Assignment 5: Collections, genericity, lambda expressions and design patterns

Part 1: Generic forms for console data input.

In this part we created the classes **form** and **field**. The class form contains several questions that are stored in a map. This map contains as a key the question itself and as a value the *field* that validates the input.

The field uses 2 functional interfaces, the first one is *Function<String, T>*, this one is used because given an input (String) we need to transform that input into a special type (the generic type T). The other functional interface is the *Predicate<T>*, this one receives a condition and checks if the input (with the type T) fulfils that condition. The *predicate* is used to add validations in the field, so in order to check whether a field is okay or not we just use the *test* method of the interface.

Given the fields of the form the method *exec* asks for an input, and then checks the fields.

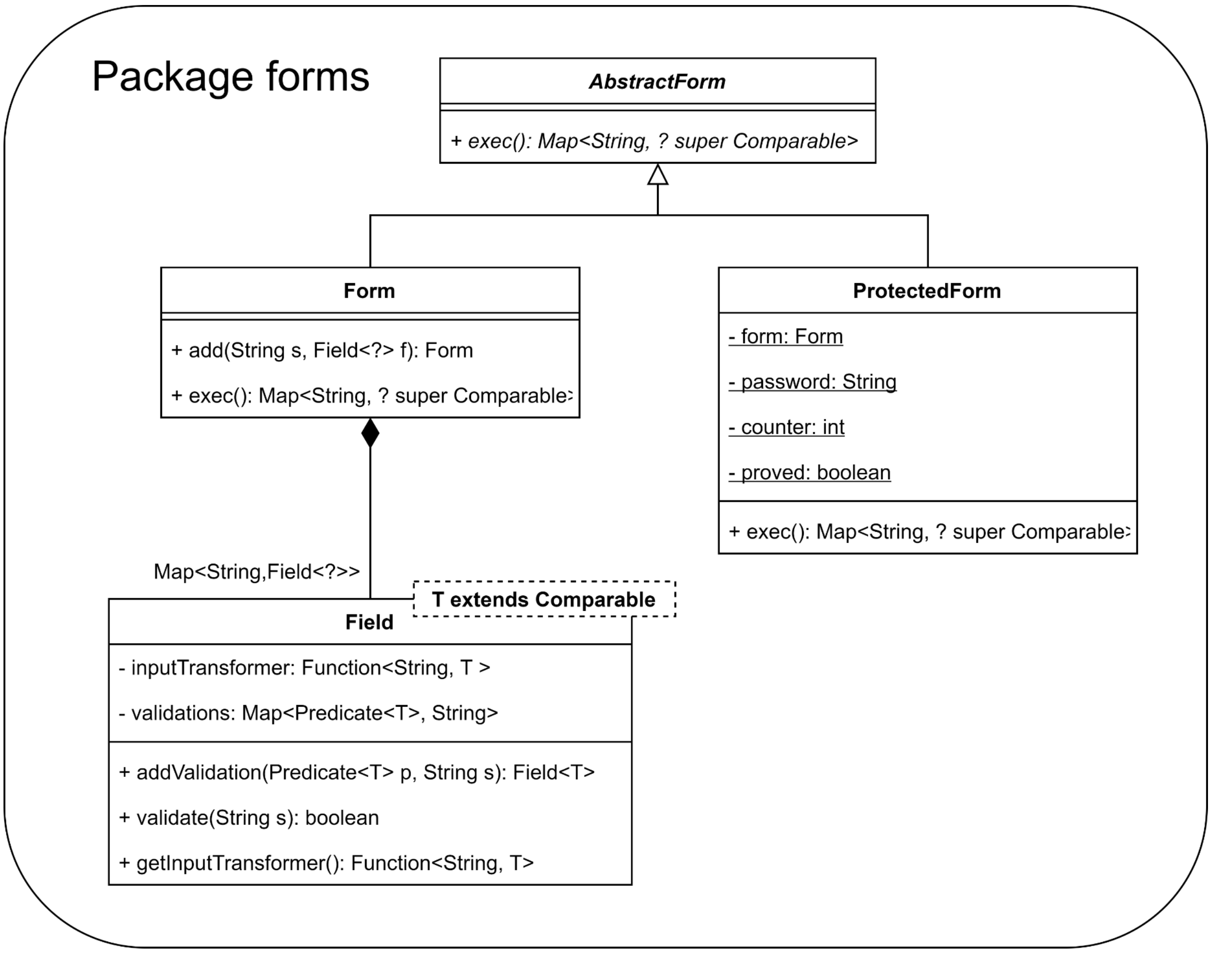
Part 2: Processing the collected data.

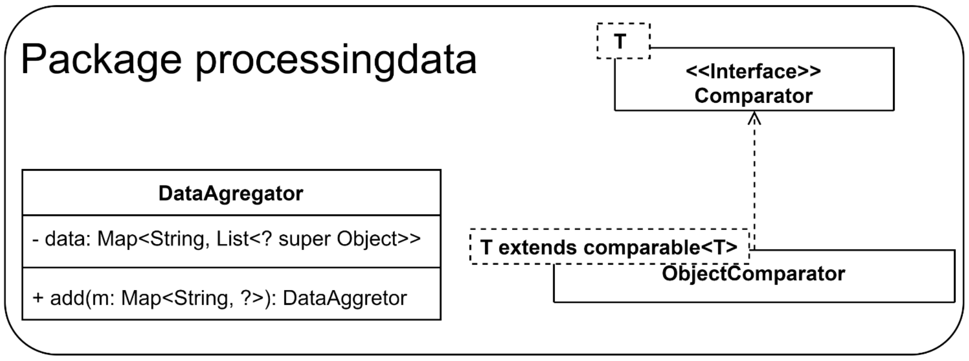
For this part the goal was to aggregate data to a class. Our approach was to create the class **DataAggregator** and inside add a *LinkedHashMap* to store all the data needed. In order to have all this data in ordered we had to make the class **Address** extend the **Comparable** class. So all the objects inside the data should be ordered and for this reason all these objects need to extend the *comparable* class. Once the objects use *comparable* class we need to create a comparator class for the data, this class is **ObjectComparator**, it implements the interface **Comparator** of any object which is comparable. This class is used to sort our data.

Part 3: Protecting the form by means of a password.

In this part we implement the Proxy design pattern, which needs the implementation of two new classes. The first one, the class **AbstractForm** which is an abstract class from which two other classes will inherit, the **Form** class already implemented, and the **ProtectedForm** class which works as a substitute of the Form class whose main purpose will be to control the access to the Form. This way whenever a form executes the function *protect()* from the ProtectedForm class it will return a ProtectedForm which will be the substitute of the original form. The protection of the form will have three attempts and it will require to enter the password only once, therefore to keep track of the attempts we use a private static integer variable, to keep track if it has been already accessed the form we use a private static boolean variable, and to check whether the password is correct or not we use the a private static String variable. The original form will be stored in this class as a static variable as well.

Class Diagram
The following picture represents the architecture we followed in order to implement every part of the assignment. We have made sure the generic classes are represented as it corresponds.


Package forms
AbstractForm
+ exec(): Map<String, ? super Comparable>
Form
+ add(String s, Field<?> f): Form
+ exec(): Map<String, ? super Comparable>
ProtectedForm
- form: Form
- password: String
- counter: int
- proved: boolean
+ exec(): Map<String, ? super Comparable>
Map<String,Field<?>>
T extends Comparable
Field
- inputTransformer: Function<String, T >
- validations: Map<Predicate<T>, String>
+ addValidation(Predicate<T> p, String s): Field<T>
+ validate(String s): boolean
+ getInputTransformer(): Function<String, T>


Package processingdata
T
<<Interface>>
Comparator
DataAgregator
- data: Map<String, List<? super Object>>
+ add(m: Map<String, ?>): DataAggretor
T extends comparable<T>
ObjectComparator