Mateo Valdés Otero

PROYECTO FINAL APO II

Requerimientos funcionales:

RF1: Mostrar el tiempo trascurrido

RF2: Guardar el estado del programa

RF3: Cargar el estado del programa

RF4: Calcular todo el dinero obtenido por los animales

RF5: Mostrar el dinero que ha sido obtenido

RF6: Comprar nuevos animales

RF7: Ver la cantidad de cada tipo de animal que han sido comprado

Especificación de requerimientos funcionales:

Nombre	RF1: Mostrar el tiempo trascurrido
Resumen	El sistema permite mostrar el tiempo que ha trascurrido desde que se inició la aplicación
Entrada	-
Salida	El tiempo trascurrido

Nombre	RF2: Guardar el estado del programa
Resumen	El sistema permite guardar el estado del programa en archivos en el computador. Esto incluye: • El tiempo trascurrido • El dinero recaudado • La cantidad de cada tipo animal
Entrada	El estado del juego y el tiempo
Salida	El estado del programa ha sido guardado exitosamente

Nombre	RF3: Cargar el estado del programa
Resumen	El sistema permite guardar el estado del programa desde archivos en el computador. Esto incluye: • El tiempo trascurrido

	• El dinero recaudado • La cantidad de cada tipo animal
Entrada	Archivos de texto con la información
Salida	El estado del programa ha sido cargado exitosamente

Nombre	RF4: Calcular todo el dinero obtenido por los animales
Resumen	El sistema permite calcular todo el dinero obtenido por los animales. • Las truchas, las tilapias y las vacas obtienen dinero al sumar su valor • Los pollos y los patos multiplican el dinero por el valor de su multiplicador
Entrada	El dinero inicial
Salida	El dinero obtenido por los animales

Nombre	RF5: Mostrar el dinero que ha sido obtenido
Resumen	El sistema permite mostrarle al usuario todo el dinero que ha sido obtenido en el trascurso del juego
Entrada	-
Salida	El dinero obtenido por el usuario

Nombre	RF6: Comprar nuevos animales
Resumen	El sistema le permite al usuario comprar nuevos animales siempre y cuando tenga el dinero necesario. Los animales son: • Tilapia • Trucha • Pollos • Patos • Vacas • Perros
Entrada	El animal que se desea comprar
Salida	El animal ha sido comprado exitosamente

Nombre	RF7: Ver la cantidad de cada tipo de animal que han sido comprado
Resumen	El sistema permite visualizar cuantos animales de cada tipo han sido comprados
Entrada	-
Salida	La cantidad de cada tipo de animal que ha sido comprada

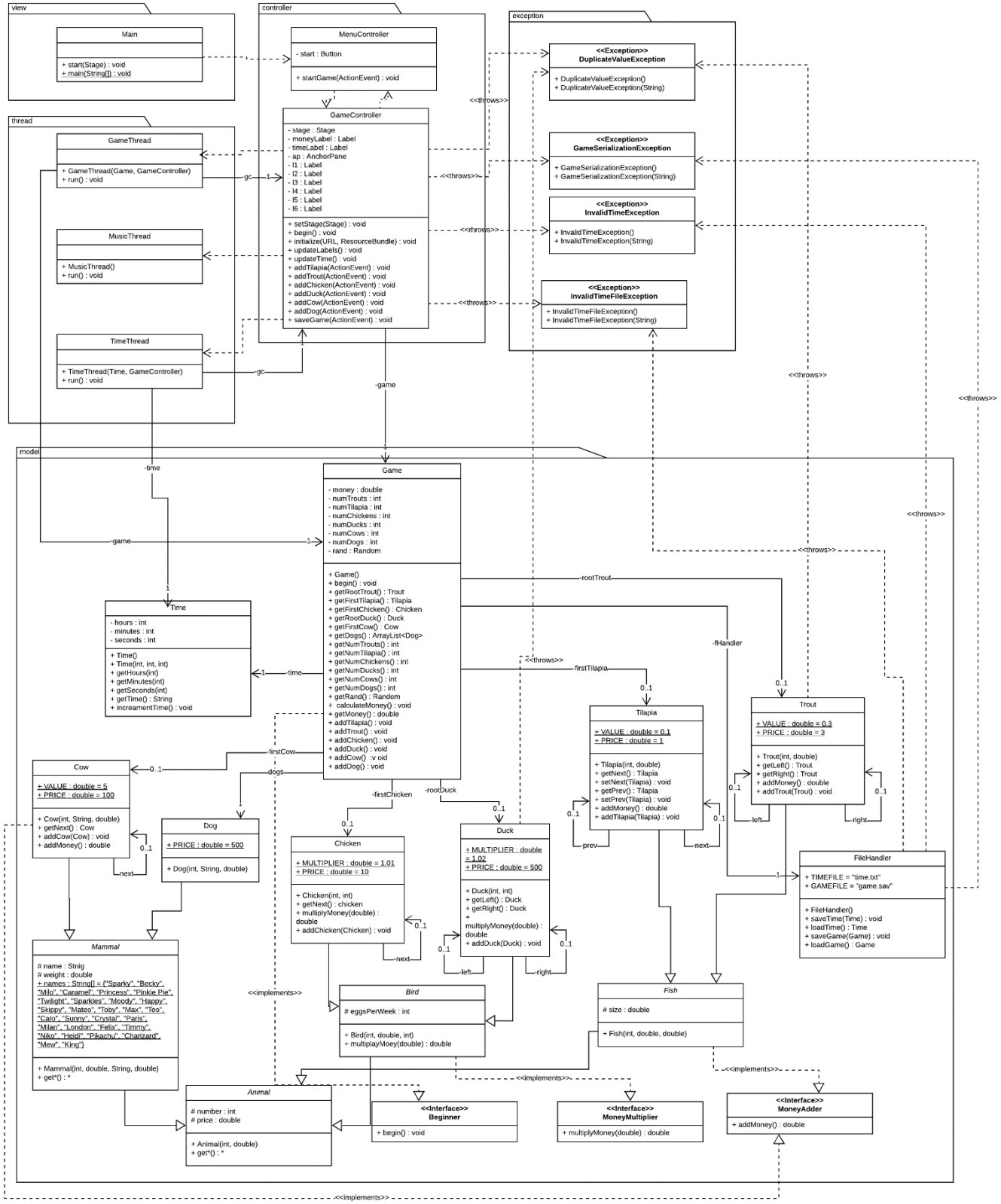

view
Main
+ start(Stage) : void
+ main(String[]) : void
controller
MenuController
- start : Button
+ startGame(ActionEvent) : void
GameController
- stage : Stage
- moneyLabel : Label
- timeLabel : Label
- ap : AnchorPane
- l1 : Label
- l2 : Label
- l3 : Label
- l4 : Label
- l5 : Label
- l6 : Label
+ setStage(Stage) : void
+ begin() : void
+ initialize(URL, ResourceBundle) : void
+ updateLabels() : void
+ updateTime() : void
+ addTilapia(ActionEvent) : void
+ addTrout(ActionEvent) : void
+ addChicken(ActionEvent) : void
+ addDuck(ActionEvent) : void
+ addCow(ActionEvent) : void
+ addDog(ActionEvent) : void
+ saveGame(ActionEvent) : void
exception
<<Exception>> DuplicateValueException
+ DuplicateValueException()
+ DuplicateValueException(String)
<<Exception>> GameSerializationException
+ GameSerializationException()
+ GameSerializationException(String)
<<Exception>> InvalidTimeException
+ InvalidTimeException()
+ InvalidTimeException(String)
<<Exception>> InvalidTimeFileException
+ InvalidTimeFileException()
+ InvalidTimeFileException(String)
thread
GameThread
+ GameThread(Game, GameController)
+ run() : void
MusicThread
+ MusicThread()
+ run() : void
TimeThread
+ TimeThread(Time, GameController)
+ run() : void
<<throws>>
gc
-game
model
Game
- money : double
- numTrouts : int
- numTilapia : int
- numChickens : int
- numDucks : int
- numCows : int
- numDogs : int
- rand : Random
+ Game()
+ begin() : void
+ getRootTrout() : Trout
+ getFirstTilapia() : Tilapia
+ getFirstChicken() : Chicken
+ getRootDuck() : Duck
+ getFirstCow() : Cow
+ getDogs() : ArrayList<Dog>
+ getNumTrouts() : int
+ getNumTilapia() : int
+ getNumChickens() : int
+ getNumDucks() : int
+ getNumCows() : int
+ getNumDogs() : int
+ getRand() : Random
+ calculateMoney() : void
+ getMoney() : double
+ addTilapia() : void
+ addTrout() : void
+ addChicken() : void
+ addDuck() : void
+ addCow() :v oid
+ addDog() : void
Time
- hours : int
- minutes : int
- seconds : int
+ Time()
+ Time(int, int, int)
+ getHours(int)
+ getMinutes(int)
+ getSeconds(int)
+ getTime() : String
+ increamentTime() : void
-time
-rootTrout
-fHandler
-firstTilapia
-firstCow
-dogs
-firstChicken
-rootDuck
Cow
+ VALUE : double = 5
+ PRICE : double = 100
+ Cow(int, String, double)
+ getNext() : Cow
+ addCow(Cow) : void
+ addMoney() : double
Dog
+ PRICE : double = 500
+ Dog(int, String, double)
Chicken
+ MULTIPLIER : double = 1.01
+ PRICE : double = 10
+ Chicken(int, int)
+ getNext() : chicken
+ multiplyMoney(double) : double
+ addChicken(Chicken) : void
Duck
+ MULTIPLIER : double = 1.02
+ PRICE : double = 500
+ Duck(int, int)
+ getLeft() : Duck
+ getRight() : Duck
+ multiplyMoney(double) : double
+ addDuck(Duck) : void
Tilapia
+ VALUE : double = 0.1
+ PRICE : double = 1
+ Tilapia(int, double)
+ getNext() : Tilapia
+ setNext(Tilapia) : void
+ getPrev() : Tilapia
+ setPrev(Tilapia) : void
+ addMoney() : double
+ addTilapia(Tilapia) : void
Trout
+ VALUE : double = 0.3
+ PRICE : double = 3
+ Trout(int, double)
+ getLeft() : Trout
+ getRight() : Trout
+ addMoney() : double
+ addTrout(Trout) : void
FileHandler
+ TIMEFILE = "time.txt"
+ GAMEFILE = "game.sav"
+ FileHandler()
+ saveTime(Time) : void
+ loadTime() : Time
+ saveGame(Game) : void
+ loadGame() : Game
Mammal
name : Stnig
weight : double
+ names : String[] = {"Sparky", "Becky", "Milo", "Caramel", "Princess", "Pinkie Pie", "Twilight", "Sparkles", "Moody", "Happy", "Skippy", "Mateo", "Toby", "Max", "Teo", "Cato", "Sunny", "Crystal", "Paris", "Milan", "London", "Felix", "Timmy", "Niko", "Heidi", "Pikachu", "Charizard", "Mew", "King"}
+ Mammal(int, double, String, double)
+ get*() : *
Bird
eggsPerWeek : int
+ Bird(int, double, int)
+ multiplayMoey(double) : double
Fish
size : double
+ Fish(int, double, double)
Animal
number : int
price : double
+ Animal(int, double)
+ get*() : *
<<Interface>> Beginner
+ begin() : void
<<Interface>> MoneyMultiplier
+ multiplyMoney(double) : double
<<Interface>> MoneyAdder
+ addMoney() : double
<<implements>>
left
right
next
prev
0..1
1

Diagrama de objetos:

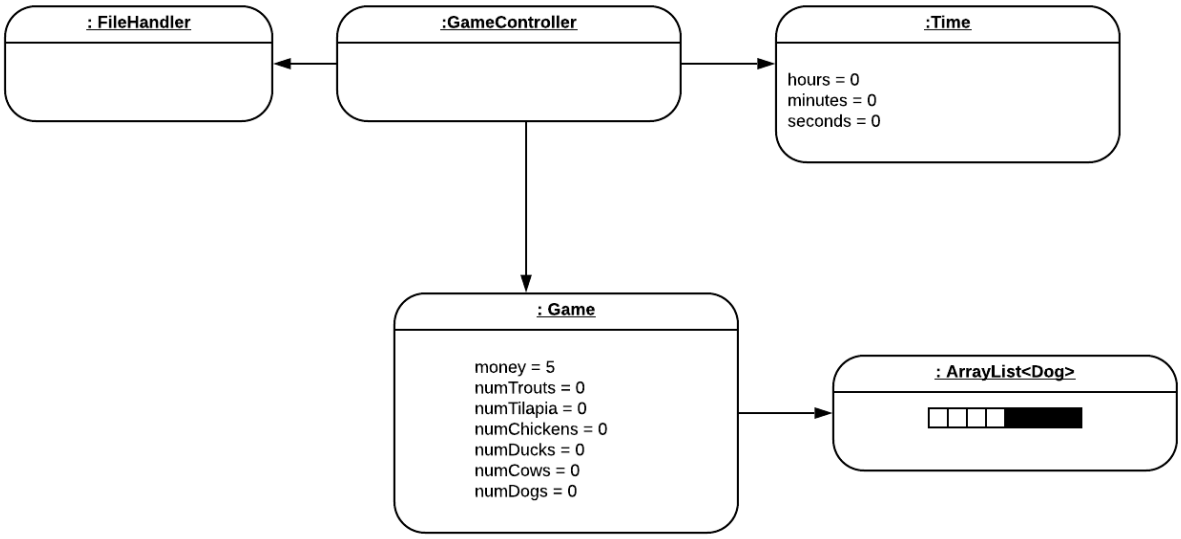
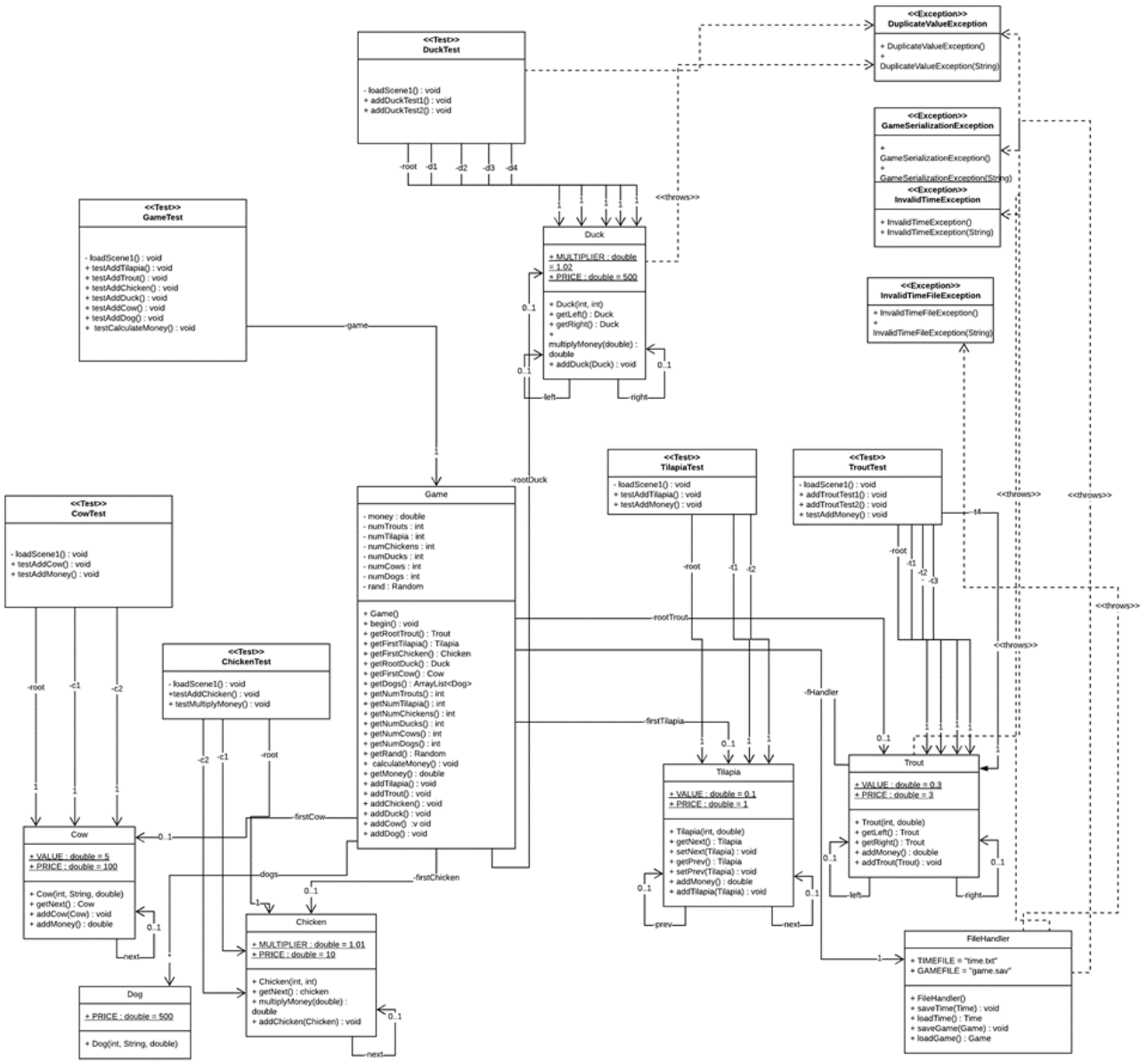


Diagrama de las pruebas:



Diseño de las pruebas:

Clase: Chicken

Escenarios:

LoadScene1(): Se crea un escenario con tres instancias de la clase Chicken. Estos pollos son representados por una lista enlazada sencilla y el primer pollo (root) es el primer elemento de esta lista enlazada. Las tres instancias tienen las siguientes características:

- Chicken #1: Su número es 123, su precio es de 10 y pone 5 huevos por semana (root)

- Chicken #2: Su número es 321, su precio es de 10 y pone 7 huevos por semana (c1)
- Chicken #3: Su número es 213, su precio es de 10 y pone 9 huevos por semana (c2)

Método	Escenario	Valores de entrada	Resultado
addChicken()	loadScene1()	-	Exitoso si los pollos se agregan a la lista enlazada que empieza con el primer pollo de una manera ordenada y sin presentar problemas
multiplyMoney()	loadScene1()	-	Exitoso si al agregar pollos a la lista enlazada el valor del método incrementa exponencialmente con el valor de los pollos como debería

Clase: Cow

Escenarios:

LoadScene1(): Se crea un escenario con tres instancias de la clase Cow. Estas vacas son representados por una lista enlazada sencilla y la primera vaca (root) es el primer elemento de esta lista enlazada. Las tres instancias tienen las siguientes características:

- Cow #1: Su número es 123, su nombre es Mateo y pesa 50.0 (root)
- Cow #2: Su número es 321, su nombre es Johan y pesa 51.0 (c1)
- Cow #3: Su número es 213, su nombre es Ariza y pesa 52.0 (c2)

Método	Escenario	Valores de entrada	Resultado
addCow()	loadScene1()	-	Exitoso si las vacas se agregan a la lista enlazada que empieza con la primera vaca de una manera ordenada y sin presentar problemas
addMoney()	loadScene1()	-	Exitoso si al agregar vacas a la lista enlazada el valor del método incrementa por el valor de cada vaca como debería

Clase: Duck

Escenarios:

LoadScene1(): Se crea un escenario con cuatro instancias de la clase Duck. Estos patos son representados por un árbol binario y el primer pato (root) es el primer nodo de este árbol binario. Las cuatro instancias tienen las siguientes características:

- Duck #1: Su número es 123 y pone 4 huevos a la semana (root)
- Duck #2: Su número es 124 y pone 5 huevos a la semana (d1)
- Duck #3: Su número es 100 y pone 8 huevos a la semana (d2)
- Duck #4: Su número es 50 y pone 12 huevos a la semana (d3)
- Duck #5: Su número es 123 y pone 4 huevos a la semana (d4)

Método	Escenario	Valores de entrada	Resultado
addDuck()	loadScene1()	-	Exitoso si se lanza una excepción cuando debería (cuando hay dos patos con un número igual, en este caso root y d4)
addDuck()	loadScene1()		Exitoso si los patos con un número menor se agregar a la izquierda del nodo y los patos con un número mayor se agregan a la derecha del nodo
multiplyMoney()	loadScene1()	-	Exitoso si al agregar un nuevo pato el dinero aumenta exponencialmente por el valor que debería

Clase: Game

Escenarios:

LoadScene1(): Se crea una instancia de la clase Game, se inicializan todas las variables numéricas a cero y se le asigna un valor a la variable de dinero de 500. Además, se inicializa un ArrayList de Dog.

Método	Escenario	Valores de entrada	Resultado
addChicken()	loadScene1()	-	Exitoso si después de llamar al método el Chicken que lo contiene el objeto (game) ya no es nulo
addTilapia()	loadScene1()	-	Exitoso si después de llamar al método la Tilapia que la contiene el objeto (game) ya no es nula
addTrout()	loadScene1()	-	Exitoso si después de llamar al método la Trout que lo contiene el objeto (game) ya no es nula

addDuck()	loadScene1()	-	Exitoso si después de llamar al método el Duck que lo contiene el objeto (game) ya no es nulo
addCow()	loadScene1()	-	Exitoso si después de llamar al método el Cow que lo contiene el objeto (game) ya no es nulo
addDog()	loadScene1()	-	Exitoso si después de llamar al método el ArrayList de Dog ya tiene su primer elemento

Clase: Tilapia

Escenarios:

LoadScene1(): Se crea un escenario con tres instancias de la clase Tilapia. Estas tilapias son representadas por una lista doblemente enlazada y la primera tilapia (root) es el primer elemento de esta lista enlazada. Las tres instancias tienen las siguientes características:

- Tilapia #1: Su número es 123, su precio es de 1 y su tamaño es de 5 (root)
- Tilapia #2: Su número es 321, su precio es de 1 y su tamaño es de 7 (t1)
- Tilapia #3: Su número es 213, su precio es de 1 y su tamaño es de 9 (t2)

Método	Escenario	Valores de entrada	Resultado
addTilapia()	loadScene1()	-	Exitoso si las tilapias se agregan a la lista doblemente enlazada que empieza con la primera vaca de una manera ordenada y sin presentar problemas
addMoney()	loadScene1()	-	Exitoso si al agregar tilapias a la lista enlazada el valor del método incrementa por el valor de cada tilapia como debería

Clase: Trout

Escenarios:

LoadScene1(): Se crea un escenario con cuatro instancias de la clase Trout. Estas truchas son representadas por un árbol binario y el primer pato (root) es el primer nodo de este árbol binario. Las cuatro instancias tienen las siguientes características:

- Trout #1: Su número es 123 y su tamaño es de 4 (root)
- Trout #2: Su número es 124 y su tamaño es de 5 (t1)
- Trout #3: Su número es 100 y su tamaño es de 8 (t2)

Trout #4: Su número es 50 y su tamaño es de 12 (t3)

- Trout #5: Su número es 123 y su tamaño es de 4 (t4)

Método	Escenario	Valores de entrada	Resultado
addTrout()	loadScene1()	-	Exitoso si se lanza una excepción cuando debería (cuando hay dos truchas con un número igual, en este caso root y t4)
addTrout()	loadScene1()		Exitoso si las truchas con un número menor se agregar a la izquierda del nodo y las truchas con un número mayor se agregan a la derecha del nodo
addMoney()	loadScene1()	-	Exitoso si al agregar un nuevo truchas el dinero se suma y aumenta por el valor que debería

LINK DE GITHUB:

https://github.com/mavaldot/JohanFarm