

COMP2511

WEEK 8

What train line do you take?

ADMIN STUFF

- Assignment-i marks out, let me know if any problems
- Assignment-ii due Week 9 Friday 5pm
- Let me know if you would like to do assignment-iii in (different) pairs or individually
- Week 10 will provide a sample exam environment

A G E N D A

- Generic Programming
- Singleton Pattern & Synchronisation

Generics

GENERIC PROGRAMMING



```
public class SortingClass {  
    public List<Integer> sort(List<Integer> unsortedList) {  
        // does sorting here  
        // return sortedList  
    }  
}
```

GENERIC PROGRAMMING



```
public class SortingClass {  
    public List<String> sort(List<String> unsortedList) {  
        // does sorting here  
        // return sortedList  
    }  
}
```

GENERIC PROGRAMMING



```
public class SortingClass {  
    public List<???> sort(List<???> unsortedList) {  
        // does sorting here  
        // return sortedList  
    }  
}
```

GENERIC PROGRAMMING

What are generics?

Generics enable types to be passed when defining classes, interfaces or methods

- Remove casting and offer stronger type checks at compile time
- Allow implementations of generic algorithms, that work on a collection of different types
- Adds stability to code by making more of your bugs detectable at run-time

The *List* class is a perfect example of *Java Generics*.

- *List<Integer>*
- *List<String>*
- *List<List<Double>>*
- *etc...*

GENERIC PROGRAMMING

Inside **src/stack**, there are a series of stubs for a Stack class which takes in a generic type. There are a series of tests inside **StackTest.java** which currently fail.

Implement the methods so that the tests pass, using an ArrayList to store the internal data structure. Answer the following questions:

1. What is E?
2. What is the Iterable interface? Why does it have an E as well? What methods does it force us to implement?
3. When completing toArrayList, why do we need to make a copy rather than just returning our internal ArrayList?
4. What does the .iterator() method allow us to do? Discuss the test inside StackTest.java.

GENERIC PROGRAMMING

- **What is E?**
 - Generic type
- **What is the Iterable interface? Why does it have an E as well? What methods does it force us to implement?**
 - Iterable: Something that can be iterated over
 - Forces us to implement the `.iterator()` method
- **When completing `toArrayList`, why do we need to make a copy rather than just returning our internal `ArrayList`?**
 - Don't want to break encapsulation
- **What does the `.iterator()` method allow us to do? Discuss the test inside `StackTest.java`.**
 - `.iterator()` allows us to loop through it like a normal collection

GENERIC PROGRAMMING



```
public class SortingClass<T> {  
    public List<T> sort(List<T> unsortedList) {  
        // does sorting here  
        // return sortedList  
    }  
}
```

T = type K = key V = value

GENERIC PROGRAMMING

What if I want to modify one method without touching the class?

Wildcards are your answer (kinda)



```
public class SortingClass {  
    public List<?> sort(List<?> unsortedList) {  
        // does sorting here  
        // return sortedList  
    }  
}
```

Note: wildcards can not be used as a type

GENERIC PROGRAMMING

Bounded wildcards

What does `<? extends Type>` and `<? super Type>` mean?

- `extends` : the parameterized type must be a class or subclass of the given type
- `super` : the parameterised type must be a class or super class of the given type



```
public static Integer sumStack(Stack<? extends Integer> stack);
```

GENERIC PROGRAMMING



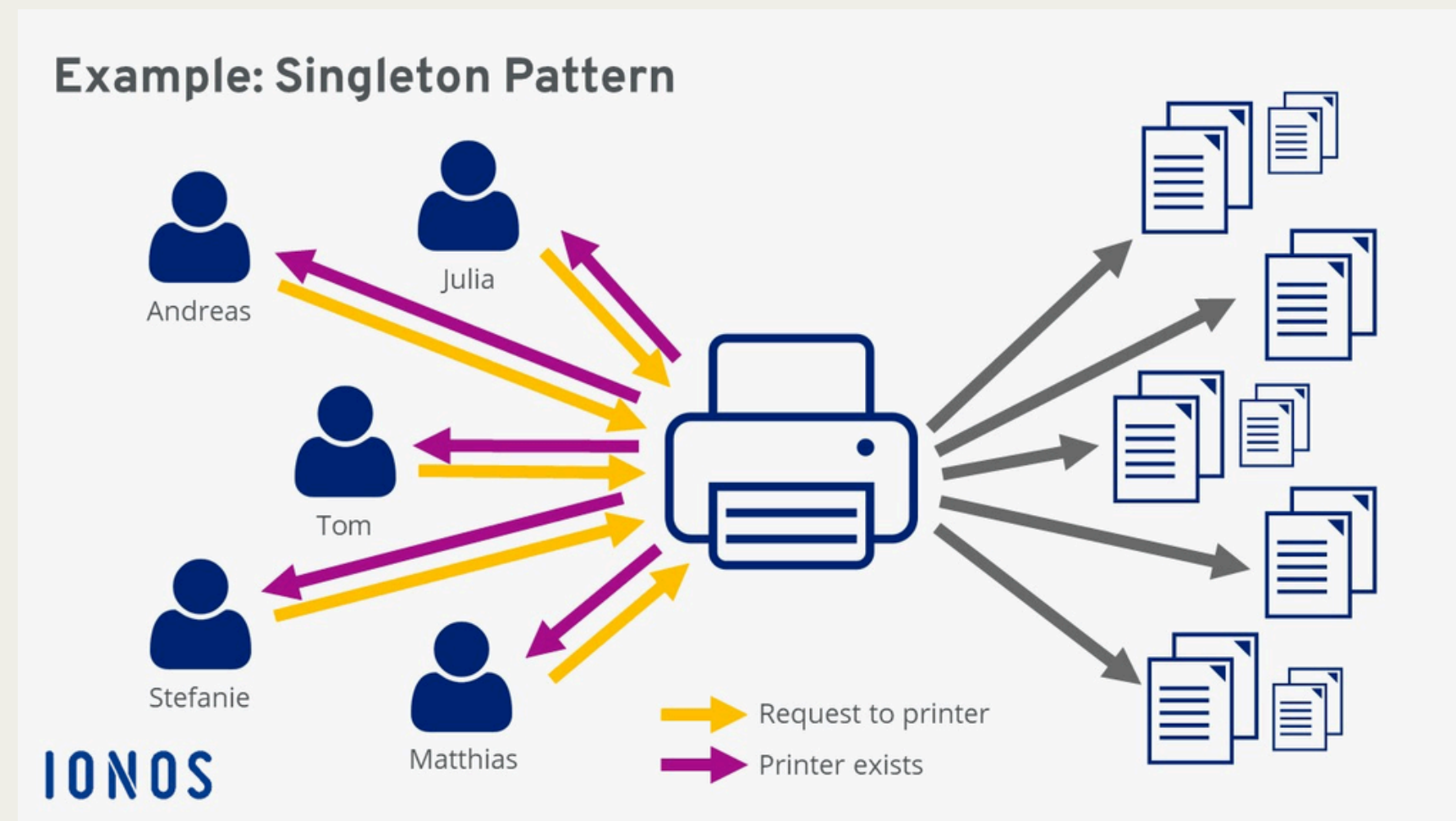
```
public class SortingClass {  
    public <T> List<T> mySort(List<T> unsortedList) {  
        // do sorting stuff  
        // return unsortedList  
    }  
}
```

Singleton Pattern

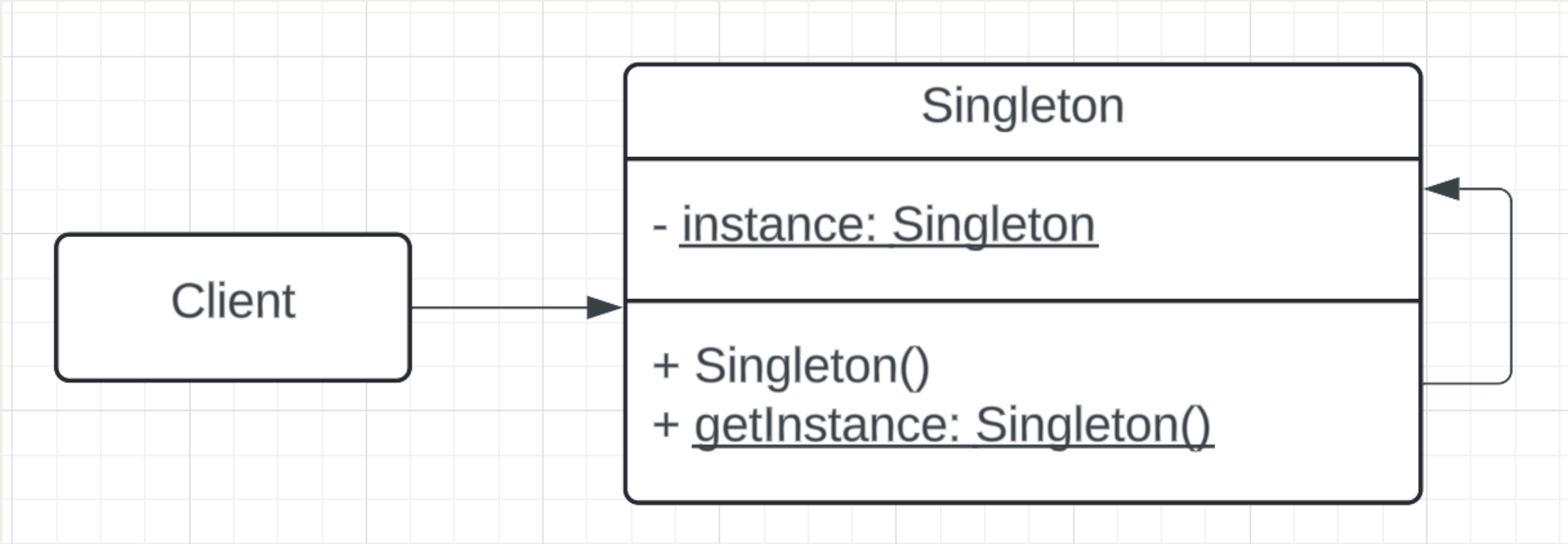
SINGLETON PATTERN

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

It helps avoid initialisation overhead when only 1 copy of an instance is needed.



SINGLETON PATTERN



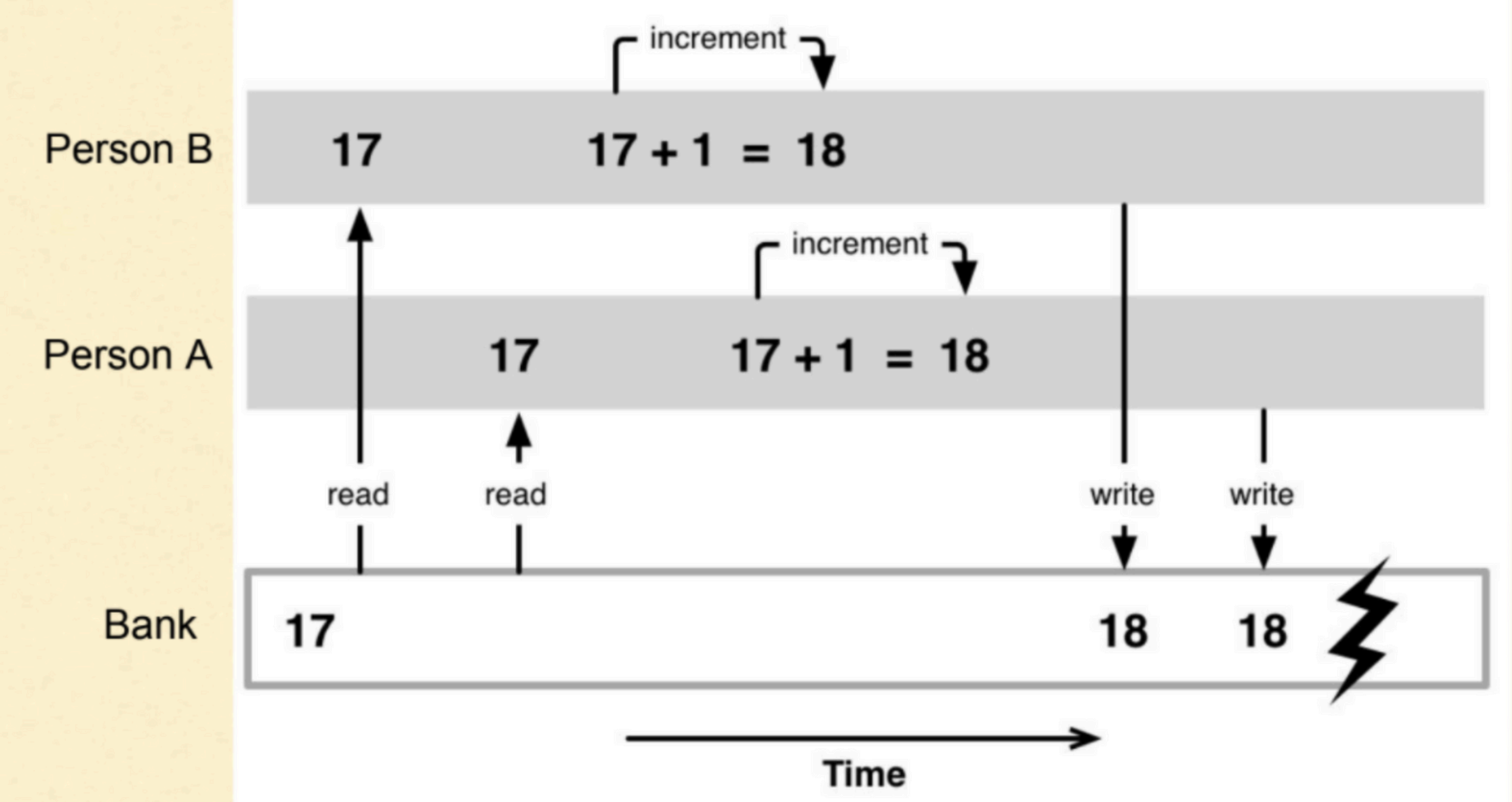
SINGLETON PATTERN

Consider the Bank Account class from Lab 04.

What if multiple people try to access the bank account at the same time? Inside **src/unsww/heist** are three classes:

- BankAccount, from Lab 04.
- BankAccountAccessor. Objects of this type are an instance of an access to a bank account to withdraw money a given number of times by given amounts.
- BankAccountThreadedAccessor, which extends Thread, and overrides the method run to create a new instance of BankAccountAccessor and access the bank.

SINGLETON PATTERN



LABBY YOO

W O O O