Soft 2 Klausuren:

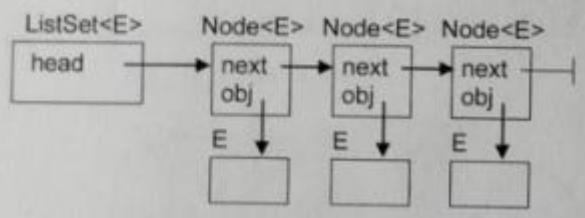
Die Punktezahl einer Aufgabe entspricht den Minuten, die Sie zu ihrer Lösung verwenden sollten. Vergeuden Sie nicht zu viel Zeit mit einer Aufgabe, bei der nur wenige Punkte zu holen sind.

1. Klassen und Interfaces (27 Punkte)

Für das Arbeiten mit Mengen enthält die Java-Bibliothek das Interface *Set<E>*, von dem hier ein vereinfachter Auszug gezeigt wird:

```
interface Set<E> extends Collection<E> {
    boolean add(E e);            // adds e to the set; true if adding was successful
    boolean remove(Object e);    // removes e from the set; true if successful
    ...
}
```

Schreiben Sie eine Klasse *ListSet<E>*, die den oben gezeigten Ausschnitt von *Set<E>* implementiert, und zwar als verkettete Liste ohne Duplikate. Beachten Sie, dass man über den Typparameter *E* nichts weiß, d.h. man kann nicht annehmen, dass *E*-Objekte einen *next*-Zeiger haben. Sie brauchen daher eine Hilfsklasse für die Listenknoten:



Klasse *ListSet<E>* mit innerer Hilfsklasse *Node<E>*

....

Schreiben Sie ein Codestück, in dem ein *ListSet<String>* erzeugt wird und die Strings "Anton" und "Berta" darin eingefügt werden:

....

2. Funktionsinterfaces und Lambda-Ausdrücke (17 Punkte)

Erweitern Sie die Klasse *ListSet<E>* um eine zusätzliche Methode

```
public ListSet<E> filter (Predicate<E> matches) {...}
```

die eine Menge jener Elemente aus dem *ListSet* liefert, die das Prädikat *matches* erfüllen. Deklarieren Sie dazu zunächst das Funktionsinterface *Predicate<E>*:

....

Implementieren Sie nun die oben beschriebene Methode *filter*() der Klasse *ListSet<E>*

...

Ergänzen Sie das folgende Codestück um einen Lambda-Ausdruck, der aus *set* die Menge jener Strings herausfiltert, deren Länge > 5 ist.

```
ListSet<String> set; // Annahme: bereits erzeugt
                     // und mit Strings gefüllt
ListSet<String> set2 =
  set.filter((String s) -> s.size() > 5          );
```

5 Punkte

3

...

3. Entwurfsmuster (15 Punkte)

Beschreiben Sie textuell und mittels Klassendiagramm das Entwurfsmuster *Adapter*.

- Für welchen Zweck wird das *Adapter*-Muster verwendet?
- Wie unterscheidet sich ein *Adapter* von einem *Dekorator*?

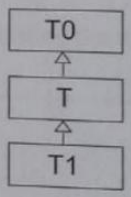
....

4. Testen und Korrektheit (6 + 8 = 14 Punkte)

a) Erklären Sie den Unterschied zwischen Black-Box- und White-Box-Testen. Was sind die Ziele der beiden Testmethoden?

...

b) Gegeben sei nebenstehende Klassenhierarchie (*T0*, *T*, *T1*) sowie folgende beide Klassen *A* und *B*:



8 Punkte

6

```
class A {
   T exec (T x) {...}
}
class B extends A {
   ... exec (... x) {...}  // überschreibt exec aus A
}
```

Override! not Overload

overload wenn nicht selber Parameter

Der Rückgabetyp von *exec()* darf in Java in der Unterklasse *B* wie folgt sein (kreuzen Sie an):

☐ T0 ☒ T ☐ T1

Der Typ des Eingangsparameters *x* von *exec()* darf in Java in der Unterklasse *B* wie folgt sein:

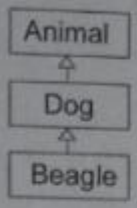
☐ T0 ☒ T ☐ T1

.....

5. Fragen (12 + 5 = 17 Punkte)

a) Gegeben sei eine generische Klasse *List<T>* sowie die Klassen *Animal*, *Dog* und *Beagle* in der dargestellten Vererbungshierarchie:

```
class List<T> {
    void  add (T x) {...}
    T     get () {...}
    int   size () {...}
}
```



Nehmen Sie ferner an, dass eine Variable *list* wie folgt deklariert ist:

```
List<? extends Dog> list;
```

Welche der folgenden Zuweisungen sind erlaubt (Mehrfachnennung möglich; falsche Kreuze bringen aber Punkteabzug)?

- ☐ list = new List<Animal>();
- ☒ list = new List<Dog>();
- ☒ list = new List<Beagle>();

6

Welche Methoden darf man für die oben deklarierte Variable *list* aufrufen (Mehrfachnennung möglich; falsche Kreuze bringen aber Punkteabzug)?

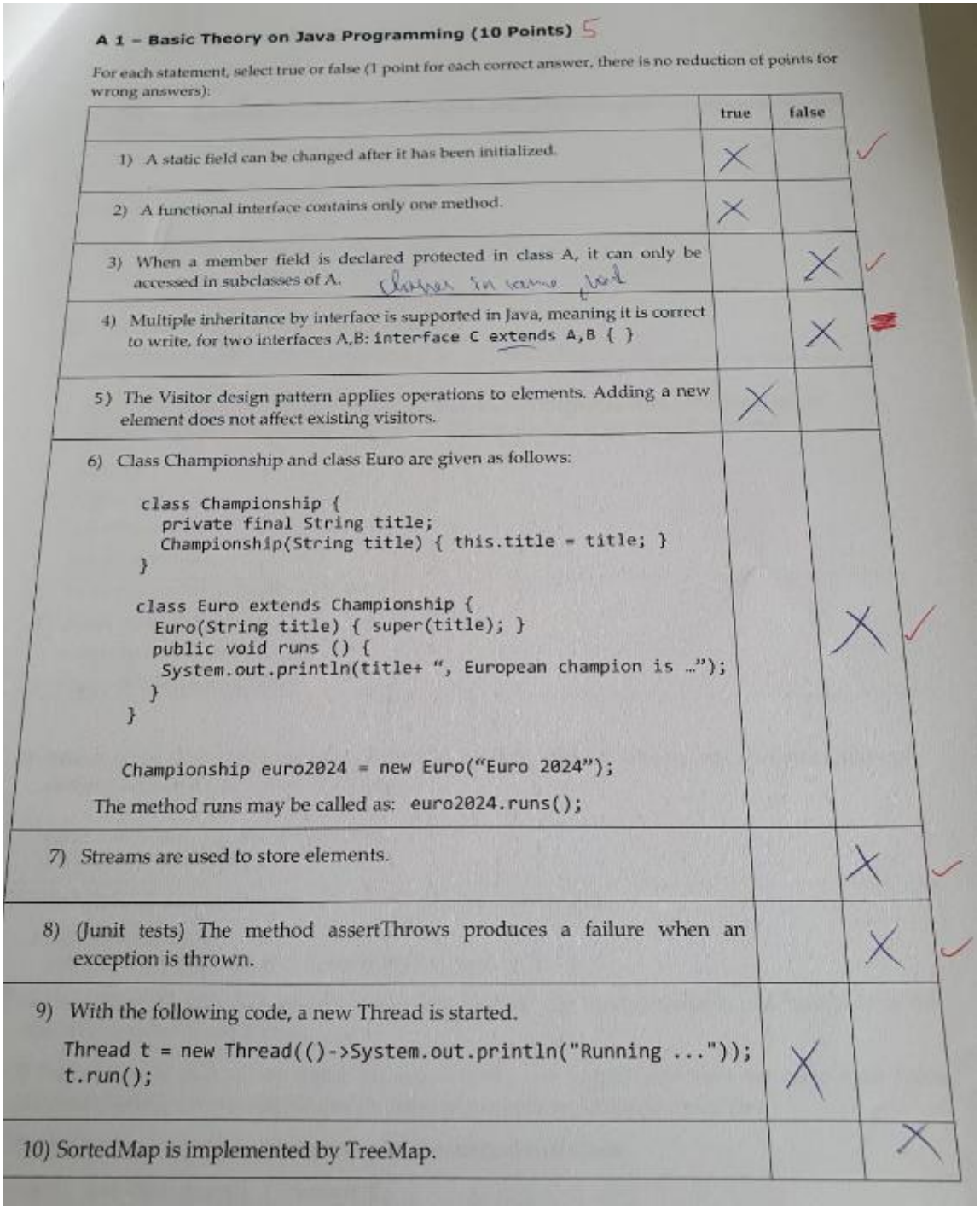
- ☒ list.add(new Dog());
- ☒ list.add(new Beagle());
- ☒ Dog dog = list.get();
- ☐ Beagle beagle = list.get();
- ☒ int size = list.size();

2

b) In welchen Programmteilen darf auf eine Methode zugegriffen werden, die mit dem Sichtbarkeitsattribut *protected* versehen ist?

....

Soft 2 Altklausur nr 2:



A 1 – Basic Theory on Java Programming (10 Points) 5

For each statement, select true or false (1 point for each correct answer, there is no reduction of points for wrong answers):

	true	false
1) A static field can be changed after it has been initialized.	X	
2) A functional interface contains only one method.	X	
3) When a member field is declared protected in class A, it can only be accessed in subclasses of A. (classes in same pkg)		X
4) Multiple inheritance by interface is supported in Java, meaning it is correct to write, for two interfaces A,B: `interface C extends A,B { }`		X
5) The Visitor design pattern applies operations to elements. Adding a new element does not affect existing visitors.	X	

6) Class Championship and class Euro are given as follows:
`class Championship { private final String title; Championship(String title) { this.title = title; } }`
`class Euro extends Championship { Euro(String title) { super(title); } public void runs () { System.out.println(title+ ", European champion is …"); } }`
`Championship euro2024 = new Euro("Euro 2024");`
The method runs may be called as: `euro2024.runs();`
		X

7) Streams are used to store elements.		X
8) (Junit tests) The method assertThrows produces a failure when an exception is thrown.		X

9) With the following code, a new Thread is started.
`Thread t = new Thread(()->System.out.println("Running ..."));`
`t.run();`
	X	

10) SortedMap is implemented by TreeMap.		X

....

A 2 – Applying Knowledge (20 Points)

Mark the correct answers. Note that for each wrong answer, there is a reduction of points. Multiple answers are possible.

1) **The method metaFunc is given as below. Which of the calls of the method metaFunc are correct? (4 Points)**

```
import java.util.function.Function;

void metaFunc (String s, Function<String, Integer> func) {
   System.out.println(func.apply(s)); }
```

- ☐ `metaFunc("The Hitchhiker's Guide to the Galaxy", s->s.length()>0);`
- ☐ `metaFunc("The Hitchhiker's Guide to the Galaxy", s->s.length());`
- ☐ `metaFunc("The Hitchhiker's Guide to the Galaxy", s);`
- ☒ `Function<String, Integer> func = s->42;`
  `metaFunc("The Hitchhiker's Guide to the Galaxy", func);`

2) **Find below example classes and interfaces. Which of the following uses of inheritance and interface implementations with classes A, B and interfaces C, D are correct? (4 Points)**

```
class A
class B<T>
interface C<R>
interface D<S>
```

- ☐ `interface E<S> extends C<R>,D<S>`
- ☒ `class F<T> extends A`
- ☐ `class G<T> extends A, B<T>`
- ☒ `class H implements D<S>`

3) **Which of the following statements about the methods equals and hashCode of class Object (see declaration below) are correct? (4 Points)**

```
public boolean equals(Object obj)
public int hashCode()
```

- ☐ The following code is a correct way of overriding method equals:

```
class SomeObject {
  public boolean equals (SmartObject obj) { } }
```

- ☒ If two objects o1 and o2 should be added to a HashSet, the methods equals and hashCode of the HashSet need to be overridden.
- ☐ If two objects o1 and o2 are equal (`o1.equals(o2) == true`) and have the same hash value (`o1.hashCode() == o2.hashCode()`), they cannot both be added to a HashSet.
- ☐ The following is a good implementation of the method hashCode:

```
public int hashCode() { return 1; }
```

....

4) **The following interface, classes, object are given. Which of the following statements are true for the Adapter pattern? (4 Points)**

```
public interface Robot { public void move(); }
public class Humanoid implements Robot { public void move() { } }
public class Machine { public void doJob() { } }
Machine assembler = new Machine ();
```

- ☐ Method move() in class Humanoid is a case of overriding.
- ☒ A correct implementation of an Adapter making Machine compatible to Robot is provided by:

```
public class MachineAdapter implements Robot{
   Machine m;
   MachineAdapter(Machine m) { this.m = m; }
   public void move() { m.doJob(); }}
```

- ☐ A correct implementation of an Adapter making Machine compatible to Robot is provided by:

```
public class MachineAdapter implements Robot{
   MachineAdapter() { }
   public void move() { Machine m = new Machine(); m.doJob(); }}
```

- ☐ A correct use of the Machine instance assembler is: `assembler.move();`

5) The following code is given. Which of the statements are true? (4 Points)

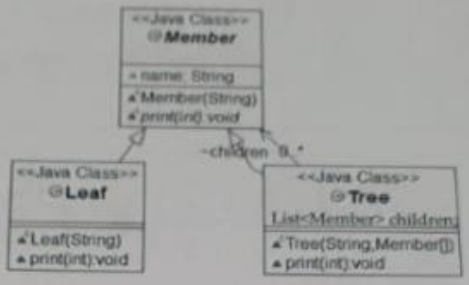
```
static class Resource { }

public static void main(String[] args) {
   Resource r1 = new Resource(); Resource r2 = new Resource();

   Thread t1 = new Thread() {
      @Override
      public void run() {
         synchronized (r1) {
            System.out.println ("T1 lock r1");
            try{ Thread.sleep(1000);}catch (InterruptedException e){ e.printStackTrace(); }
            synchronized (r2) { System.out.println ("T1 lock r2");}}}};

   Thread t2 = new Thread() {
      @Override
      public void run() {
         synchronized (r2) {
         System.out.println ("T2 lock r2");
         try { Thread.sleep(1000);} catch (InterruptedException e) { e.printStackTrace(); }
         synchronized (r1) { System.out.println ("T2 lock r1");}}}};

   // HERE - use the threads t1 and t2
}
```

- ☒ The implementation of run() in threads t1 and t2 assures that only one thread may get a lock on a Resource object at the same time.
- ☐ When executing the following (at position // HERE), a deadlock may happen: `t1.run(); t2.run();`
- ☐ After starting threads t1 and t2 (at position // HERE), the main thread may wait for t1 and t2 to finish with: `t1.join(); t2.join();`
- ☐ Using the keyword `synchronized` assures that when starting first t1 and then t2, always t1 enters the run() method first.

....

A3 – Inheritance and Composition (15 Points)

The task is to implement a family tree. Each Member of a family tree may be either a Leaf (person without children) or a Tree (person with children, who may be either a Leaf or a Tree). The printout of the family tree should result in the name of the family member and reflect the hierarchy of the ancestors by adding a dash ("-") for each hierarchy level. The following UML diagram visualizes the class structure consisting of the abstract class Member and the two classes Leaf and Tree, which are sub-classes of Member:



Example use of Member for a part of the British royal family tree:

```
Member windsors =
      new Tree("Queen Victoria",
            new Tree("King Edward VII",
                  new Tree("King George V",
                        new Tree("King George VI",
                              new Tree("Queen Elizabeth II",
                                    new Tree("King Charles III",
                                          new Leaf("William"),
                                          new Leaf("Harry")))))));
windsors.print(0);
```

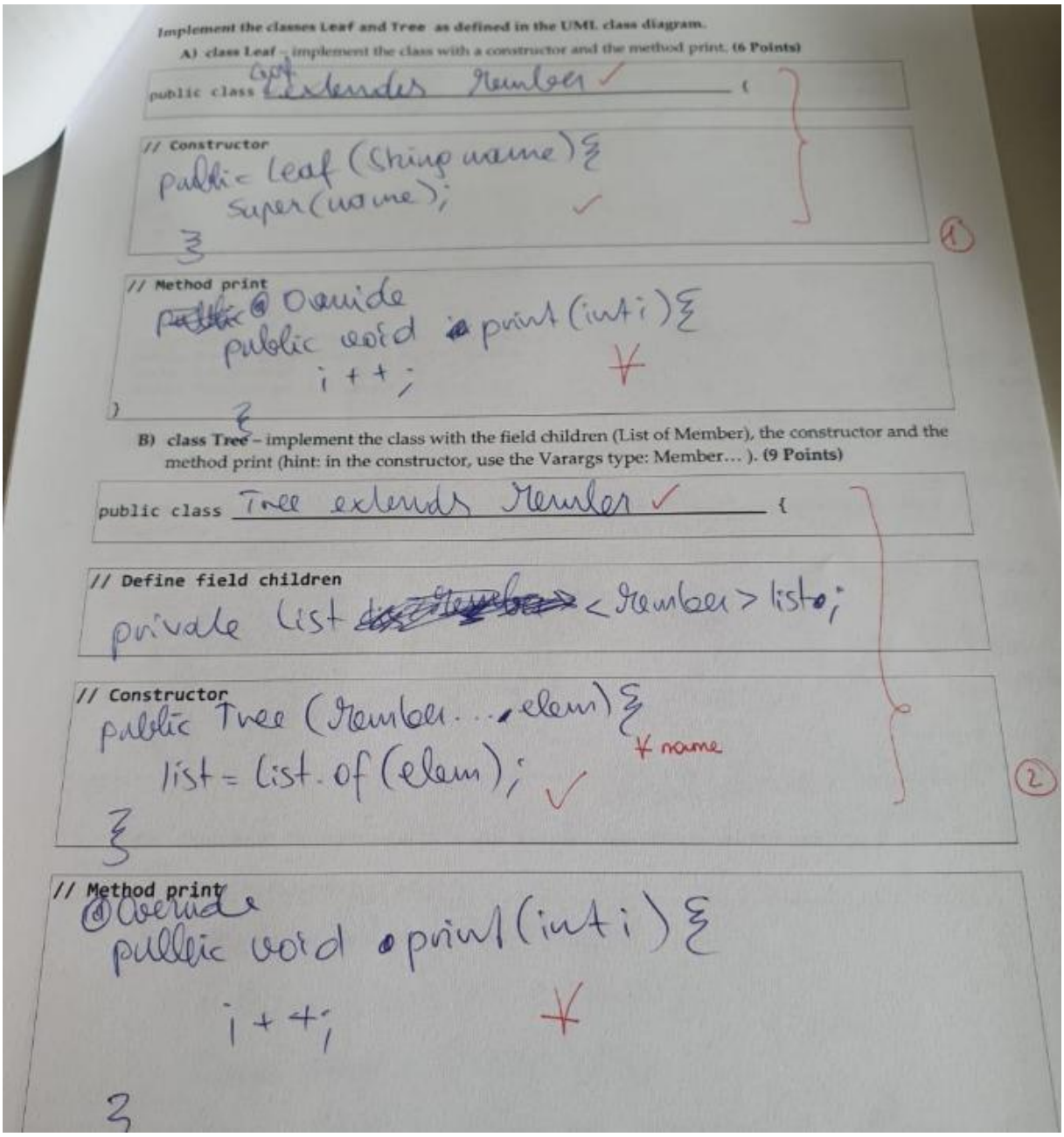
The printout of windsors should look as follows:

```
Queen Victoria
-King Edward VII
--King George V
---King George VI
----Queen Elizabeth II
-----King Charles III
------William
------Harry
```

The abstract class Member is given as follows (the level represents the hierarchy level):

```
public abstract class Member {
  String name;
  Member (String name) { this.name = name; }
  abstract void print(int level);
}
```

.....



Implement the classes Leaf and Tree as defined in the UML class diagram.

A) class Leaf – implement the class with a constructor and the method print. **(6 Points)**

```
public class Leaf extends Member ✓ {

// Constructor
public Leaf (String name) {
    super(name);
}

// Method print
@Override
public void print(int i) {
    i++;
}
}
```

① ✓ ✓ ✗

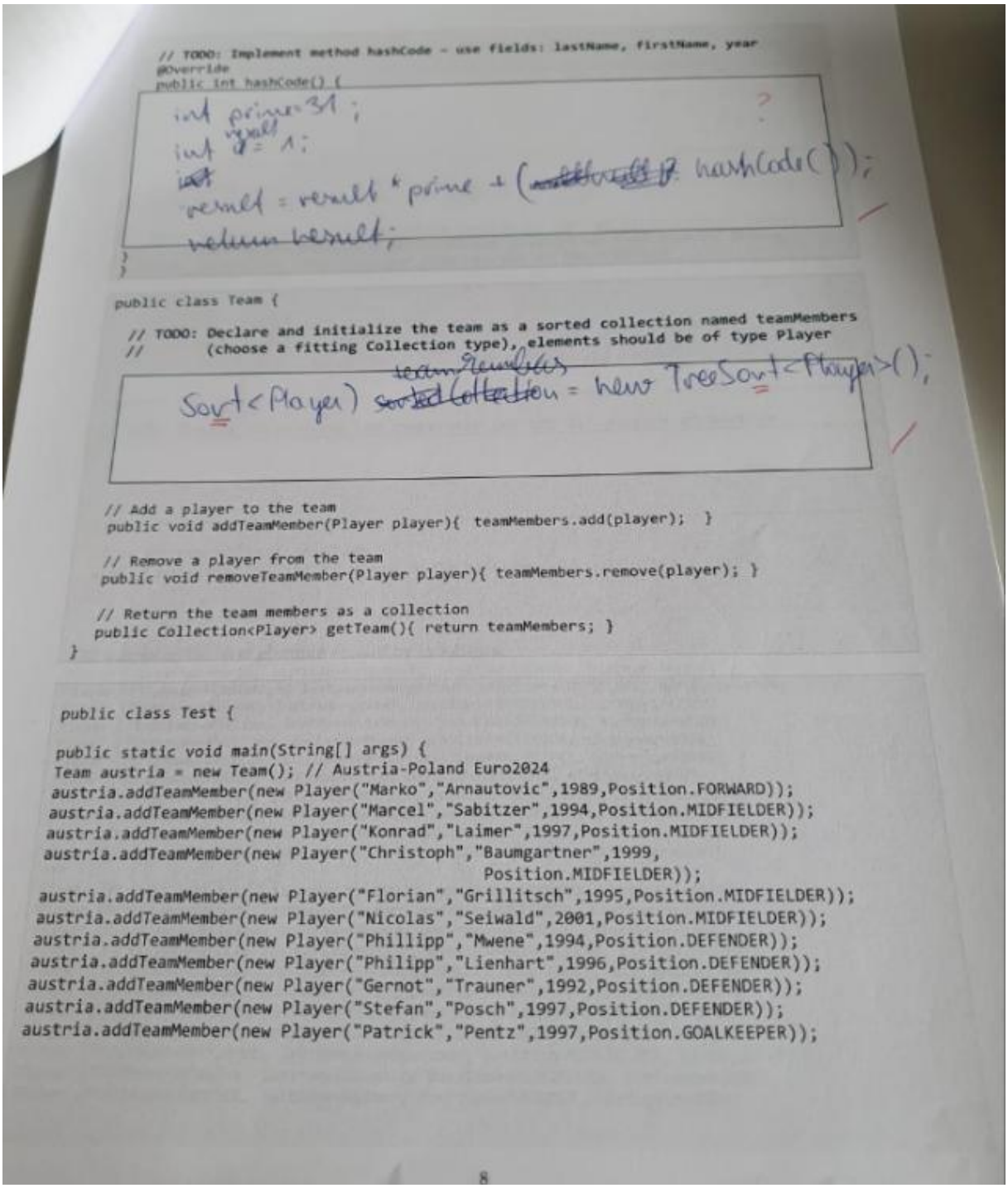
B) class Tree – implement the class with the field children (List of Member), the constructor and the method print (hint: in the constructor, use the Varargs type: Member…). **(9 Points)**

```
public class Tree extends Member ✓ {

// Define field children
private List<Member> list;

// Constructor
public Tree (Member... elem) {
    list = List.of(elem); ✓
}

// Method print
@Override
public void print(int i) {
    i++;
}
```

✗ name

② ✗

.....



```
// TODO: Implement method hashCode - use fields: lastName, firstName, year
@Override
public int hashCode() {
    int prime = 31;
    int result = 1;
    result = result * prime + (... hashCode());
    return result;
}
}

public class Team {

    // TODO: Declare and initialize the team as a sorted collection named teamMembers
    //       (choose a fitting Collection type), elements should be of type Player
    SortedSet<Player> teamMembers = new TreeSet<Player>();

    // Add a player to the team
    public void addTeamMember(Player player){ teamMembers.add(player); }

    // Remove a player from the team
    public void removeTeamMember(Player player){ teamMembers.remove(player); }

    // Return the team members as a collection
    public Collection<Player> getTeam(){ return teamMembers; }
}

public class Test {

    public static void main(String[] args) {
    Team austria = new Team(); // Austria-Poland Euro2024
    austria.addTeamMember(new Player("Marko","Arnautovic",1989,Position.FORWARD));
    austria.addTeamMember(new Player("Marcel","Sabitzer",1994,Position.MIDFIELDER));
    austria.addTeamMember(new Player("Konrad","Laimer",1997,Position.MIDFIELDER));
    austria.addTeamMember(new Player("Christoph","Baumgartner",1999,
                                     Position.MIDFIELDER));
    austria.addTeamMember(new Player("Florian","Grillitsch",1995,Position.MIDFIELDER));
    austria.addTeamMember(new Player("Nicolas","Seiwald",2001,Position.MIDFIELDER));
    austria.addTeamMember(new Player("Phillipp","Mwene",1994,Position.DEFENDER));
    austria.addTeamMember(new Player("Philipp","Lienhart",1996,Position.DEFENDER));
    austria.addTeamMember(new Player("Gernot","Trauner",1992,Position.DEFENDER));
    austria.addTeamMember(new Player("Stefan","Posch",1997,Position.DEFENDER));
    austria.addTeamMember(new Player("Patrick","Pentz",1997,Position.GOALKEEPER));
```

8

....

```
// TODO: Exchange Florian Grillitsch with Patrick Wimmer,2001,Position.DEFENDER
//       in Team austria

// print team
austria.getTeam().forEach(System.out::println);

// TODO: Create a new collection aTeam ordered by year of birth (eldest first)
//       then by the natural order of Player (collection type same as before)
// TODO: Create the comparator for aTeam to define the ordering

// TODO: Create aTeam using the comparator and add all players of austria

// print new collection aTeam
System.out.println("***"); aTeam.forEach(System.out::println);
}
}
```

The output of the Test program should be as follows:

```
Player [firstname=Marko, lastname=Arnautovic, position=FORWARD, birthyear=1989]
Player [firstname=Christoph, lastname=Baumgartner, position=MIDFIELDER, birthyear=1999]
Player [firstname=Konrad, lastname=Laimer, position=MIDFIELDER, birthyear=1997]
Player [firstname=Philipp, lastname=Lienhart, position=DEFENDER, birthyear=1996]
Player [firstname=Phillipp, lastname=Mwene, position=DEFENDER, birthyear=1994]
Player [firstname=Patrick, lastname=Pentz, position=GOALKEEPER, birthyear=1997]
Player [firstname=Stefan, lastname=Posch, position=DEFENDER, birthyear=1997]
Player [firstname=Marcel, lastname=Sabitzer, position=MIDFIELDER, birthyear=1994]
Player [firstname=Nicolas, lastname=Seiwald, position=MIDFIELDER, birthyear=2001]
Player [firstname=Gernot, lastname=Trauner, position=DEFENDER, birthyear=1992]
Player [firstname=Patrick, lastname=Wimmer, position=DEFENDER, birthyear=2001]
***
Player [firstname=Marko, lastname=Arnautovic, position=FORWARD, birthyear=1989]
Player [firstname=Gernot, lastname=Trauner, position=DEFENDER, birthyear=1992]
Player [firstname=Phillipp, lastname=Mwene, position=DEFENDER, birthyear=1994]
Player [firstname=Marcel, lastname=Sabitzer, position=MIDFIELDER, birthyear=1994]
Player [firstname=Philipp, lastname=Lienhart, position=DEFENDER, birthyear=1996]
Player [firstname=Konrad, lastname=Laimer, position=MIDFIELDER, birthyear=1997]
Player [firstname=Patrick, lastname=Pentz, position=GOALKEEPER, birthyear=1997]
Player [firstname=Stefan, lastname=Posch, position=DEFENDER, birthyear=1997]
Player [firstname=Christoph, lastname=Baumgartner, position=MIDFIELDER, birthyear=1999]
Player [firstname=Nicolas, lastname=Seiwald, position=MIDFIELDER, birthyear=2001]
Player [firstname=Patrick, lastname=Wimmer, position=DEFENDER, birthyear=2001]
```


.....

A 5 – Streams (15 Points)

The class Movie contains the movie title, year of production, director, a set of actors, and rating:

```
public class Movie {
        private String title;
        private int year;
        private String director;
        private Set<String> actors;
        private long rating;

        public Movie (String t, int y, String d, Set<String> actors, long r){
          this.title = t;
          this.year = y;
          this.director = d;
          this.actors = actors;
          this.rating = r;
        }

       String getTitle() { return title; }
       int getYear() { return year; }
       String getDirector () { return director; }
       Set<String> getActors() { return actors; }
       long getRating() { return rating; }
}
```

The list movies contains the following movies:

```
List<Movie> movies = List.of(
   new Movie ("Anora",2024,"Sean Baker", Set.of("Mikey Madison", "Mark Eydelshteyn", "Yury
Borisov"), 3),
   new Movie ("Evil Does Not Exist",2023,"Ruyusuke Hamaguchi", Set.of("Hitoshi Omika", "Ryo
Nishikawa", "Ryuji Kosaka"), 3),
   new Movie ("Poor Things", 2023, "Yorgos Lanthimos", Set.of("Emma Stone","Willam Dafoe",
"Mark Ruffalo"), 4),
   new Movie ("Tenet",2020, "Christopher Nolan", Set.of("Elizabeth Debicki","John David
Washington", "Robert Pattinson"), 3),
   new Movie ("La La Land", 2016, "Damien Chazelle", Set.of("Emma Stone","Ryan Gosling","Fin
Wittrock"), 4),
   new Movie ("Casablanca", 1942, "Michael Curtiz", Set.of("Humphrey Bogart","Ingrid
Bergman", "Peter Lorre"), 5),
   new Movie ("Whiplash", 2014, "Damien Chazelle", Set.of("Miles Teller","Melissa
Benoist","J.K. Simmons"), 5),
   new Movie ("Avengers: Infinity War", 2018, "Anthony Russo, Joe Russo",Set.of("Robert
Downey Jr.","Chris Hemsworth, Scarlett Johansson, Benedict Cumberbatch"), 4));
```

Use the following STREAM OPERATIONS to answer the following questions about the list movies:

- `Stream<T> filter(Predicate<? super T> predicate)`
- `Stream<T> sorted()`
- `Stream<T> sorted(Comparator<? super T> comparator))`
- `Stream<T> distinct()`
- `<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper)`
- `<R> Stream<R> map(Function<? super T, ? extends R> mapper)`
- `void forEach(Consumer<? super T> action)`
- `Optional<T> findAny()`
- `long count()`
- `<R, A> R collect(Collector<? super T, A, R> collector)`
    - *To create a list:* `static Collectors<T, ?, List<T>> toList()`
- `Optional<T>: T get()`

.....

1) In how many movies is Emma Stone one of the actors? **2 Points** 1

```
long emmaStoneCount =
    (long)movies.stream().filter(e->e.equals("Emma Stone")).
    count();
```

2) Return an alphabetically sorted list of all movie titles contained in the movies list. **4 Points** 1

```
List<String> movieTitles =
    movies.stream().sorted().forEach(
    e -> System.out.println(e));
```

Return!

3) Print out all movie titles sorted by year ("aufsteigend") where the director is Damien Chazelle. **4 Points** 1

```
List<String> movieTitles = movies.stream().filter(e->
    e.compareTo(year).sort(A).forEach(System.out::
    print);
```

4) Return an actor (any actor out of all actors) who played in a movie rated with 5. **5 Points** 2

```
String actorsFiveStars =
    movies.stream().anyMatch(e->e.rating==5).
    get();
```


...

A 6 – Model View Controller (20 Points)

Implement a Swing application that allows to change the color of a circle between red and green by using the menu item Control->Switch:

Green circle

Red circle

Based on the MVC (Model View Controller) architecture, the following interfaces and classes are used:

Model

- `Class ColorModel` is the model; it contains a switch counter incremented when the menu item Control->Switch is clicked. If the counter is even ("gerade"), the color is green, otherwise red.
- `Interface ColorListener` is a listener for changes of `ColorModel`.
- `Class ColorEvent` is the respective class for the color change event.

View/Controller

- `Class ColorViewController` and `class ColorPanel` are the classes responsible for visualizing the color status and observing user clicks on the menu item Control->Switch.

The application is pre-programmed. Read the code, then **fill-in code parts of the interaction between model and view/control in the gap areas (//TODO:** note the comments right before the areas).

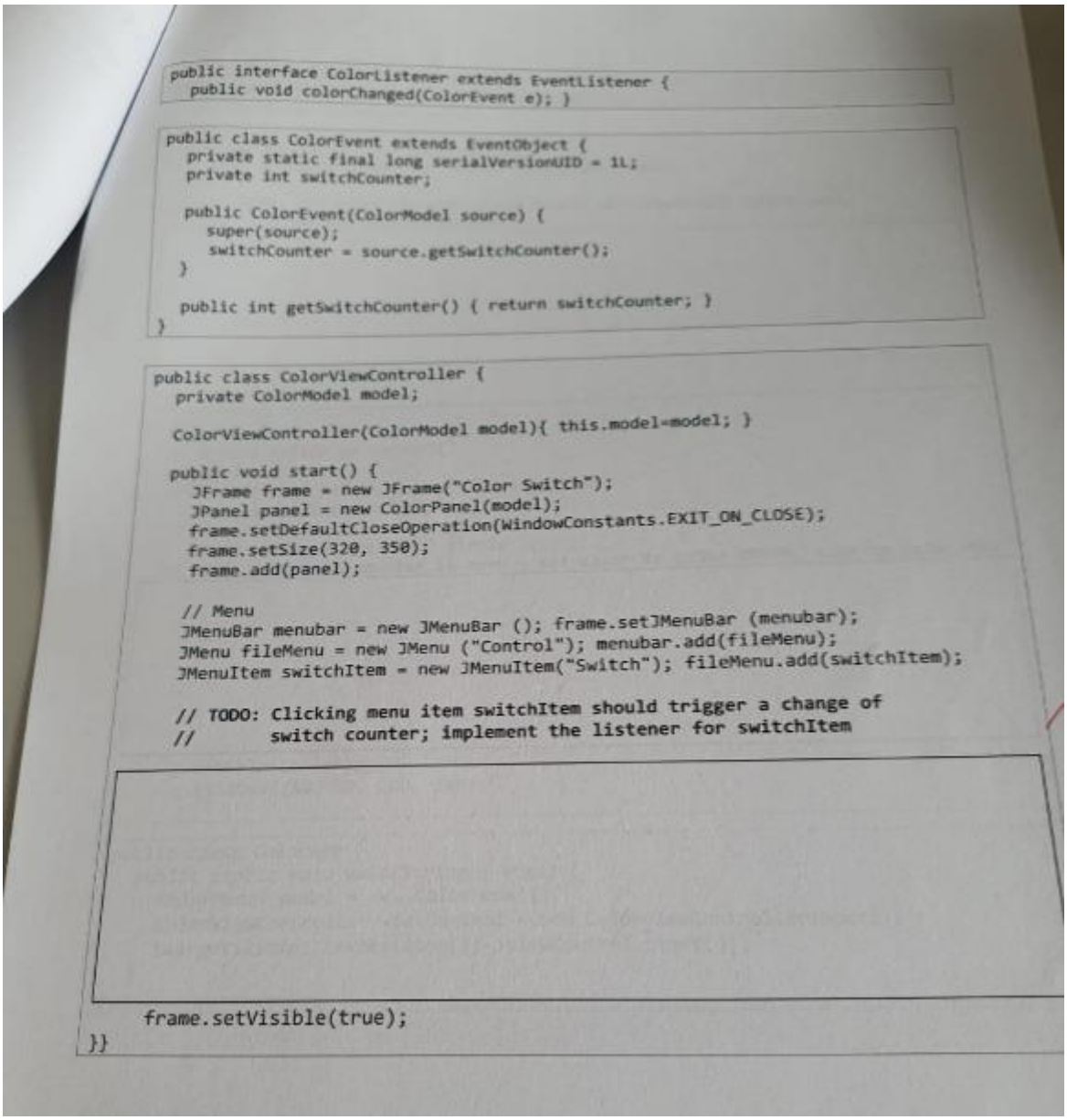
```
public class ColorModel {
  private int switchCounter = 0; // even->green, odd->red
  private List<ColorListener> listeners = new ArrayList<ColorListener>();

  public void addColorListener(ColorListener l) { listeners.add(l); }
  public void removeMColorListener(ColorListener l) { listeners.remove(l); }

  public int getSwitchCounter() { return switchCounter; }

  public void setSwitchCounter() { // Method that changes the switch counter
    switchCounter += 1;

  // TODO: switch counter has changed, call the fire event method
```

```
}
```

```
// TODO: Implement the fire event method
```

```
}
```


...



```
public interface ColorListener extends EventListener {
  public void colorChanged(ColorEvent e); }

public class ColorEvent extends EventObject {
  private static final long serialVersionUID = 1L;
  private int switchCounter;

  public ColorEvent(ColorModel source) {
    super(source);
    switchCounter = source.getSwitchCounter();
  }

  public int getSwitchCounter() { return switchCounter; }
}

public class ColorViewController {
  private ColorModel model;

  ColorViewController(ColorModel model){ this.model=model; }

  public void start() {
    JFrame frame = new JFrame("Color Switch");
    JPanel panel = new ColorPanel(model);
    frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    frame.setSize(320, 350);
    frame.add(panel);

    // Menu
    JMenuBar menubar = new JMenuBar (); frame.setJMenuBar (menubar);
    JMenu fileMenu = new JMenu ("Control"); menubar.add(fileMenu);
    JMenuItem switchItem = new JMenuItem("Switch"); fileMenu.add(switchItem);

    // TODO: Clicking menu item switchItem should trigger a change of
    //       switch counter; implement the listener for switchItem

    frame.setVisible(true);
}}
```

...