Specification

For this assignment you will write a program to evaluate the effectiveness of employing specially trained dogs to protect the fairy penguins in the penguin reserve on Philip Island. This section specifies the required functionality of the program.

Background

On Phillip Island, Victoria, there is a colony of fairy penguins that inhabit the shoreline of Summerland peninsula. The penguin is known to the indigenous Boonwurrung people as Djinan Yawa-dji Goyeep.

Penguins build nests in the grasslands by the shoreline where they lay eggs and raise their chicks. The breeding season is between August and February and during this time the female penguin will lay from zero to two eggs each month. During the day the penguins fish in the ocean, returning at sunset to feed their chicks. The return of the penguins from the ocean is known as the "penguin parade".

Unfortunately, the penguin colony is vulnerable to predators and has been under threat in recent years, particularly the baby penguins left in their nests while their parents are out fishing. The main predators on land are foxes and cats, and in the sea the main predators are sharks.

The Phillip Island rangers who look after the penguin colony have an idea that they could use specially trained dogs to help protect the colony. The dogs can chase predators away and sometimes kill them. The use of such dogs has been successful in protecting other penguin colonies. A local Paw Patrol group can train dogs for this purpose; however, the dogs are expensive to train and maintain. To understand more about the effectiveness of using dogs to protect penguins the rangers would like some more information.

in this assignment you will write a computer program to simulate the penguin colony over the course of a year with different levels of Paw Patrol dog protection.

The simulation will start with an established colony and update the numbers once a month. At the end of the year the program will report the changes to the colony.

Paw Patrol simulation

The Paw Patrol for Penguins program will simulate one year of a colony of penguins. The simulation steps through each month of the year, beginning in July and ending in June, 12 months later. The user chooses whether to run the simulation with no Paw Patrol dog protection, with one Paw Patrol dog or with two Paw Patrol dogs.

The colony will start with the same number of penguin families for each simulation. At the end of a simulation the penguin colony survival statistics are reported.

Program start up

The program begins by displaying a brief welcome message.

The user is prompted for the number of dogs they would like from Paw Patrol to protect the penguins. The user can choose no dogs, one dog or two dogs.

The numbers of penguin families, foxes, cats and sharks are read in from a file *colony.txt*. These will be four comma separated integers. There is no other reading from the file during the actual running of the program.

The program creates collections of penguin families, foxes, cats and sharks. The details of these are as follows:

1. Each penguin family will have:

- a unique identification code in the following format **Pnnn** (where nnn represents a sequence of 3 digits
- two penguins (one male and one female). Each penguin will have flag indicating whether it is a male or female and a status indicating whether it is alive or not.
- a number of eggs. Each egg will have an age (in months) and a status indicating whether it is alive (i.e., has not been eaten) or not
- a number of chicks. Each chick will have an age (in months) and a status indicating whether it is alive or not.

2. At the start of the simulation, each penguin family will have a live male and a live female penguin, no eggs and no chicks.

3. Each fox will have separate counts of the penguins, chicks and eggs it has killed and a status indicating whether it is alive or not.

4. Each cat will have separate counts of penguins and chicks it has killed and a status indicating whether it is alive or not.

5. Each shark will have a count of the penguins it has killed.

Specific actions each month

Each month the following events can occur:

- In the months from August to February a female penguin from a family group can lay 0, 1, or 2 eggs, with each number of egss have equal probability.
- The ages of existing chicks and eggs is increased by 1 month.
- Eggs that were laid in the previous month (i.e., are one month old) will be due to hatch. The probability of a successful hatching is 0.70. (Hint: to calculate the probability of this event, generate a random number from 1 to 10. There is a 10% chance of each of these numbers being generated so you can nominate numbers 1-7 for a successful hatching). If a hatching is successful, the number of chicks in a family group is increased by one and the number of eggs is decreased. If a hatching is unsuccessful then the number of chicks remains the same and the number of eggs is reduced by one.
- The penguins, chicks and eggs are vulnerable to predators. Foxes can eat penguins, chicks and eggs. Cats can eat penguins and chicks, and sharks can eat penguins. The probability of each of these events each month is as follows:
    - The probability of each penguin, chick or egg being killed or eaten by a fox is 0.08. With Paw Patrol protection with one dog the probability of each event is 0.02 and with two dogs is 0.008.
    - The probability of each penguin or chick being killed or eaten by a cat is 0.04. With Paw Patrol protection with one dog the probability of each event is 0.01 and with two dogs is 0.004.
    - The probability of each penguin being killed by a shark is 0.02. Unfortunately, the Paw Patrol dogs cannot help penguins in the sea.
- If a penguin is eaten then its alive status changes to false. If the other penguin in the family group is alive then they continue looking after any eggs and chicks, but no more eggs are laid for the rest of the season. If the other penguin in the family group is not alive then unfortunately the eggs and chicks do not survive. In this case the numbers of eggs and chicks in the family group are set to 0.
- If a chick is eaten, then its alive status changes to false and the number of chicks in the family group is reduced by 1.
- If an egg is eaten, then its alive status changes to false and the number of eggs in the family group is reduced by 1.
- The cats and foxes can be killed by the Paw Patrol dogs. The probability of each of these events each month is as follows:
    - the probability of each cat or fox being killed is 0.01, if there is one Paw Patrol dog.
    - the probability of each cat or fox being killed is 0.1, if there are two Paw Patrol dogs.

After these events a summary of the status of the colony is displayed.

- Number of complete family groups
- Number of live chicks
- Number of live (uneaten) eggs

Specific actions at the completion of the simulation

At the end of the simulation a summary is displayed.

- Number of complete family groups.
- Number of live penguins
- Number of live chicks

Family group survival rate:

(total_penguin_families_with_two_parents / total_penguin_families_at_the_start)) * 100

Penguin survival rate:

(total_penguin_alive / total_penguins_at_the_start) * 100

Egg survival rate:

(total_eggs_hatched / total_eggs_laid) * 100

Chick survival rate:

(total_chicks_alive / total_chicks hatched) * 100

Overall colony survival:

(total_penguins_alive + total_chicks_alive / total_penguins_at_the_start)

(note that an overall colony survival < 1 means that the colony will decline and an overall colony survival > 1 means that the colony will grow)

A summary is written to the file *colonyFinal.txt*. The details written to the file will be the four survival rates and overall survival.

Important Notes

1. Your program must demonstrate your understanding of the object-oriented concepts and general programming constructs. Consider carefully your choice of classes, how they interact and the fields and methods of each class.

You must use appropriate data structures to store the various objects (penguins) in the program. You must make use of **both Arrays and ArrayLists** in your program. Make sure that you discuss your design with your tutor. You must document any additional assumptions you made.

2. You will be required to justify your design and the choice of any data structures used at the interview.
3. Validation of values for fields and local variables should be implemented where appropriate. You should not allow an object of a class to be set to an invalid state (i.e., put some simple validations in your mutator methods).
4. Your program should handle incorrect or invalid input and present the user with relevant error messages. No invalid input should crash the program.
5. Exception handling should be used where appropriate.

Plagiarism and collusion

Plagiarism and collusion are viewed as serious offences. All submitted code will be subjected to a similarity checker, and any submissions determined to be similar to a submission from a current or past student will be investigated further. The outcome of the decision pertaining to plagiarism and/or collusion will be determined by the faculty administration. If it is determined that plagiarism or collusion has occurred then you may be severely penalised, from losing all marks for the assignment, to facing disciplinary action at the Faculty level. To ensure compliance with this requirement, be sure to do all your coding in the Ed workspace environment and do not copy and paste any code into the workspace environment.

In cases where cheating has been confirmed, students have been severely penalised, from losing all marks for an assignment, to facing disciplinary action at the Faculty level.

POINT TO CONSIDER WHILE CODING:

Your code must include the following:

1) 12 classes (Excluding inheritance)

2) Array & Array List usage

3) inheritance & Polymorphism

4) Input, Validation & Field I/O classes

5) A complete class template must include:

i) one default constructor

ii) at least one non-default constructor (if the class has fields)

iii) accessors for each field

iv) mutators for each field

v) a display/toString method

6) Mutator Validation

7) Invalid input handling

8) Exception handling