ICAI

2h30'

JManzanas

EXAMINATION RULES

1. Download the statement and the supplied codes.
2. Prepare the exam environment:
    a. Console.
    b. File Explorer.
    c. Text editor.
    d. Browser window with the API doc:
        i. https://docs.oracle.com/en/java/javase/16/docs/api/index.html
3. Start recording with OBS: DO NOT start the exam until you are sure that the recording has started.
4. Open the text editor.
    a. Type the word POO.
    b. Copy it.
    c. Glue it twice.
5. Press the key combination Windows + Tab.
6. Start the exam.

Develop a game in the Java programming language that follows the logic described below and whose JAR is provided in Moodle along with this statement so that it can be tested.

Description of the objects in the game

The game is played on a fixed position on the Y-axis, with all objects moving along the X-axis over the width of the window, from right to left (Fig 1).

In the game there is a **main character** (white square with no filler) who can only have two positions:

1. Its starting position: x=50 y on the base Y that the student takes as a reference.
2. A higher position, -100 of the Y-axis, by performing a jump. This position will be reached by pressing the space bar (SPACE). When the space bar is released, return to your base Y position.

IMPORTANT: there will be no transition up or down. It will go directly from Y to Y-100 and vice versa.

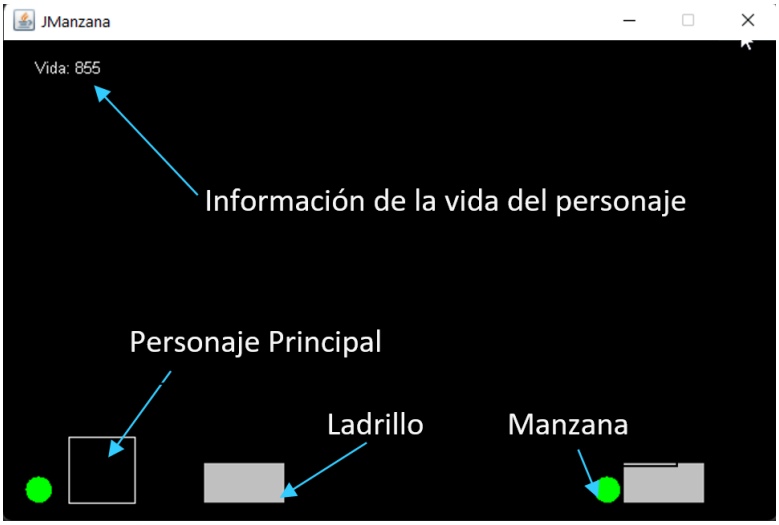

Fig 1. Window of the game and its objects

As previously described, the objects in the game (bricks and apples) will move horizontally only on the X axis, from right to left, automatically (without responding to any keystrokes) at a given speed. The way to set the speed at which each object will move is by invoking the supplied Generator class. This class will return valid random velocities (an integer in the range 1 to 4) that will be assigned to each object created. This number will determine the number of pixels each object will move each time the game asks it to. A speed of 4 will mean that we will have to move the object 4 pixels.

The initial X position of the objects (so that they are not all together and not deterministic) will also be calculated randomly (remember that they will all be fixed at a Y position) thanks to the supplied Generator class. Note how it will have to be passed a minimum and maximum value of X within which it will calculate the random value.

The default dimensions of the objects will be:

- Main character: an unfilled white square of side 50.
- Brick: a 30x60 LIGHT_GRAY rectangle with infill.
- Apple: a circle of 20 radius GREEN with filling.

Although we start from these default values for this solution and for all objects of the same type, the solution must be programmed for any possible values. The color and fill values will always be the same. Therefore, tomorrow we will be able to have big and small apples, but all of them green and filled.

Objective of the game

The object of the game will be to keep the initial life provided to the character (1000) above 0. When this life reaches 0, the game will end and display the message:

Fig 2. State of the game at the end (life <=0)

The way to modify the character's life will be as follows:

- When the character hits with an apple, his life will be increased by the life provided by that item, by default, all apples will have a value of 5.
- When the character hits a brick, his life will be decreased by the amount of damage that object causes, by default, all bricks, 20.
- If we left it at this point, the character could be always jumping (always keeping SPACE pressed) and we would never lose. Therefore, to make the game attractive we'll decide that jumping is tiring, that is, it consumes life. The loss of life in the jump will be the same as that provided by an apple, 5.

You could think that the strategy to keep the same initial life always (initial life = 1000) would be not to hit a brick and eat as many apples as you jump.

Game design

In addition to the Generator class, we also provide the Game class, which is fully developed and nothing needs to be touched, and the Board class that represents the support (canvas) where the game will be created. This class will have to be completed to achieve the desired functionality.

The Board class has an attribute called *fps* (frames per second) that allows us to calculate the time we will let pass (delay) until we update all the characters of the game with their new calculated position. The general logic of updating the movements of the objects in the game (only the apples and bricks since the character moves by keyboard events) will be as follows:

```
while(personThisLive)
{
        wait(delay)
        moveAllObjectsToYourNewPosition()
        checkPersonagePrincipalImpactsWithObjects()
        repaintBoard()
}
```

Impact calculation

This pseudocode shows how after moving the objects it will check if the bricks or apples have hit the character. The way to know if an object hits the character will be through the intersection of the rectangles that delimit the area occupied by the objects. A simple way to solve this functionality is thanks to the java.awt.Rectangle class and its intersect method.

To simplify the implementation of the game, items will not disappear or be removed after hitting the character, as you would expect. This means that, when an apple passes through the character, it will bring 5 health to the character for the duration of the intersection. Looking at the pseudocode loop, if the time it takes for the apple to pass through the character is 1000ms (1sec), for example, and there is a delay of 100ms, the impact will be repeated 10 times, giving 50 health. This leads us to believe that a slow apple is healthier than a fast one. 😊

The displacement of the objects is produced every certain time (delay) by moving a certain number of pixels (speed).

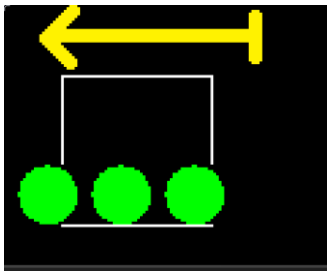

Fig 3. Repetition of impacts from the same block.

Initializing the game

The program will read from a file (data/settings.txt) the input information of the game, which will be:

- The number of apples in the game.
- The number of bricks.
- The default fps value of the game.

```
blocks: 3bricks
: 2
fps: 60
```

Fig 4. Contents of the file

The following is an example of a run when changing the file values to 30 blocks and 1 brick, for example.

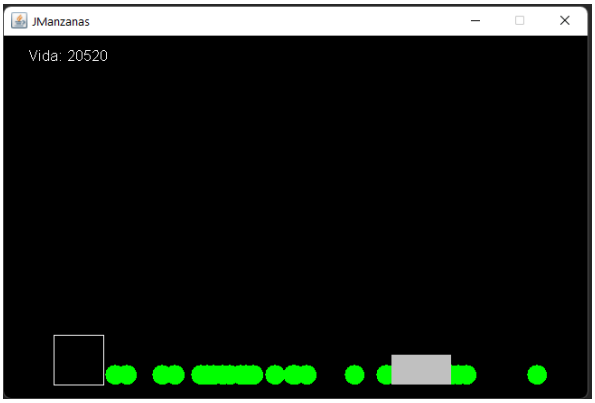

Fig 4. Contents of the file

Implementation Recommendations

It is recommended to make an iterative implementation of the game, in parts that are 100% functional. Iterations to be carried out:

1. Structure of the required classes of the object logic.
2. Character with jump functionality.
3. Moving objects.
4. Impact detection.
5. Life calculation.
6. Finish the game

An additional point will be awarded (up to a score of 11) if the following functionality is implemented after all the previous functionality has been done: displaying a list of the 5 best time results for the life of the game and an option to replay the game.

Tools that may be used in the exam:

- Text editor.
- Compiler.
- JShell: if someone wants to execute Java code in a fast way.
- It will not be possible to consult any website.
- Only the API found at the following URL: https://docs.oracle.com/en/java/javase/16/docs/api/index.html will be available.
- No IDE (Visual Studio Code, IntelliJ, Eclipse, etc.) can be used.

It is requested:

- Implement all the classes to give an answer to the statement complying with the object oriented principles seen in the course.
- Follow the package structure of the student's choice.
- You can use any JDK class and programming style seen in class: functional, for example.
- Import with * may be used.
- The exam should be recorded with OBS.