Programming Assignments

Programming Assignment 1

Programming Assignment 1
Due Date: April 1, 2019 midnight.

In this assignment, you will implement a 2D game called KoçCat using Java. The purpose of the assignment is to develop a complete Java program with emphasis on class design, inheritance, polymorphism, and organizing the related classes in packages. The graphical interface of the program will be based on Swing framework. You will practice basic graphics, anonymous classes, and event-driven programming using swing.

Please read this document carefully before starting your design and coding. This is an **individual assignment**. You **should not share your code** with others. All programming should be yours. Any coding that is not yours should be explicitly stated, otherwise, it will be considered as plagiarism. Please be aware of the KU Statement in Academic Honesty.

KoçCat Game

KoçCat is a game similar to famous PackMan. It is played on a 2D square board which is composed of NxN cells. KoçCat move constantly from one cell to the neighboring cell on the board. Cells on the board can be empty or may contain some food and/or some ghosts. The player controls KoçCat movement by changing its direction anytime using his/her keyboard. The player aim is to increase the score by eating fruits and escape from poisonous food and ghosts. If KoçCat eats a poisonous food, the player's score reduces and if it collides with a ghost then the game is over. The game description and the rules in detail is as follows:

Game rules:

1. There will be a game board with NxN squares. Each square will be of size WxW pixels.
2. When the game starts, KoçCat appears in the middle of the screen and starts to go right. A player can change its direction by using keyboard

arrow keys. There are four directions possible: East, South, West, and North. The borders of the window are boundaries. KoçCat cannot exceed these boundaries. When it reaches a boundary, it stays there until its direction has changed.

3. There will be randomly located food -fruits and poisonous food- on the board. The number of each food type will be asked to the player at the beginning of the game (see **Task 1**).
4. When KoçCat collides with food, i.e., they are in the same location (square), it means KoçCat eats the food and when eaten, food disappears.
    - When KoçCat eats a **Fruit**, its score increases.
    - When KoçCat eats a **Poison**, its score decreases. If the score becomes negative then the game is over.
5. Fruits and poisonous foods grow in time. While growing, their colour will change. Also, their effect on score changes by time. (For details, see **Task 3**)
6. There will be randomly located ghosts of three types **Casper**, **Ash**, and **Dolley** on the baord. The total number of ghosts will be asked to a player at the beginning of the game (see **Task 1**) The behaviour and division of each ghost type is describe in **Task 4**.
7. If KoçCat collides with any ghost then the game is over.
8. If a ghost will collide with another ghost or with food, then nothing happens, ghosts continue their movements.

Implementation Tasks:

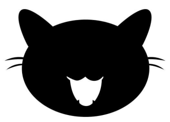
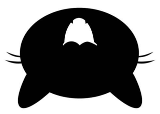
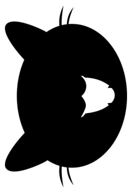
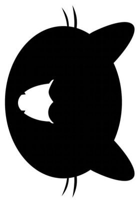
Task 1: **(10 points)**

Using Swing UI elements, ask player the number of each food type and the number ghosts which will appear on the game screen. You will initiate the game accordingly. If the user enters zero or negative number in any field you should raise an exception and game will not start.

Task 2: KoçCat **(10 points)**

Each entity that can be shown on the board (KocCat, Ghost, or Food) knows how to draw itself. You will make an abstract class (or Interface) called Drawable with a method called draw (a polymorphic method). Any entity that need to be show on the board need to Drawable.

Display the image (icon) of the cat in the middle of the screen using the draw method from the Drawable class. You will be given 4 different icons (see below figures) and you will change the cat image depending upon the direction given by the player during the animation (see **Task 5**). For instance, initially, if the KoçCat was moving towards the right side and the player press the UP arrow then you will change the image of the KoçCat with the Up.png and starts moving upward. (You will animate the KoçCat in **Task 5**)

Down.png Up.png Right.png Left.png

Task 3: Food (Fruit & Poison) (15 points)

Similarly like KoçCat, Fruit and Poison are 2 different types of food and are also drawable objects. You will draw them **randomly** at different locations (squares) on the screen by their corresponding numbers specified by the user in **Task 1**. For fruit draw a circle filled with **blue** and for poison draw a rectangle filled with **yellow**. Each time, there should be a constant number of food on the screen. Therefore, if food gets eaten, then other food should appear on the screen.

Each food type will also have the **age**. In the start, food will have **age = 1** and they grow in time. There have to be two polymorphic methods called **consumed** and **grow**. You will override these methods for Fruit and Poison. The behaviour of each type of food is as follows:

Fruit: Fruit grows in time until it decays and dies. It will decay and die when its **age = 10**. When it dies, it should disappear from the screen and new fruit will be created at a different location on the screen. When a fruit is consumed before it dies, the score will increase by the following formulae:

KoçCat.score = KoçCat.score + Food.age*5

When the age is equal to **5 minutes**, the color of the fruit will be changed from blue to green.

Poison: Poisonous food will grow forever. When it is eaten, it decreases the score of KoçCat by age*10 but it will remain on its and it will not disappear from the screen. Therefore, when KoçCat eats a living fruit what happens is that:

KoçCat.score = KoçCat.score - Food.age*10

When the age is equal to **10**, the color of the poison will be changed from **yellow** to **red**.

If the score gets negative after eating the poison, KoçCat will die and the game will be over.

The age is defined as simulation_ticks/Some_Constant.

Here are some suggested values for yoru simulation ticks:

Timer ticks/sec = 10 (i.e, 100 millisecond timer delay)
KocCat or Ghost moves W/10 pixels every tick (so in one second it moves to the next square).
Fruit gets 1 age in 10 seconds (that is Some_Constant is 100).

KocCat is considered to be in the next square if its middle pixel is in the next square.

(**Hint:** When you want to create a new Fruit on the screen, you can change the coordinate of the same food to some other place (square) and set age equal to 1.)

Task 4: The Ghosts (20 points)

In this task, you have to draw the ghosts on the screen. There are 3 different types of ghosts **Casper**, **Ash**, **Dolley** as shown in the below figure. These ghosts will move on the game screen with different strategies which are described below. In **Task 1**, if player enters number of ghosts equals to 15 then you will draw 5 ghosts of each type on the screen at random location. Ghosts are both movable and drawable. A collision with any type of ghosts means certain death for KoçCat.

Movement Strategies:

- **Casper** will move randomly, on the screen (e.g some time it will start moving upward and then left etc).
- **Ash** will move horizontally. It will always go left until it reaches the wall and it changes its direction and go always right and so on.
- **Dolley** will move in vertical direction, only up and down.

When ghosts collide with food or with each other nothing will happen (ghost will not die and similarly score will not increase or decrease).

Ash   Dolly   Casper

Task 5: The animation and proper graphics (40 points)

Here is where things get really interesting. You will have to make an animation for food, ghosts and KoçCat. Every class which need to be animated are of type Drawable. Drawable objects do have a behaviour called doAction(). You will make the animation for **KoçCat's movement**, **ghosts movement** and **for food growth**.

Your animation loop will iterate on the list of objects of type **Drawable** and will call the **doAction** and **draw** methods for each game object to animate them.

KoçCat will also respond to the keyboard events. For instance, if the player press UP key arrow KoçCat will start moving upward and so on. (You can use other Keyboard keys for the changing the direction of KoçCat instead of arrow keys.)

We expect a proper graphical interface. That is, board and drawable objects are drawn correctly, and animated correctly.

Task 6: Packaging and Programing Style(5 points)

Modularity is an important concept in software development. In order to reduce the complexity of the development, package related classes together. Why we use packages?: We which class belongs to which package, basically, to achieve information hiding. The interactions between classes usually higher with a package (related classes), and lower across packages. Classes already achieve information hiding by encapsulating data and methods and exposing only a small set of public methods. Packages serve a similar purpose: group related classes. Across packages, usually, few selected classes representing the package will have most public methods that other packages will use (We call these classes Facade or Controller, a design pattern that we will learn in Software Engineering course). In this game, you don't have many classes, its package structure is relatively simple. See the next section for more details.

IMPORTANT RULES FOR GRADING:

1. You should use polymorphism.
2. Suggestions for packages and modularity: Typically you will have software classes corresponding to important problem domain concepts such as Fruit, KocCat, Ghost, Board etc. You can put all edible objects in one package and moving objects in another package. GUI/animation related classes can be in one package. Typically, the main package will have a main class that initializes the game (i.e. creates other classes etc). You might consider putting game-control in a class. All these depend on how complex/big your classes are. Try to put tightly-related tasks in a class. Unrelated tasks should not be in the same class.
3. Try to eliminate use of **instanceof** for foods or ghosts. Overwrite necessary methods and call them without knowing food type, ghost type, etc.
4. Your methods should not be longer than 30 lines. If your method does not fit into a screen, it means, it is doing a lot of stuff, try to decompose of smaller methods and try reusing code.
5. You should properly name your packages, classes, and you should use access modifiers correctly. Your source code will be checked for programming style.
6. Your game should have only one animation loop. It will just animate super class of movable objects (ghost and the KoçCat).
7. Your animation should be smooth like the animated clock example in the PS 5 and moving car example in PS 6.
8. If you have any question, please post it on the course forum.
9. There will be no extension of the deadline. The late submissions will be penalized as follows: 1 points per hour upto 24 hours. Submissions after

24 hours will not be graded.

SUBMISSION:

1. It is important that your program compiles without an error. Programs that do not compile will not be graded. Your program should compile and run without errors.
2. You should submit your Java project package in a zip file using the course blackboard page. The name of your project file must be <your_username>_PA1.zip
3. Your source code should include a signed disclaimer that states the work is your own. Write the following comment at the beginning of your source code:

   THIS CODE IS MY OWN WORK. I DID NOT CONSULT TO ANY PROGRAM WRITTEN BY OTHER STUDENTS.
   I READ AND FOLLOWED THE GUIDELINE GIVEN IN THE PROGRAMMING ASSIGNMENT. NAME: __________

4. Failure to follow these instructions will adversely affect your grade for the assignment.

Good Luck!