappers

```java
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
   final static class Person{
     private String name;
     public String name(){ return name; }
     /*here*/ Person(String name){ this.name=name; }
     private List<Person> friends = new ArrayList<>();
     public List<Person> friends(){
       return Collections.unmodi[illegible]List(friends);
     }
   }
}
```


```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    private String name;
    public String name(){ return name; }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodi        List(friends);
    }
  }
}
```




```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    private String name;
    public String name(){ return name; }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodi      List(friends);
    }
  }
}
```


```
public class Group{
  private List<Person> ps = new ArrayList<>();
  public List<Person> persons(){
    return Collections.unmodifiableList(ps);
  }
  public void addPerson(String name){ ps.add(new Person(name)); }
  final void checkInGroup(Person p){
    if(!ps.contains(p)){ throw new Error(".."); }
  }
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    private String name;
    public String name(){ return name; }
    /*here*/ Person(String name){ this.name=name; }
    private List<Person> friends = new ArrayList<>();
    public List<Person> friends(){
      return Collections.unmodi[illegible]List(friends);
    }
  }
}
```


private?


private?
No, if it was private,
we could not use it
in Group.addPerson


```
public class Group{
   ...
   public void addPerson(String name){ ps.add(new Person(name)); }
   ...
}
```


private?
No, if it was private, we could not use it in Group.addPerson
public class Group{
...
public void addPerson(String name){ ps.add(new Person(name)); }
...
}
But if it was public, the user could make new Persons,


private?
No, if it was private, we could not use it in Group.addPerson
public class Group{
...
public void addPerson(String name){ ps.add(new Person(name)); }
...
}
But if it was public, the user could make new Persons,
and they would not belong to any group!


Wait a moment!

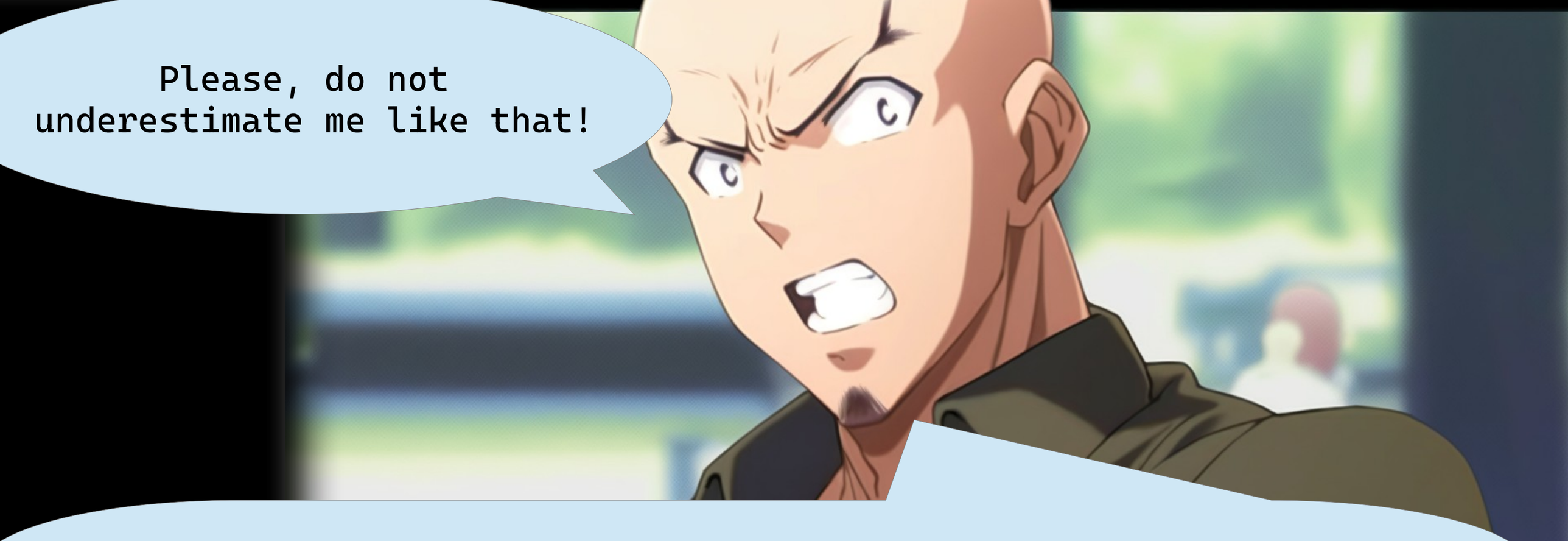
Wait a moment!

We are already using private stuff from Person inside Group.connect


```java
public class Group{
  ...
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    ...
    private List<Person> friends = new ArrayList<>();
  }
```

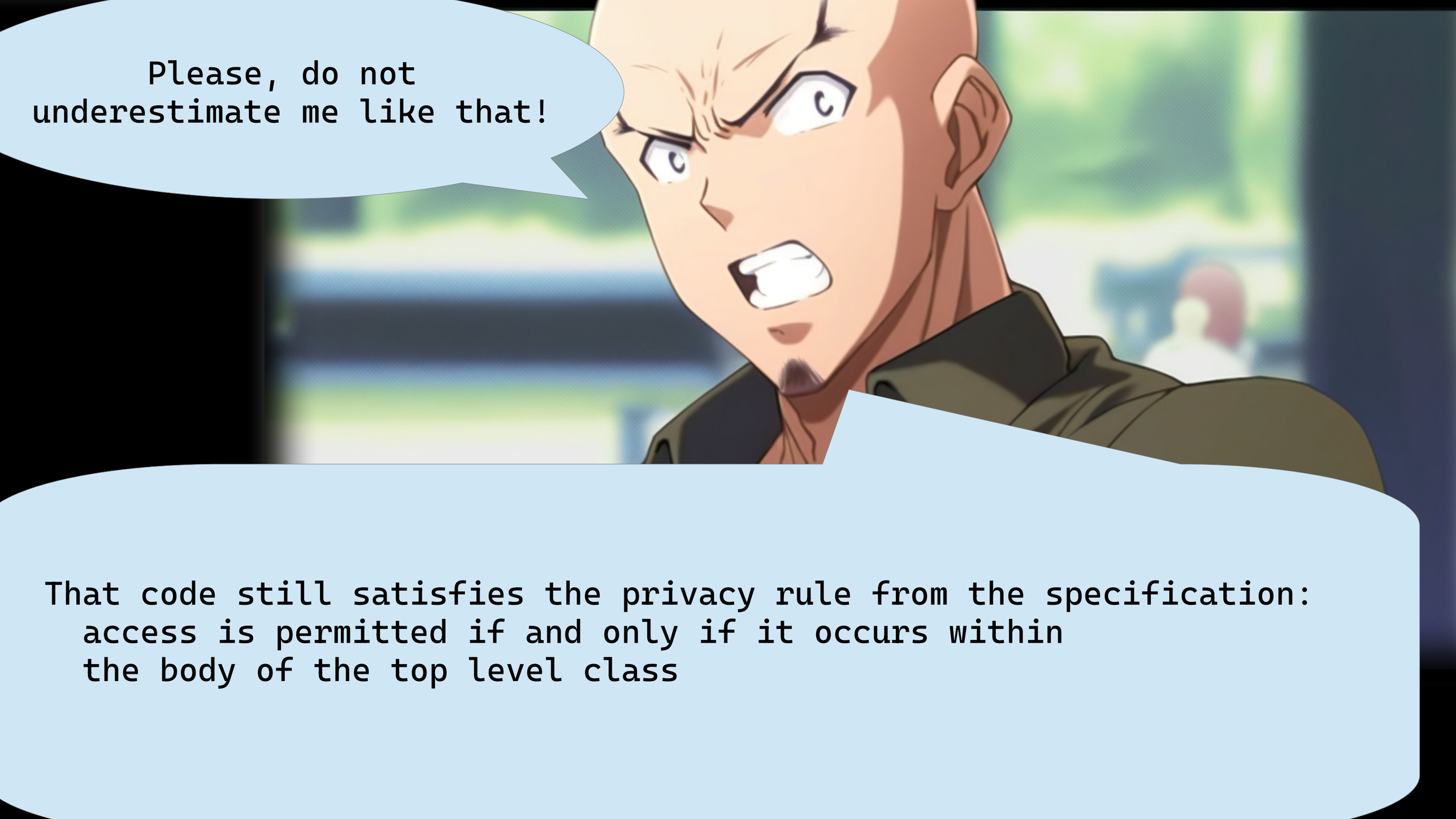

```java
public class Group{
  ...
  public void connect(Person p1, Person p2){
    checkInGroup(p1);  checkInGroup(p2);
    if(p1==p2){ return; }
    if(p1.friends.contains(p2)){ return; }
    p1.friends.add(p2);  p2.friends.add(p1);
  }
  final static class Person{
    ...
    private List<Person> friends = new ArrayList<>();
  }
```


Please, do not
underestimate me like that!

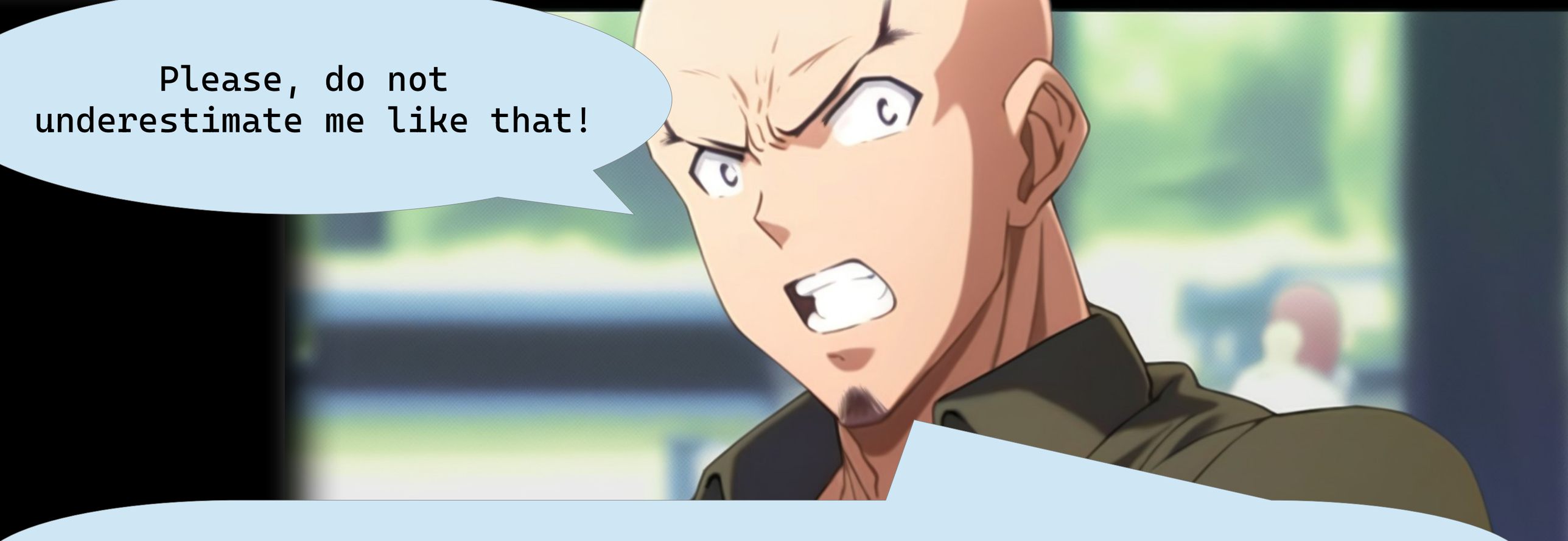


That code still satisfies the privacy rule from the specification:

That code still satisfies the privacy rule from the specification:
access is permitted if and only if it occurs within the body of the



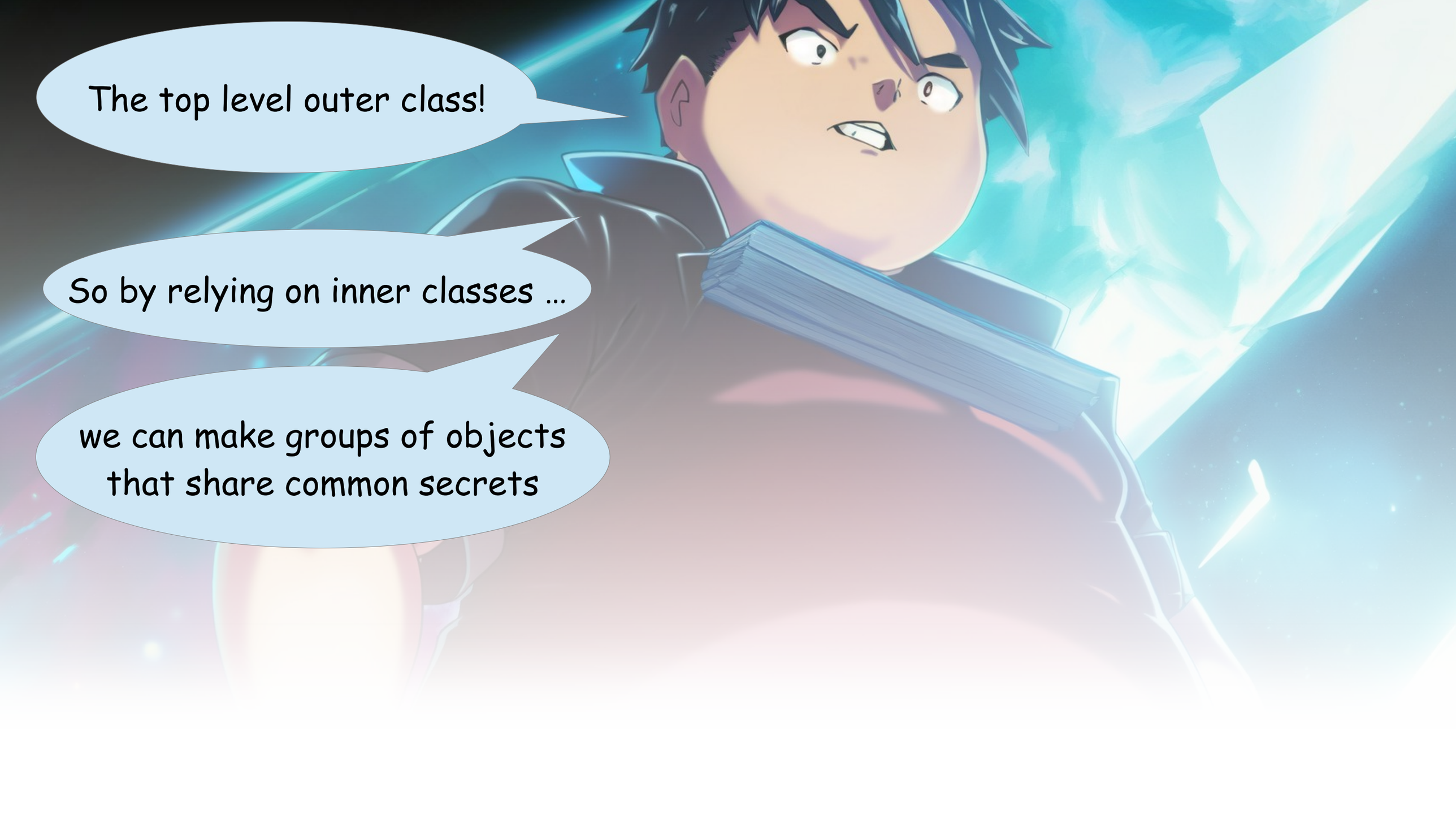
That code still satisfies the privacy rule from the specification:
access is permitted if and only if it occurs within the body of the top level class



That code still satisfies the privacy rule from the specification:
access is permitted if and only if it occurs within
the body of the top level class that encloses
the declaration of the member or constructor


The top level outer class!


The top level outer class!
So by relying on inner classes ...


The top level outer class!
So by relying on inner classes ...
we can make groups of objects that share common secrets


Exactly!
the privacy rule may
seem weak at first...


Exactly!
the privacy rule may
seem weak at first...
but thanks to
this property, …


Exactly!
the privacy rule may
seem weak at first...
but thanks to
this property, …
we can ensure
multi-class invariants
on private state!


This is all for today.
See you in the lecture,
and do not break any
more windows!


Oh, … I feel so much better


Indeed, with his guidance, we will overcome any struggle!
Oh, … I feel so much better


Indeed, with his guidance, we will overcome any struggle!
Oh, … I feel so much better
I wonder how Hanton managed to get this good. He is really impressive.


Indeed, with his guidance, we will overcome any struggle!
Oh, … I feel so much better
I wonder how Hanton managed to get this good. He is really impressive. Has he really memorized all of the Java specification?


Indeed, with his guidance, we will overcome any struggle!
Oh, … I feel so much better
I wonder how Hanton managed to get this good. He is really impressive. Has he really memorized all of the Java specification?
Should we get another pizza?


04:
The tome of knowledge


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