

LP2A Written Exam
Friday April 30th, 2024
Duration 1h30
No documents

You must answer on the subject !! Other type of paper will be refused

NOM	Prénom	Signature

Exercise #0 Back to the lectures (4 points)

Question#1: What are the 4 pillars of OOP?

Question#2: What is the purpose of interfaces? How to write a class which implements an interface in java? How many interfaces a class can implement?

Exercise #1 What is appearing on the screen? (8 points)

```
1
2 public abstract class Animal {
3
4     abstract void makeNoise() ;
5
6     void walk() {
7         System.out.println("The animal is walking");
8     }
9
10 }
```

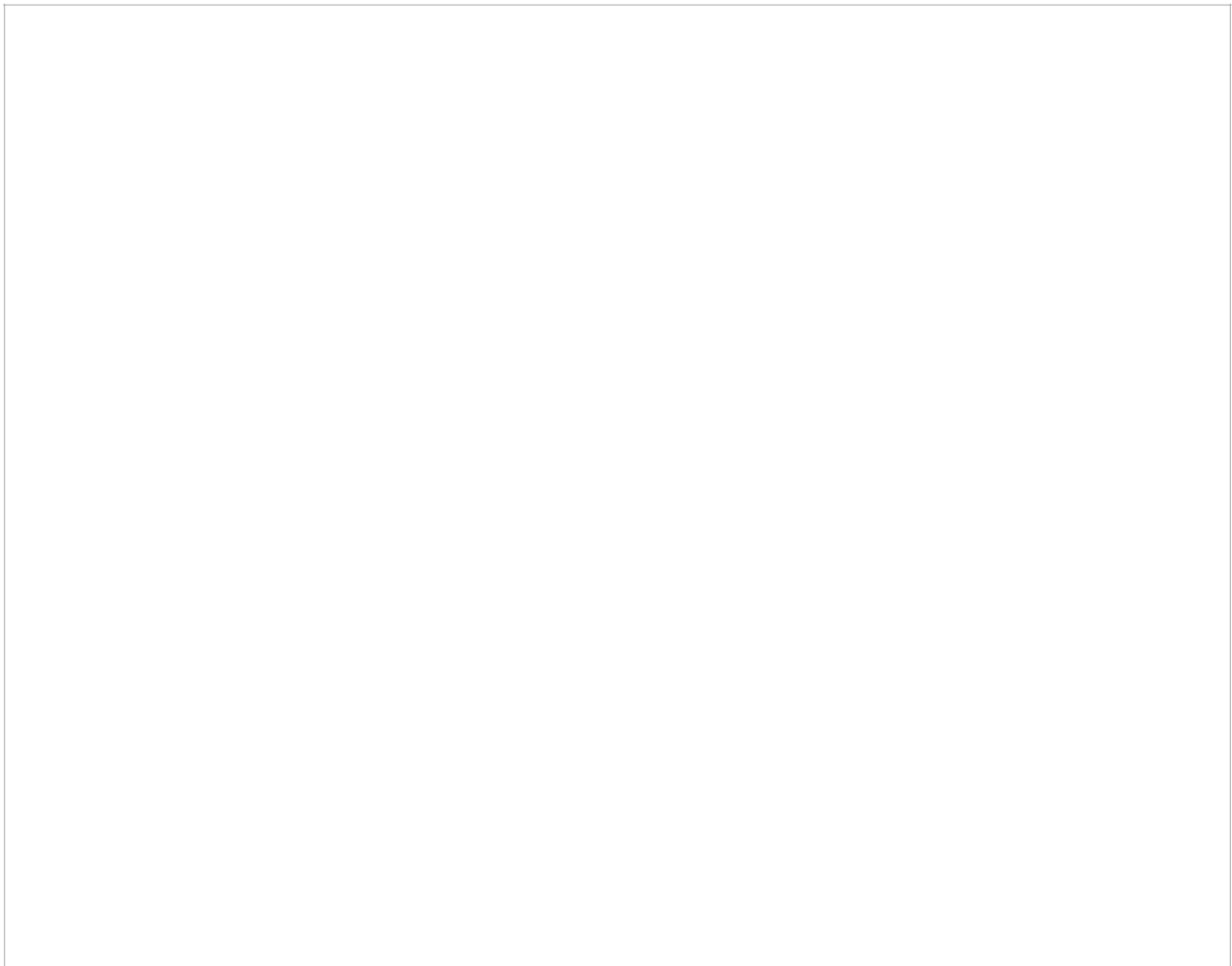
```
1
2 public class Mammal extends Animal{
3
4
5     final void walk() {
6         System.out.println("I'm a moon walking Mammal");
7     }
8
9     void makeNoise() {
10        System.out.println("A mammal sound");
11    }
12 }
```

```
1
2 public class Cat extends Mammal{
3
4     void makeNoise() {
5         System.out.println("Meow");
6     }
7
8 }
```

```
1
2 public class Dog extends Mammal{
3
4     void walk() {
5         System.out.println("I'm a walking dog");
6     }
7 }
```

```
1 import java.util.LinkedList;
2 import java.util.List;
3
4 public class Main {
5
6     public static void main(String[] args) {
7         List<Animal> animals = new LinkedList<>();
8
9         animals.add(new Animal());
10        animals.add(new Dog());
11        animals.add(new Cat());
12        animals.add(new Mammal());
13
14        for (Animal animal : animals) {
15            animal.makeNoise();
16            animal.walk();
17        }
18    }
19 }
20
21 }
```

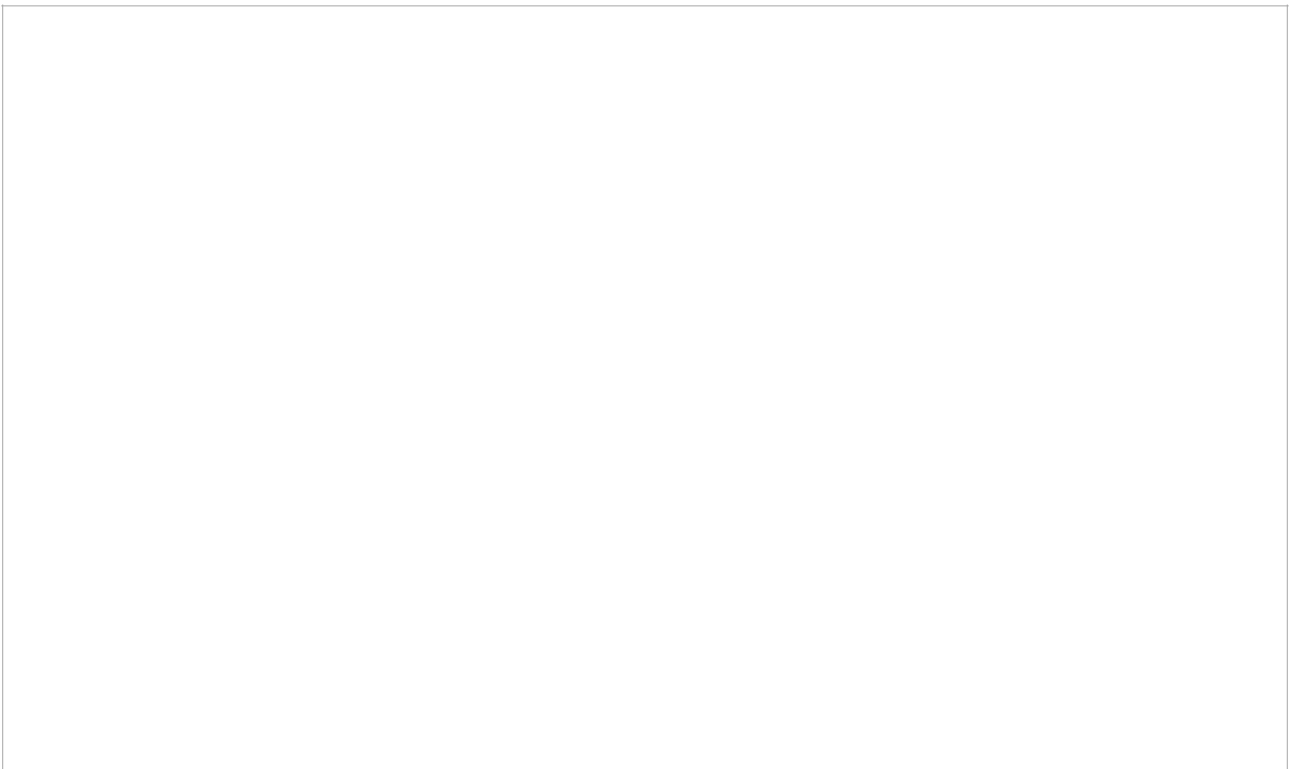
Question #1: Analyse the following code and draw the associated class diagram.



Question #2: There is an error in the code. Can you propose corrections? You can use the name of the class and the line numbers to explain where are the errors.



Question#3: After the correction what this program is printing on the screen?



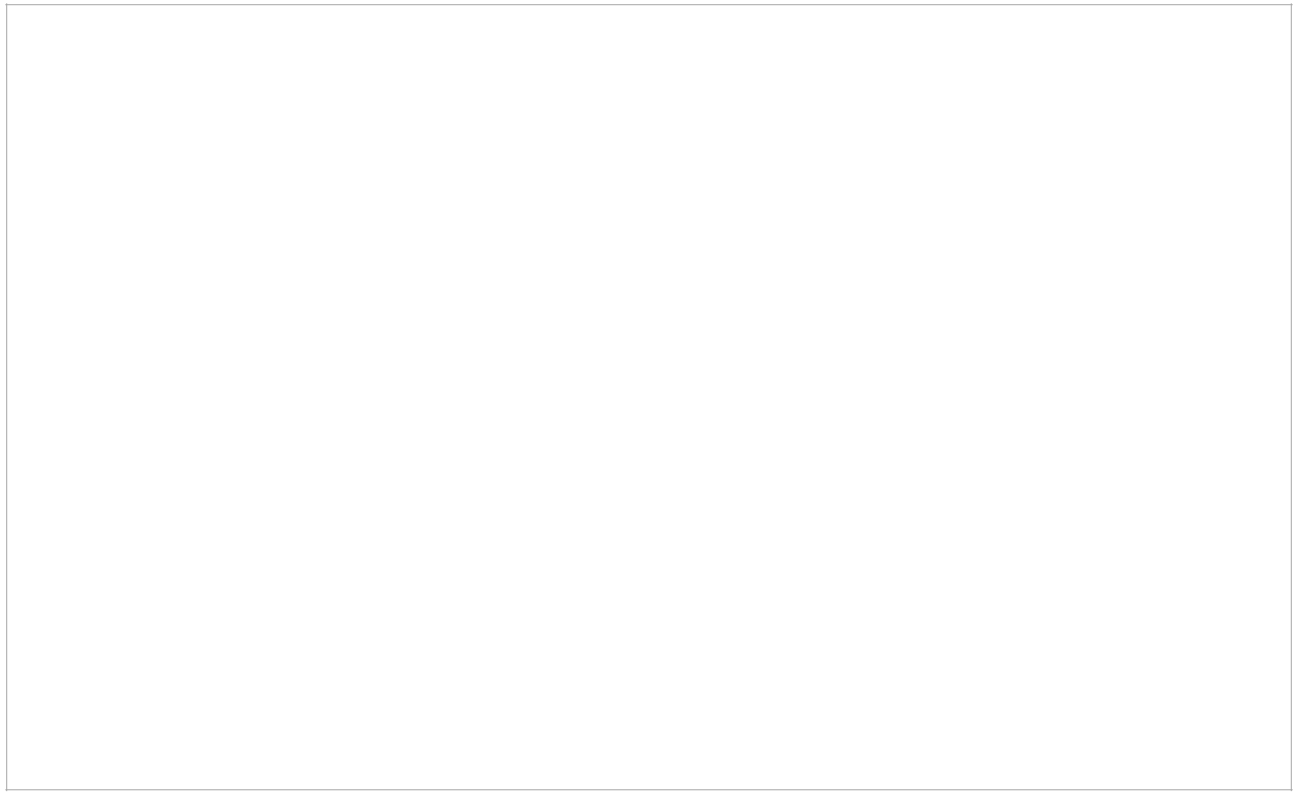
Exercise #2 It reminds me something (8 points)

We want to store students' information into an ordered data structure which allows to optimize the research time. One of the good way to do this is to use the TreeSet (cf. Javadoc description below).

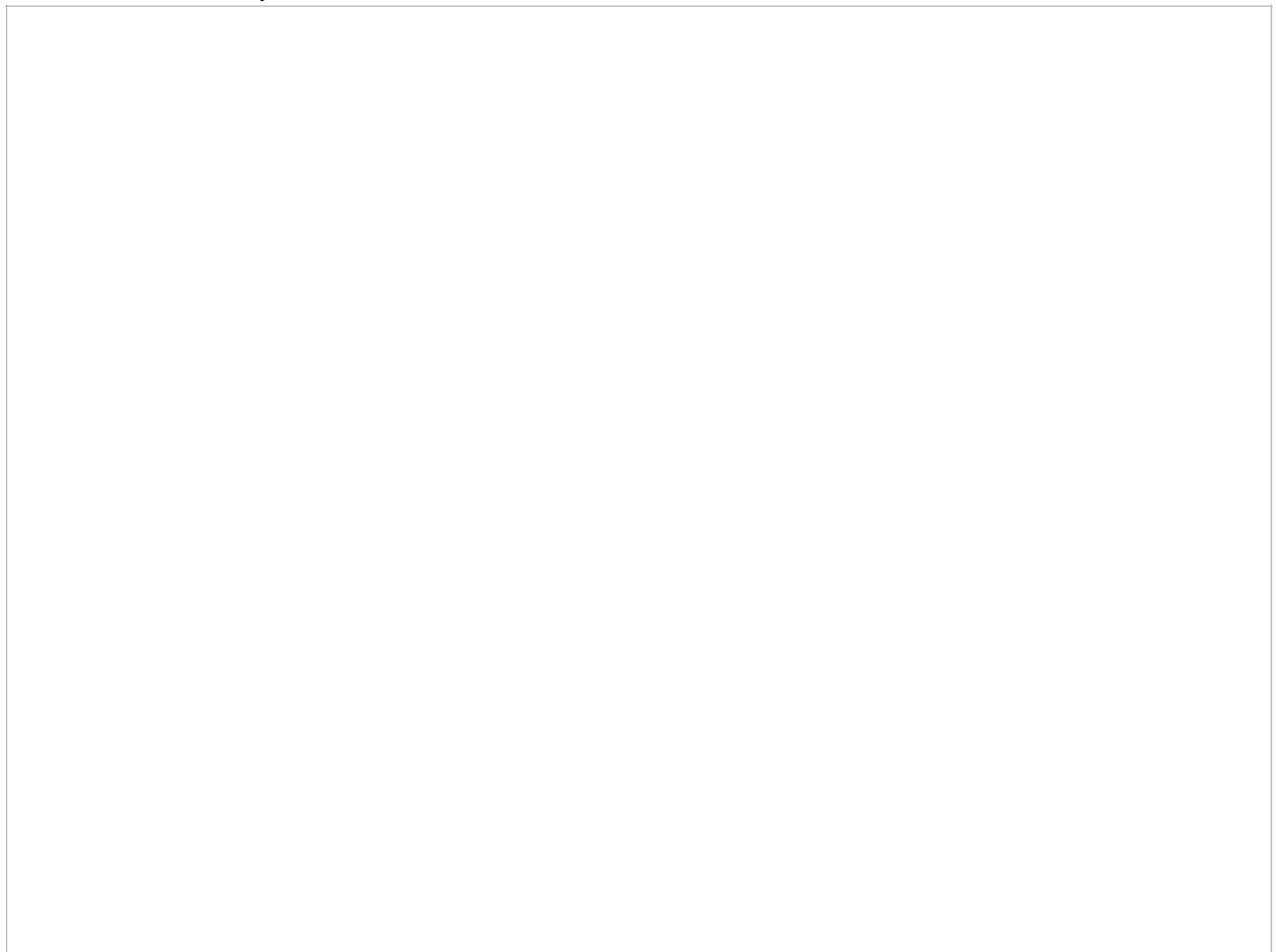
Question #1: Create a Student class with two attributes: name (String) and age (int). Add a constructor to initialize these attributes and a toString method to display the student's information.



Question #2: Create an AgeComparator class that implements the interface: Comparator<Student>. Define the compare method to compare two students based on their age.



Question #3: Create a main class using a `TreeSet<Student>` created by passing an instance of `AgeComparator` to the constructor. Add a few students to the `TreeSet` and print them out.



Question #4: We can avoid to use a comparator if the Student class is implementing the Comparable interface. Explain what modifications, we need to make it work this way.

public interface **Comparable**<T>

This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's *natural ordering*, and the class's `compareTo` method is referred to as its *natural comparison method*.

Lists (and arrays) of objects that implement this interface can be sorted automatically by `Collections.sort` (and `Arrays.sort`). Objects that implement this interface can be used as keys in a [sorted map](#) or as elements in a [sorted set](#), without the need to specify a [comparator](#).

The natural ordering for a class C is said to be *consistent with equals* if and only if `e1.compareTo(e2) == 0` has the same boolean value as `e1.equals(e2)` for every `e1` and `e2` of class C. Note that `null` is not an instance of any class, and `e.compareTo(null)` should throw a `NullPointerException` even though `e.equals(null)` returns `false`.

Method Summary

All Methods

Instance Methods

Abstract Methods

Modifier and Type

Method and Description

int

`compareTo(T o)`

Compares this object with the specified object for order.

```
public class TreeSet<E>
```

Type Parameters:

E - the type of elements maintained by this set

All Implemented Interfaces:

[Serializable](#), [Cloneable](#), [Iterable](#)<E>, [Collection](#)<E>, [NavigableSet](#)<E>, [Set](#)<E>, [SortedSet](#)<E>

The elements are ordered using their [natural ordering](#), or by a [Comparator](#) provided at set creation time, depending on which constructor is used.

[TreeSet](#)([Comparator](#)<? super E> comparator)

Constructs a new, empty tree set, sorted according to the specified comparator.

boolean [add](#)(E e)

Adds the specified element to this set if it is not already present.

E [ceiling](#)(E e)

Returns the least element in this set greater than or equal to the given element, or null if there is no such element.

E [floor](#)(E e)

Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.

E [higher](#)(E e)

Returns the least element in this set strictly greater than the given element, or null if there is no such element.

E [lower](#)(E e)

Returns the greatest element in this set strictly less than the given element, or null if there is no such element.

[SortedSet](#)<E> [subSet](#)(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)

Returns a view of the portion of this set whose elements range from fromElement to toElement.

[SortedSet](#)<E> [subSet](#)(E fromElement, E toElement)

Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.

[SortedSet](#)<E> [tailSet](#)(E fromElement)

Returns a view of the portion of this set whose elements are greater than or equal to fromElement.


```
public interface Comparator<T>
```

A comparison function, which imposes a *total ordering* on some collection of objects. Comparators can be passed to a sort method (such as `Collections.sort` or `Arrays.sort`) to allow precise control over the sort order. Comparators can also be used to control the order of certain data structures (such as `sorted sets` or `sorted maps`), or to provide an ordering for collections of objects that don't have a `natural ordering`.

```
int compare(T o1, T o2)
```

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.