CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
DIT-043	Objektorienterad programmering			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
~~DIT043~~-0022-AJL		04.01.23	15	5

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	18	
2	X	41	
3	X	30	
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng på tentamen		89	

Overloading entails a method having the same name, but diffrent ~~signature~~, i.e. a different set of arguments

Overriding on the other hand is when a subclass overrides a method of its superclass, meaning it needs to have the ~~same~~ signature.

↳ Replaces old method

6/8

Charlie is Correct. Overloading is useful.

2

Spaces are denoted with the underscore character (_), since it is otherwise unclear.
There are no empty lines
Answer bellow:

x=_6_y=_1
x=_-2_y=_3
x=_7_y=_8
total=_22
p1_=_16_p2_=_23
total=_27
x=_-2_y=_3
x=_7_y=_8
x=_-2_y=_3
x=_7_y=_8
x=_-2_y=_8
x=_7_y=_8

Awesomely done :)

12/12

2

The field "temperatures" in the WeatherRecorder class is never initialized, and is therefore null.

The consequences of this is that whenever, for instance, a method on this null reference is called, such as get, add or size, a NullPointerException will be thrown. In the main method, that would be on line 47, assuming that registerDator would in turn call temperatures.add(x).

5/5 Perfect explanation! :)

To fix the problem, one can make sure that the field is initialized like so:

```
private List<TemperatureDator> temperatures
    = new ArrayList<>();
```

Alternatively, the field can be initialized in the constructor

2

```
public boolean registerData(double temperature,
                            String month) {

    for (String m : MONTHS){
        if(m.equals(month)){
            TemperatureData data =
              new TemperatureData(temperature,
                                  month);

            temperatures.add(data);
            return true;
        }
    }
    return false;
}
```

or shorter ↓

```
if (MONTHS.contains
    (month))
```

8/8

Awesomely done! (y)

2

```
public String getLowestTemperatureData(){
    if(temperatures.isEmpty()){
        return "";
    }
    TemperatureData lowest = temperatures.get(0);
    for(int i=1; i<temperatures.size(); i++){
        TemperatureData current = temperatures.get(i);
        if(current.temperature < lowest.temperature){
            lowest = current;
        }
    }
    return lowest.toString();
}
```

Small implementation! :)

5/5

```
public String getSmallestAbsDiff() {
    if(temperatures.size() < 2) {
        throw new RuntimeException("Invalid request."
            +" We require more than 2 temperatures registered");
    }
    TemperatureData lowA = temperatures.get(0);
    TemperatureData lowB = temperatures.get(1);

    for(TemperatureData curA : temperatures) {
        for(TemperatureData curB : temperatures) {
            if(curA != curB && Math.abs(curA.temperature
                - curB.temperature) < Math.abs(
                lowA.temperature - lowB.temperature){
                lowA = curA;
                lowB = curB;
            }
        }
    }
}
```

10/10

Perfect implementation :)

//next page

```
return "Diff: " + Math.abs(lowA.temperature
    - lowB.temperature) + ". Between "
    + lowA.toString() + " and " lowB.toString();
}
```

2

Disagree. Regardless of whether the elements of a list are immutable, they can still be removed from the list as long as the list itself is not immutable or otherwise unmodifiable, which is the case here.

Since the list itself is mutable, the method can be implemented

5/5

good job

2

WeatherRecorder is a composite composite object where TemperatureData is an indirect component

4/6

forwarding?

line numbers?

2

Scanner can not be used, since its purpose is for reading from input streams, not writing to output streams.

An Object output stream can not be used, since that would require a modification of the TemperatureDator class, in the form of implementing the Serializable interface.

Therefore, out of the 3 options, you'd use the text output stream → why?

4/5

2

Disagree. All the fields are private and access is controlled through setters and getters.

Now if the intention is to limit access to changing the name or the health value directly, that is another issue. The class is well encapsulated since it gives no direct access to its fields

3p

→ Encapsulation is not just about access. It's control over state changes. HP is incorrect in the class.

2

1. Single-Responsibility Principle,

2. Open-Closed Principle

Inheritance can improve the design, as well as make sure the code follows both of these principles. Each class, whether the base Champion class, or the subclasses for each of the subtypes, will have a single responsibility of handling the behavior specific to it. The other principle can also then be followed by the design being closed to modification, but open to extension by implementing new sub-classes for new types of champions.

6p

1. Dependency inversion principle.

The principle states that high level abstractions should not depend on low level details. As an example, the principle would not be followed if a method that downloads and saves a file to disk, would deal with the details of resolving a domain name, or opening a TCP connection.

↳ How to see this in the class relationship ?

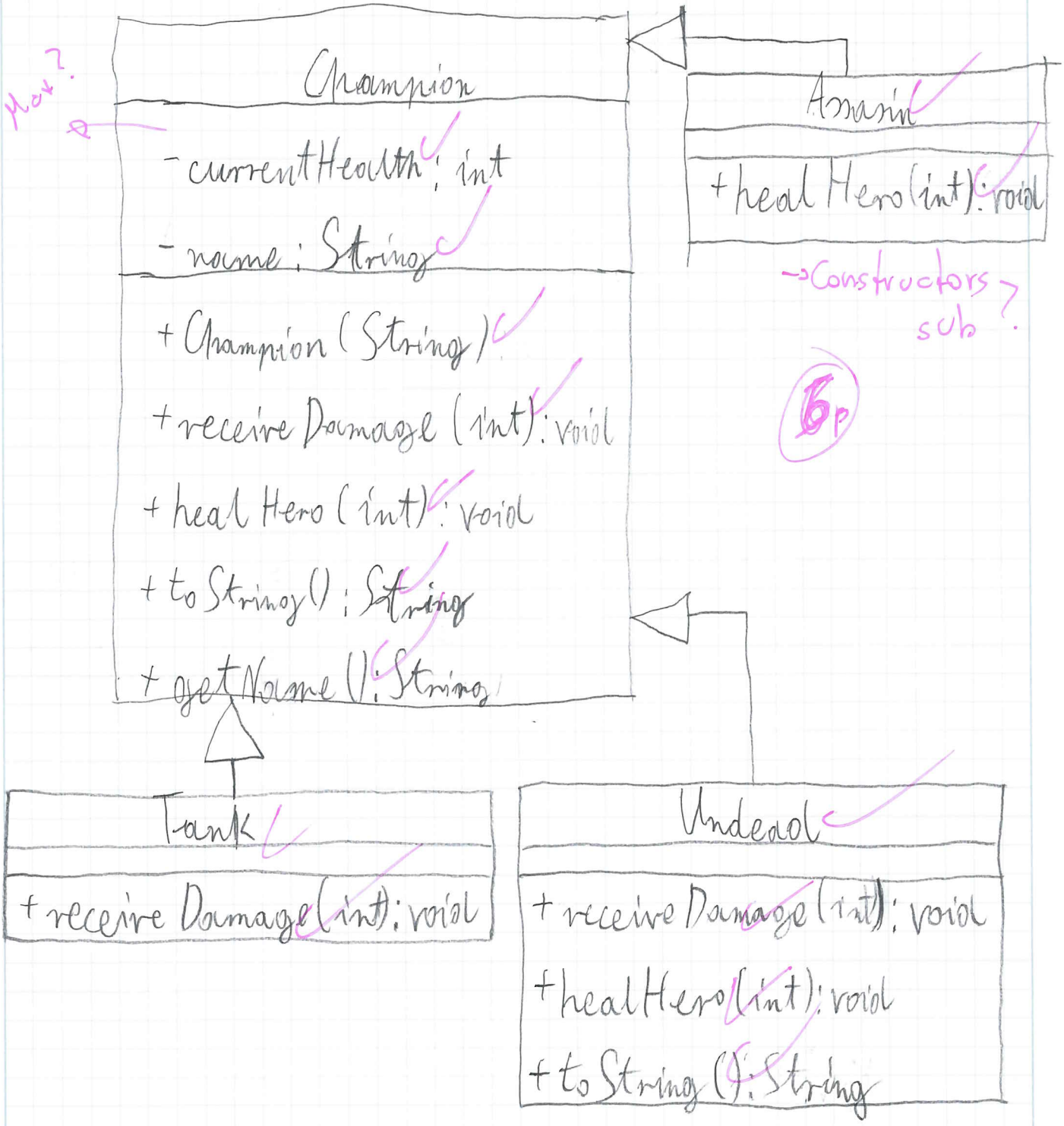
8p

2. Liskovs substitution principle.

States that any subclass should be able to substitute its superclass in its place. In the instance of the Champion classes, being able to substitute a subclass in the place of the superclass would mean that the principle is followed

↳ Circular explanation. Clarify how.

2



How?

→ Constructors sub?

6p

```
public class Undead extends Champion {
    public Undead(String name) {
        super(name);
    }

    @Override
    public void receiveDamage(int amountDamage) {
        super.healHero(amountDamage);
    }

    @Override
    public void healHero(int amountHealed) {
        super.receiveDamage(amountHealed);
    }

    @Override
    public String toString() {
        return "Undead " + super.toString();
    }
}
```

7p