Program 2: Inheritance/Polymorphism (50 Points)

**Objectives:**

The focus of this assignment is to better understand the concepts of inheritance and polymorphism.

**Program Background:**

**Resource management** and concepts of **triage** are useful in many disciplines and everyday life. **Games** can be created in order to train many skills.

**Program Description:**

This program will be a simple simulation game where the player is a Zoo Keeper managing a small zoo. There are two types of resources the player must manage:

1. Energy – Every day the Zoo Keeper starts with 100 energy. Performing tasks in the zoo costs energy.

2. Food – Food is in limited supply and resupplies are random.

The zoo has only four animals, each of which has their own Hunger and Happiness rating. Keeping these high leads to a better score. The animals have different behaviors and actions that affect their Happiness and react to the types of Food differently.

The game passes in days, at the beginning of each the Zoo Keeper’s Energy refreshes to 100 and a random type of Food is delivered.

The following classes are required:

* Food
* Fish
* Fruit
* Grass
* Meat
* Seeds
* Animal
* Dolphin
* Elephant
* Lion
* Ostrich
* Zoo
* ZooDriver (Provided)

**UML DIAGRAM FOR AND DISCUSSION FOR Food**

|  |
| --- |
| Food **{abstract}** |
| * exhaustionCost : int * nutrition : int * amount : int |
| <<constructor>> Food(exC : int, nut : int )  + getExhaustionCost( ) : int  + getNutrition( ) : int  + getAmount( ) : int  + receiveDelivery( )  + consume( )  + toString( ) : String |

Food will serve as the **super class** for all of the types of foods that can be given to the animals. It contains the attributes that all foods have in common.

Data Members:

exhaustionCost **–** This is the exhaustion the food will cost the Zoo Keeper to feed an animal.

nutrition – The nutritional value of the food.

amount – The amount of food that is available.

Methods:

Food( ) – Class constructor. Sets exhaustionCost and nutrition to the values that are passed. Initializes amount to 1.

getExhaustionCost( ) – Returns value of exhaustionCost.

getNutrition( ) – Returns value of nutrition.

getAmount( ) – Returns value of amount.

receiveDelivery( ) – Increase amount by 1.

consume( ) – Decrease amount by 1.

toString( ) – Returns a String of the following format (uppercase denotes data member values to be output):

Exhaustion: EXHAUSTIONCOST

Nutrition: NUTRITION

Amount: AMOUNT

Food has five **subclasses**, each differing in exhaustion cost and nutritional value.

**UML DIAGRAM FOR AND DISCUSSION FOR Grass**

|  |
| --- |
| Grass **extends** Food |
|  |
| <<constructor>> Grass( )  + toString( ) : String |

Methods:

Grass( ) – Class constructor. Calls upon **super class constructor** with exhaustion cost of 30 and nutrition of 30.

toString( ) **–** Uses **super class toString()** to return a String of the following format (uppercase denotes data member values to be output):

Grass

Exhaustion: EXHAUSTIONCOST

Nutrition: NUTRITION

Amount: AMOUNT

**UML DIAGRAM FOR AND DISCUSSION FOR Fish**

|  |
| --- |
| Fish **extends** Food |
|  |
| <<constructor>> Fish( )  + toString( ) : String |

Methods:

Fish( ) – Class constructor. Calls upon **super class constructor** with exhaustion cost of 40 and nutrition of 25.

toString( ) **–** Uses **super class toString()** to return a String of the following format (uppercase denotes data member values to be output):

Fish

Exhaustion: EXHAUSTIONCOST

Nutrition: NUTRITION

Amount: AMOUNT

**UML DIAGRAM FOR AND DISCUSSION FOR Fruit**

|  |
| --- |
| Fruit **extends** Food |
|  |
| <<constructor>> Fruit( )  + toString( ) : String |

Methods:

Fruit( ) – Class constructor. Calls upon **super class constructor** with exhaustion cost of 25 and nutrition of 10.

toString( ) **–** Uses **super class toString()** to return a String of the following format (uppercase denotes data member values to be output):

Fruit

Exhaustion: EXHAUSTIONCOST

Nutrition: NUTRITION

Amount: AMOUNT

**UML DIAGRAM FOR AND DISCUSSION FOR Meat**

|  |
| --- |
| Meat **extends** Food |
|  |
| <<constructor>> Meat( )  + toString( ) : String |

Methods:

Meat( ) – Class constructor. Calls upon **super class constructor** with exhaustion cost of 50 and nutrition of 40.

toString( ) **–** Uses **super class toString()** to return a String of the following format (uppercase denotes data member values to be output):

Meat

Exhaustion: EXHAUSTIONCOST

Nutrition: NUTRITION

Amount: AMOUNT

**UML DIAGRAM FOR AND DISCUSSION FOR Seeds**

|  |
| --- |
| Seeds **extends** Food |
|  |
| <<constructor>> Seeds( )  + toString( ) : String |

Methods:

Seeds( ) – Class constructor. Calls upon **super class constructor** with exhaustion cost of 25 and nutrition of 20.

toString( ) **–** Uses **super class toString()** to return a String of the following format (uppercase denotes data member values to be output):

Seeds

Exhaustion: EXHAUSTIONCOST

Nutrition: NUTRITION

Amount: AMOUNT

**UML DIAGRAM FOR AND DISCUSSION FOR Animal**

|  |
| --- |
| Animal **{abstract}** |
| * hunger : int * happiness : int |
| <<constructor>> Animal( )  + getHunger( ) : int  + getHappiness( ) : int  + modifyHunger( hun : int )  + modifyHappiness( hap : int)  + toString( ) : String  + eat( meal : Food ) **{abstract}**  + sleep( ) : int **{abstract}**  + clean( ) **{abstract}** |

Animal is the **super class** for all the animals in the zoo. Each Animal has a hunger and happiness value the Zoo Keeper attempts to maximize. The methods, eat(), sleep(), and clean() are **abstract** since each Animal **subclass** will treat these actions differently.

Data Members:

hunger **–** Hunger rating for the Animal, minimum of 0, maximum of 100.

happiness **–** Happiness rating for the Animal, minimum of 0, maximum of 100.

Methods:

Animal( ) – Class constructor. Sets both hunger and happiness to their maximum value of 100.

getHunger( ) – Returns value of hunger.

getHappiness( ) – Returns value of happiness.

modifyHunger( hun : int ) – Modifies hunger by the passed value, hunger must be kept at a minimum of 0 and maximum of 100. If the resulting modification would be out of bounds, set it to the closest min/max.

modifyHappiness( hap : int ) – Modifies happiness by the passed value, happiness must be kept at a minimum of 0 and maximum of 100. If the resulting modification would be out of bounds, set it to the closest min/max.

toString( ) **–**Returns a String of the following format (uppercase denotes data member values to be output):

Hunger: HUNGER

Happiness: HAPPINESS

Animal **subclasses** differ in what percentage of nutrition they receive from different foods when they eat (**rounded down to the whole number**), and how food affects their happiness. Use **instanceof** to determine specifically what type of Food they are being fed. Changes in hunger and happiness also differ when they sleep and when their cage is cleaned. Some Animals have some unique behaviors and data members that also must be managed by the Zoo Keeper.

**UML DIAGRAM FOR AND DISCUSSION FOR Dolphin**

|  |
| --- |
| Dolphin **extends** Animal |
|  |
| <<constructor>> Dolphin( )  + eat( meal : Food )  + sleep( ) : int  + clean( )  + swim( )  + toString( ) : String |

Methods:

Dolphin( ) – Class constructor. Calls upon **super class constructor**.

eat( ) – **Overrides inherited abstract method** from **super class** Animal. Changes hunger and happiness based upon type of Food and their preferences as follows:

Grass: Hunger value increased by 40% of nutritional value, happiness decreased by 10.

Fish: Hunger value increased by 100% of nutritional value, happiness increased by 10.

Fruit: Hunger value increased by 100% of nutritional value, happiness increased by 20.

Meat: Hunger value increased by 80% of nutritional value.

Seeds: Hunger value increased by 20% of nutritional value, happiness decreased by 30.

sleep( ) – Sleeping decreases hunger by 10 and happiness by 30. Returns the score for this Animal for that day, which is the sum of hunger and happiness.

clean( ) **–** Clean increases happiness by 10.

toString( ) **–** Uses **super class toString()** to return a String of the following format (uppercase denotes data member values to be output):

Dolphin

Hunger: HUNGER

Happiness: HAPPINESS

swim( ) – Swimming increases happiness by 30.

**UML DIAGRAM FOR AND DISCUSSION FOR Elephant**

|  |
| --- |
| Elephant **extends** Animal |
| * loneliness : int |
| <<constructor>> Elephant( )  + eat( meal : Food )  + sleep( ) : int  + clean( )  + toString( ) : String |

Data Member:

loneliness – Elephants get lonely without attention. If they get too lonely they will lose additional happiness when they sleep. When modifying this value it must remain within the range of 0 – 100.

Methods:

Elephant( ) – Class constructor. Calls upon **super class constructor**. Initializes loneliness to 100.

eat( ) – **Overrides inherited abstract method** from **super class** Animal. Changes hunger and happiness based upon type of Food and their preferences as follows:

Grass: Hunger value increased by 100% of nutritional value.

Fish: Hunger value increased by 40% of nutritional value, happiness decreased by 10.

Fruit: Hunger value increased by 100% of nutritional value, happiness increased by 20.

Meat: Hunger value increased by 70% of nutritional value, happiness decreased by 20.

Seeds: Hunger value increased by 50% of nutritional value.

Being fed anything increases loneliness by 10.

sleep( ) – Sleeping decreases loneliness by 20, hunger by 15, and happiness by 10. If loneliness is less than 50 happiness is decreased an additional 20. Returns the score for this Animal for that day, which is the sum of hunger and happiness.

clean( ) **–** Clean increases happiness by 30 and loneliness by 20.

toString( ) **–** Uses **super class toString()** to return a String of the following format (uppercase denotes data member values to be output):

Elephant

Hunger: HUNGER

Happiness: HAPPINESS

Loneliness: LONELINESS

**UML DIAGRAM FOR AND DISCUSSION FOR Lion**

|  |
| --- |
| Lion **extends** Animal |
| * hasBall : boolean |
| <<constructor>> Lion( )  + eat( meal : Food )  + sleep( ) : int  + clean( )  + toString( ) : String  + giveBall( ) |

Data Member:

hasBall – The Lion has a toy ball it plays with. Sometimes it plays too rough and breaks it, which may affect its happiness. This data member denotes whether or not it has its ball.

Methods:

Lion( ) – Class constructor. Calls upon **super class constructor**. Initializes hasBall to true.

eat( ) – **Overrides inherited abstract method** from **super class** Animal. Changes hunger and happiness based upon type of Food and their preferences as follows:

Grass: Hunger value increased by 50% of nutritional value, happiness decreased by 10.

Fish: Hunger value increased by 70% of nutritional value, happiness increased by 10.

Fruit: Hunger value increased by 100% of nutritional value, happiness increased by 20.

Meat: Hunger value increased by 100% of nutritional value, happiness increased by 20.

Seeds: Hunger value increased by 15% of nutritional value, happiness decreased by 10.

sleep( ) – Sleeping decreases hunger by 20 and happiness by 10. There is then a 30% chance the Lion breaks his ball. If his ball is broken happiness is decreased by an additional 20. Returns the score for this Animal for that day, which is the sum of hunger and happiness.

clean( ) **–** Clean increases happiness by 20.

toString( ) **–** Uses **super class toString()** to return a String of the following format (uppercase denotes data member values to be output):

Lion

Hunger: HUNGER

Happiness: HAPPINESS

Has Ball

**OR**

Lion

Hunger: HUNGER

Happiness: HAPPINESS

Broke Ball

giveBall( ) – Gives the Lion a new ball, sets hasBall to true.

**UML DIAGRAM FOR AND DISCUSSION FOR Ostrich**

|  |
| --- |
| Ostrich **extends** Animal |
| * eggs : int |
| <<constructor>> Ostrich( )  + eat( meal : Food )  + sleep( ) : int  + clean( )  + toString( ) : String |

Data Member:

eggs – Ostriches have a chance of laying an egg every night. If the nest gets too full without the eggs being collected its happiness will be affected.

Methods:

Ostrich( ) – Class constructor. Calls upon **super class constructor**. Initializes eggs to 0.

eat( ) – **Overrides inherited abstract method** from **super class** Animal. Changes hunger and happiness based upon type of Food and their preferences as follows:

Grass: Hunger value increased by 65% of nutritional value.

Fish: Hunger value increased by 80% of nutritional value, happiness increased by 20.

Fruit: Hunger value increased by 100% of nutritional value, happiness increased by 20.

Meat: Hunger value increased by 30% of nutritional value, happiness decreased by 15.

Seeds: Hunger value increased by 100% of nutritional value, happiness increased by 10.

sleep( ) – Sleeping decreases hunger by 10 and happiness by 20. There is then a 50% chance the Ostrich lays an egg, increasing the value of eggs by 1. If there are 3 or more eggs happiness is decreased by an additional 30. Returns the score for this Animal for that day, which is the sum of hunger and happiness.

clean( ) **–** Clean increases happiness by 20 and all eggs are collected, setting eggs to 0.

toString( ) **–** Uses **super class toString()** to return a String of the following format (uppercase denotes data member values to be output):

Ostrich

Hunger: HUNGER

Happiness: HAPPINESS

Eggs: EGGS

**UML DIAGRAM FOR AND DISCUSSION FOR Zoo**

|  |
| --- |
| Zoo |
| * cages : Animal[] * foodCrates : Food[] * keeperEnergy : int * score : int   + ELEPHANTLOCATION : int  + DOLPHINLOCATION : int  + OSTRICHLOCATION : int  + LIONLOCATION : int  + SEEDSCRATE : int  + MEATCRATE : int  + GRASSCRATE : int  + FISHCRATE : int  + FRUITCRATE : int |
| <<constructor>> Zoo( )  + feedAnimal( cage : int, crate : int) : boolean  + cleanCage( cage : int ) : boolean  + swimWithDolphin( ) : boolean  + giveLionBall( ) : boolean  + endDay( ) : int  + toString( ) : String |

The Zoo class is what contains all the Animals, Food, and manages the players Keeper Energy and Score. It contains methods to perform Zoo Keeper actions and move to the next day. Uppercase data members in UML denote the data member is **final**. These should be initialized to 0 – 3 and 0 – 4 as listed in the UML. These can be used to easily remember the indices of the instances of the **subclasses**.

Data Members:

cages – Array of all Animals in the zoo.

foodCrates – Array of all the Food available.

keeperEnergy – Energy the Zoo Keeper has for the current day.

score – Total player score.

Methods:

Zoo( ) – Class constructor. Creates a 4 element Animal array for the cages data member, and a 5 element Food array for the foodCrates array. Creates instances of the **subclasses** of Animal and Food and stores them in the array in the locations denoted by the **final** data members (cages[0] will have an instance of Elephant, etc).

Sets keeperEnergy to 100 and score to 0.

feedAnimal( cage : int, crate : int) – Request to feed the Animal in the passed cage index from the passed food crate index. If not possible due to exhaustion cost or food not being available this method returns false. If the action is possible will feed the animal the food and adjust keeperEnergy and food (using consume method) amount and returns true.

cleanCage( cage : int ) – Request to clean the cage at the passed index. This action costs 30 keeperEnergy. If not possible due to exhaustion cost this method returns false. If the action is possible, cleans the cage and reduces keeperEnergy by 30 and returns true.

swimWithDolphin( ) – Requests to swim with the Dolphin. This action costs 40 keeperEnergy. If not possible due to exhaustion cost this method returns false. If the action is possible, swims with Dolphin and reduces keeperEnergy by 40 and returns true.

giveLionBall( ) \_ Requests to give Lion a new ball. This action costs 20 keeperEnergy. If not possible due to exhaustion cost this method returns false. If the action is possible, Lion is given a new ball and reduces keeperEnergy by 20 and returns true.

endDay( ) – Method called when player decides to end the current day. Sends all Animals to sleep and updates score (the sleep() method returns the score for that Animal for that day. Randomly increases the amount of one of the Foods by 1 (using receiveDelivery() method). KeeperEnergy is reset to 100 for the following day. Returns the total score.

toString( ) **–** Returns a String representing the current state of the Zoo; Animals, Food, keeperEnergy, and score in the following format: (initial game values shown):

ANIMALS

Elephant

Hunger: 100

Happiness: 100

Loneliness: 100

Dolphin

Hunger: 100

Happiness: 100

Ostrich

Hunger: 100

Happiness: 100

Eggs: 0

Lion

Hunger: 100

Happiness: 100

Has Ball

FOOD CRATES

Seeds

Exhaustion: 25

Nutrition: 20

Amount: 1

Meat

Exhaustion: 50

Nutrition: 40

Amount: 1

Grass

Exhaustion: 30

Nutrition: 30

Amount: 1

Fish

Exhaustion: 40

Nutrition: 25

Amount: 1

Fruit

Exhaustion: 25

Nutrition: 10

Amount: 1

ENERGY: 100

SCORE: 0

**ZooDriver and Other Notes**

The ZooDriver is provided to you. This will create an instance of Zoo and run the game. The player chooses for how many days the would like to play. Menus are provided for the player to make actions. Look through and understand the ZooDriver, it will help to understand how the Zoo class is being used. Do not make any changes to this.

Order of actions is important! Follow method descriptions exactly, including the order in which actions occur. For example, in the Lion’s sleep() method it states to determine if it breaks its ball, *then* check if ball is broken to further decrease happiness. This means the Lion will receive the happiness penalty on the same night it breaks its ball. If this order is not followed your results will differ and points may be lost.

**Helpful methods:**

For determining random events you can either use the Random class or the random method in the Math class.

Using the Math class’ min and max methods will help in keeping values in range.

**Package Structure**

Programs with multiple classes benefit from utilizing a package structure for organization and potential reusability.

All of your source code (your .java files) should be in their package directory. Only those .java files that do not declare a package should be in your source directory.

All of the Food classes (Food, Grass, Fish, Fruit, Meat, Seeds) must be declared to belong to the Zoo.Foods package.

All of the Animal classes (Animal, Dolphin, Elephant, Lion, Ostrich) must be declared to belong to the Zoo.Animals package.

Since these classes will belong to packages and not in the submission directory, they will need to be imported in other classes that use them.

The Zoo class is not declared to be in any package and will therefore be directly accessible in the submission directory.

**Assignment Questions**

Provide the answers to these questions in your submission directory within a file called Assignment2Questions.

1. What are the advantages of utilizing the concepts of inheritance and polymorphism in this program?

2. Design and write (create a separate .java file) your own Animal subclass with something that sets it apart from other animals and describe it in your Assignment2Questions file. What would you need to do in order to integrate it into the game?