# OOP2 Assignment 2 – 2022-2023 – Term 2

## Introduction

The total assignment consists of three parts, each with their own weight in the final grade. This is the first one. The deadline is January 29th 2023, 23:00. This is an individual assignment; it is not allowed to work in groups.

When scoring less than a 5.5 for an assignment you get feedback and have two weeks starting from the moment of feedback to hand in an improved version. When not handing in the first version before the deadline (or handing in something so bad it was graded at no more than a 1.0) no feedback is given and the deadline for the improved version is February 12th 2023, 23:00.

## Context

There is a zoo, somewhere. This zoo has a very hierarchical structure: the zoo itself is considered a Zone, which has subzones, each of which can have subzones itself, up to as many levels of subzones as is deemed necessary. Each (sub)zone can have zero or more Pens, which is where the Animals are kept.

Since not all animals would be friendly to each other the decision was made to only have one Species of animal in a pen. A zone is considered to be for one species as well, but that turned out to cause read headaches. It was decided to never enforce this rule, only define it (so in your code make sure a Pen is parameterized for an Animal, but don’t bother to enforce it in any other way.)

The zoo has all kinds of personnel, but for this assignment only Caretakers are taken into account. A caretaker is the one feeding the animals (other duties are not relevant for this assignment.) Each animal has one primary caretaker, and each caretaker knows which animals are under his / her care.

For now, the concept of Food is very much simplified. Food is something with a feedingValue between 0 and some unspecified maximum, we assume any food can be eaten by any animal. An animal has a hungriness value, which can be between 0 (not hungry at all) and a species-defined maximum (ravingly hungry.) Hungriness increases over time (depending on the type of species) and is decreased by feeding.

The hungriness value of an animal is very important; if it reaches its maximum the animal might become dangerous, so if this happens all caretakers need to be informed. Since each animal has its own quirks and maybe dietary requirements the rule is that only the primary caretaker feeds it. It will not be blocked that another person feeds the animal – after all, the caretaker might be sick – but if anyone else does feed the animal, the primary caretaker should be informed.

How does the zoo get its animals? There are several ways, depending on the kind of animal and the situation. Each species is of a certain type (for example tigers are scary, a tarantula is creepy and a panda is fluffy.) The zoo specified that by default common or fluffy animals will be adopted and creepy or scary animals will be bought. This can be overridden in two ways however: by specifically specifying it (in the application code) or by placing a plugin in a specific directory which overrides the default behavior.

All acquirements of new animals cost money (except adopting, which is free.) The zoo keeps a Till with money for this purpose; if there is no money, the animal cannot be acquired (except by adoption, which is free.)

## Assignment

The application will consist of two modules, zoo and util. For util a project with a starting point has been given. The code is not complete; you will have to add classes and interfaces, change existing methods to implement necessary design patterns, and add code to make it all work. The application structure also is bad; correct it to follow a strong architecture.

In the code you will find comments; make sure to use all of it to make the application work as specified in the context.

You need to create a second project called util and put classes and/ or interfaces there which would work for all kind of projects. Have this one export the module util and have the module zoo use it.

Rename both projects to include your name and student number. Then export both projects to a zip file and put the password *Welkom01* on it before uploading it to Moodle.

## Grading

Your work is graded according to the following table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Weight** | **Topic** | **Excellent (10)** | **Good (8)** | **Sufficient (6)** | **Minimal (4)** | **Poor (2)** |
| 3 | Architecture | Classes have been well defined.  The Liskov Subsitution Principle and DRY have been (mostly) followed.  Methods are defined and implemented in the suitable classes  Classes are placed in packages suiting a strong architecture such as layering and/ or subsystems  Classes from different packages are weakly correlated. Packages are strongly cohesive.  Interactions between packages are unidirectional  The code compiles without needing modifications | Classes have been well defined.  The Liskov Subsitution Principle and DRY have been (mostly) followed.  Methods are defined and implemented in the suitable classes  Classes are placed in packages suiting a strong architecture such as layering and/ or subsystems  Classes from different packages are weakly correlated. Packages are strongly cohesive.  The code compiles without needing modifications | Classes have been well defined.  The Liskov Subsitution Principle and DRY have been (mostly) followed.  Methods are defined and implemented in the suitable classes  Classes are placed in packages suiting a strong architecture such as layering and/ or subsystems  The code may or may not compile, but making it do so will only need trivial modifications | Classes have been well defined.  The Liskov Subsitution Principle *and/ or* DRY has been (mostly) followed.  Methods are defined and implemented in the suitable classes  The code may or may not compile, but making it do so will only need trivial modifications | Classes have been well defined.  The Liskov Subsitution Principle *and/ or* DRY has been (somewhat) followed.  The code may or may not compile, but making it do so will only need trivial modifications |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | Design Patterns | All Design Patterns asked have been implemented  The implementation is according to the GoF specification; no non-trivial errors have been made | All Design Patterns asked have been implemented; one missed pattern is acceptable.  The implementations follow the GoF specification  At most one implementation has non-trivial errors  Despite the errors the purpose of the patterns has still been achieved | All Design Patterns asked have been implemented; one missed pattern is acceptable.  The implementations follow the GoF specification, but have non-trivial errors  Despite the errors the purpose of the patterns has still been achieved | Most Design Patterns asked have been implemented (no more than one has been missed)  The implementations follow the GoF specification, but have non-trivial errors.  Because of these errors the purpose of the pattern(s) has not been achieved | At least one Design Pattern has been implemented  None of the implementations is according to the GoF specification  *or*  the implementations have non-trivial errors  The purpose of the design patterns has not been achieved due to the errors made |
| 1 | Inner classes | Where applicable inner classes and anonymous inner classes have been used  The inner class(es) have the correct level of encapsulation  The inner class(es) access members of the outer class in the most effective and efficient way, not using accessor methods | Where applicable inner classes and anonymous inner classes have been used  No more than 25% of the instances where an (anonymous) inner class could have been used has been missed  The inner class(es) have the correct level of encapsulation  The inner class(es) access members of the outer class in the most effective and efficient way, not using accessor methods | Where applicable inner classes and anonymous inner classes have been used  No more than 25% of the instances where an (anonymous) inner class could have been used has been missed  The inner class(es) have the correct level of encapsulation  The inner class(es) access members of the outer class using accessor methods | Where applicable inner classes and anonymous inner classes have been used  No more than 50% of the instances where an (anonymous) inner class could have been used has been missed  *or*  The inner class(es) do not have the correct level of encapsulation  The inner class(es) access members of the outer class using direct access or accessor methods | Only standard inner classes have been used  The inner class(es) do not have the correct level of encapsulation  The inner class(es) access members of the outer class using accessor methods or non-standard ways |
| 1 | Lambda expressions | Lambda expressions have been used to replace simple anonymous inner classes  More complicated or large anonymous inner classes have not been replaced by lambda expressions  The format and implementation of the expression is correct | N/A | Lambda expressions have been used to replace simple anonymous inner classes  The format and implementation of the expression is correct with trivial errors | N/A | Lambda expressions have been used to replace simple anonymous inner classes  The implementation of the expression is incorrect; non-trivial errors have been made |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Java modules | No automatic module has been created  At least two modules have been defined  One module requires the other module  One of the used modules has been opened to allow reflective access  The difference between classpath and modulepath has been taken into account correctly | At least two modules have been defined  One module requires the other module  One of the used modules has been opened to allow reflective access  The difference between classpath and modulepath has been taken into account correctly | At least two modules have been defined  One module requires the other module  The difference between classpath and modulepath has been taken into account correctly | Modules have only been defined for export, not for requiring  *or*  Only requiring modules have been defined, not exporting  The difference between classpath and modulepath has been taken into account correctly | Modules have only been defined for export, not for requiring  *or*  Only requiring modules have been defined, not exporting |
| 1 | Writing plugins | Using the default ClassLoader and the Reflection API a plugin has been created  The implementation handles the Java Security mechanism correctly  The implementation handles errors gracefully  An auxiliary class has been created to test and demonstrate the ability of the plugin | Using the default ClassLoader and the Reflection API a plugin has been created  The implementation handles the Java Security mechanism correctly  The implementation handles some but not all errors gracefully  An auxiliary class has been created to test and demonstrate the ability of the plugin | Using the default ClassLoader and the Reflection API a plugin has been created  The implementation handles the Java Security mechanism correctly  The implementation handles some but not all errors gracefully | Using the default ClassLoader and the Reflection API a plugin has been created  The implementation handles at most the most basic errors and exceptions | Using the default ClassLoader and the Reflection API a plugin has been created  The plugin has the bare structure it needs but is incorrect and/ or incomplete otherwise |

If a criterion does not achieve the standard described under Poor a score of Absent (0) is assigned. For each topic you score between 0 (Absent) and 10 (Excellent) points. The base grade is the weighted average of points scored, with the weights as indicated in the table.

All criteria need to score at least the green level; if no level for a criterion is green there is no minimum. If one or more criteria score one level below the minimum score the maximum grade is a 4.0. If one or more criteria score two or more levels below the minimum score the maximum grade is a 2.0.

**If the work handed in cannot be imported as a Java or Maven project into Eclipse the grade will be a 1.0, and no feedback is given. An improved version can be handed in as a resit; see the rules at the beginning of this document. If for the resit the work handed in cannot be imported the grade is a 1.0 and there is no more resit opportunity this year.**