"Duck shooter"

Write a program that is a game like "duck shooter". During the game, ducks of various colors move from left to right and from right to left. In game we need 'shoot' the ducks and prevent them from getting to the other side of the window. The player clicks on the ducks as many times as needed to 'shoot them down'.

For example: Yellow Ducks will need 1 click, Red Ducks will need 5 clicks, Purple Ducks will need 10, and Pink Ducks will need 20. These are sample values and colors.

The player's goal is to keep the field clean without ducks for as long as possible. The game ends when more than 10 ducks get to the other side of the window (the player has 10 lives, and each passing a duck to the other side takes 1 life).

You should also implement obstacles (at least clouds and trees) that will cover the chickens and protect them from being clicked. Clouds are moving obstacles that must move left or right.

Additionally, an improvement system should be implemented, in which we will be able to 'improve' our weapons to a certain rational extent. We will pay for upgrades with points that we will earn. The game is supposed to get heavier every 5 seconds - we can do it by speeding up the ducks, multiplying obstacles or increasing the "life" of the ducks.

A fully functional graphical interface should be provided. The command line console (*CLI*) can only be used as a help, but no user interaction with the program can occur there.

After starting the program should display the main menu consisting of the options:

- *New Game*
- *High Scores*
- *Exit*

After starting a new game, player will be asked in a separate window about the game difficulty level (at least 3 levels). After selecting level of difficulty, game window is displayed in the new window and time counter starts (it's worth noting that the time counter, ducks behavior and others, must be ***implemented in separate threads using Thread class***). During game must be visible points and time counter, which are constantly updated. The game is played according to the rules mentioned above. It should be possible to interrupt game at any time through the ***compound keyboard shortcut*** (***Ctrl+Shift+Q***), which will return you to the main menu.

After finishing the game, in the new window the player is asked for his name under which he will be saved in the ranking. Ranking is calculated based on the time, effect obtained and difficulty level (any implementation). You should save the ranking so that you do not lose saved records after closing the application. You must use interface ***Serializable***).

After selecting the ranking option from the main menu, it is displayed to the user. There may be a relatively large number of saved results, so you should take care of scrollbars in case it does not fit in the window of a reasonable size.

Hints:

- Take care of exceptions in the program. If any occurs, display its message to the user.
- High Scores list must be implemented using ***JList*** component and own data model using ***AbstractListModel***
- Ducks can be implemented using buttons (however, so that it looks aesthetic), but you can also design your own component.
- Not all windows need to be implemented via the JFrame class. Dialogs can be used.
- Take care of the appearance of the application

The project is based on GUI material.

The MVC design pattern should be used in the project.

Attention:

- ***It is not possible to use WYSIWYG tools to generate windows (e.g. Window Builder).***
- ***In the case of receiving a project with significant deficiencies in implementation or a non-compiling solution, the result for such a project will be 0 points.***
- ***Lack of knowledge of any line of code or plagiarism will result in obtaining 0 points for this project with the possibility of failing the entire subject.***
- ***Not only the practical and substantive correctness of the solution will be assessed, but also the optimality, quality and readability of the code written by you.***
- ***An important part of the project is the use of: inheritance, collections, interfaces or abstract classes, lambda expressions, Java Generics, additional functionalities or structures and other characteristic elements presented in the classes and lectures.***