CS 1120 – Computer Science II (with Java)

Instructor: *Leszek T. Lilien*, Western Michigan University

Lab TA: *Fola Olagbemi*

*Neil Macklin-Camel*

**SOFTWARE LIFE CYCLE REPORT – FOR LAB ASSIGNMENT** ***4***

# PHASE 1: SPECIFICATION (“What do we build?”)

Write five classes, **Race**, **Animal**, **Ostrich**, **Turtle**, **AnimalRace**, and implement one interface, **IRace**, where the class **Race** implements **IRace**. These five classes and one interface can:

1. Implement the IRace interface in the Race class.
2. Read the binary file given that contains an animal type of ostrich or turtle, the name of an animal, and the max speed that that animal can go.
3. Create a randomly generated character array that is a racetrack composed of four different terrains, the four terrains being: open plains, forest, desert or lake.
4. A race between the all the animals read from the binary file, whilst tracking the animals current position in the race.
5. Display the animal’s type being an ostrich or turtle, the name of that specific animal, the animal’s serial number, and then the animal’s current position on the racetrack once an animal or animals have won.
6. Display the winning animal’s name(s).

|  |  |
| --- | --- |
| Ostrich (Chacha) 1: | . O O O # ~ # # ~ ~ O ~ O # ~ . . # <~> | |
| Turtle (Sissy) 2: | . O O O # ~ # # ~ ~ O ~ O # ~ . . # ~ <|> |
| Turtle (Tata) 3: | . O O O # ~ # # ~ ~ O ~ O # ~ . . # ~ <|> |
| Ostrich (Chooby) 4: | . O O O # ~ # # ~ ~ O ~ O # ~ . . # ~ <|> |
| Ostrich (Lightning) 5: | . O O O # ~ # # ~ ~ O ~ O # ~ . . # <~> | |
| Ostrich (Speedo) 6: | . O O O # ~ # # ~ ~ O ~ O # ~ . . # <~> | |
| Turtle (Steady) 7: | . O O O # ~ # # ~ ~ O ~ O # ~ . . # ~ <|> |
| Ostrich (Osha) 8: | . O O O # ~ # # ~ ~ O ~ O # ~ . . <#> ~ | |
| Turtle (Swinger) 9: | . O O O # ~ # # ~ ~ O ~ O # ~ . . # ~ <|> |
| Turtle (Coach) 10: | . O O O # ~ # # ~ ~ O ~ O # ~ . . # ~ <|> |

Winner(s):

Sissy

Tata

Chooby

Steady

Swinger

Coach

Test the class by performing the following steps:

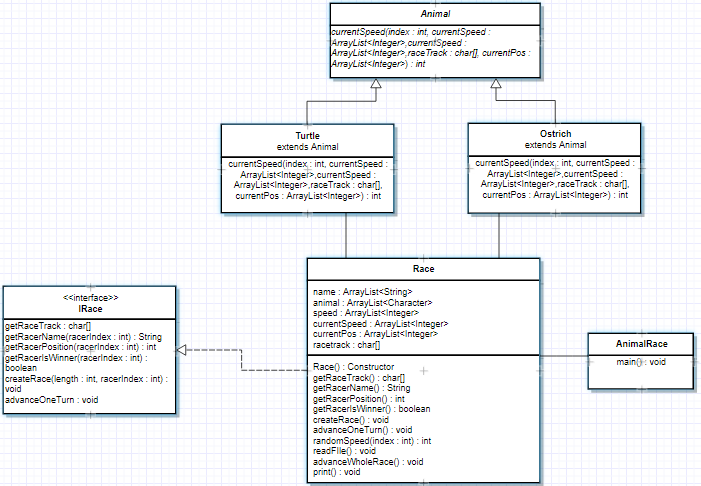
1. Use the **Race** class given.
2. Read the binary file.
3. Create the randomly generated racetrack designated for the 4 different terrains.
4. Advance through each terrain for each animal, while calling either the **Ostrich** class or the **Turtle** class to see the how fast that animal can move according to that specific animal.
5. Continue the race between animals until an animal or animals have won the race, by reaching the end of the racetrack.
6. Use the **AnimalRace** class given.
7. Create a Race object.
8. Call the methods necessary to test the program.

# PHASE 2: DESIGN

**2.1 Modules and Their Basic Structure**

My program will have five modules.

1. Module 1: Class **Race** will contain
2. Implementation of the **IRace** interface.
3. Method getRacerIsWinner that takes an index of a racer, to tell whether this specific racer won the race, which entails the racer’s current position being at the finish line.
4. Method createRace that takes length of the racetrack and the number of racers. Then creates a racetrack that is filled with randomly generated characters that can be any of the four terrains: open plains, forest, desert, or a lake, then make the last spot in the race a finish line.
5. Method advanceOneTurn that circles through all of the racers once and calls either the **Ostrich** class or the **Turtle** class depending on which animal the method is currently on. This method will set an array to store the exact position of each animal throughout the advancing of it.
6. A method that read the binary file and stores each of the animal’s type, name, and max Speed into arrays.
7. A method that takes the max speed read from the binary file and generates a random number between one and that max speed of that specific animal and does this for all animals, then returns an array with this data.
8. A method that calls the advanceOneTurn method and calls this method until an animal or animals have won the race.
9. A method that displays the type of animal, the name of the animal, the serial number of the animal, and the racetrack position of that animal for all animals, then displays the name of the winning animal(s).
10. Module 2: Class **Animal** will contain:
11. An abstract method that takes an index, the speed or amount of movement points all the animals have, the racetrack, and the position on the racetrack of all the animals.
12. Module 3: Class **Ostrich** will contain:
13. Extends the abstract class: **Animal**.
14. The method that is from the **Animal** class.
15. This method will find if any of the animals being sent in are a winner, and then this method will check the following: if the next racetrack position is the same as the current, the current speed of the animal will be subtracted one and then returned, if the next racetrack position is a forest, the speed of the animal will be subtracted three and then returned, if the next racetrack position is a lake, the speed of the animal will be subtracted two and then returned, and lastly, if the next racetrack position is either a desert or an open plain, the speed of the animal will have one added the it and then returned.
16. Module 4: Class **Turtle** will contain:
17. Extends the abstract class: **Animal**.
18. The method that is from the **Animal** class.
19. This method will find if any of the animals being sent in are a winner, and then this method will check the following: if the next racetrack position is a desert, an open plain, or a forest, then the speed of the animal will be subtracted one and then returned and if the next racetrack position is the same as the current racetrack position and the next racetrack position is not a lake, then the speed of the animal will be subtracted one and then returned.
20. If the next racetrack position for a turtle is ever a lake, then the speed of the animal will not be changed.
21. Module 5: Class **AnimalRace** will contain:
22. Instantiate a Race object.
23. Call all necessary methods from the **Race** class to test the program.

**UML diagram:**

**2.2 Pseudocode for the Modules**

**2.2.1 Pseudocode for** Race implements IRace

1a) **Race** Pseudocode refinement #1

// This class implements the IRace interface.

// Private Fields: name (ArrayList<String>), animal (ArrayList<Character>),

// speed (ArrayList<Integer>), currentSpeed (<Integer>), currentPos (ArrayList<Integer>),

// racetrack (character array

// Constructor: Accepts one parameter: the number of racers (int). Initialize currentPos,

// fill currentPos with zeros to account for every racer starting at position zero

// call the readFile method, and copy the speed ArrayList to currentSpeed.

// Methods: --------------------------------------------------------------------------------------------------------------

// Method getRacerIsWinner: Take one parameter: racerIndex (int). Check if the current

// position of the racerIndex has won. Return true if the racer has won and return false if

// the racer has not won.

// Method createRace: Takes two parameters: length (int) and numRacers (int). Initialize

// racetrack with the size of the parameter length. Use a Random object to generate

// a number between 0 and 3. Do this length amount of times and if a zero is generated, // then an open plain terrain is added to racetrack, if a one is generated, then a forest

// terrain is added to racetrack, if a two is generated, then a desert is added to racetrack, // and lastly, if a 3 is generated, then a lake is added to racetrack. The last spot of

// racetrack will always be a finish line.

// Method advanceOneTurn: Instantiate an Ostrich object. Instantiate a Turtle object.

// Set currentSpeed to output of the randomSpeed method for each racer. Choose if

// each racer will move their position on the racetrack by calling the currentSpeed method // in either the **Ostrich** class or the **Turtle** class depending on which animal the racer is.

// Method randomSpeed: Takes one parameter that is the counter (int) that is used in the

// advanceOneTurn method. Creates a Random object. Generates a random integer from // one to max speed of the racer. Returns the random integer to the advanceOneTurn

// method.

// Method readFile: Initialize animal, initialize speed, and initialize name. Create a

// FileInputStream object and a DataInputStream object. Read a character into the

// animal ArrayList, an integer into the speed ArrayList, and a String into the name

// ArrayList until the end of the binary file.

// Method advanceWholeRace: Using a loop, call the advanceOneTurn method and then loop

// through the getRacerIsWinner method until the getRacerIsWinner method returns true, // which means that a racer has won.

// Method print: Print the type of animals, the names of animals, the serial numbers of the

// animals, and the position of the animals on the racetrack along with the entire

// racetrack for each racer. Then print each the winning racer’s names.

1b) **Race** Pseudocode refinement #2

// This class implements the IRace interface.

// Private Fields: name (ArrayList<String>), animal (ArrayList<Character>),

// speed (ArrayList<Integer>), currentSpeed (<Integer>), currentPos (ArrayList<Integer>),

// racetrack (character array

// Constructor: Accepts one parameter: the number of racers (int). Initialize currentPos,

// fill currentPos with zeros to account for every racer starting at position zero

// call the readFile method, and copy the speed ArrayList to currentSpeed.

// Methods: --------------------------------------------------------------------------------------------------------------

// Method getRacerIsWinner: Take one parameter: racerIndex (int). Check if the current

// position of the racerIndex has won. Return true if the racer has won and return false if

// the racer has not won.

// If a racer has won, then set their current position on the racetrack to the spot of the // finish line for printing uses.

// Method createRace: Takes two parameters: length (int) and numRacers (int). Initialize

// racetrack with the size of the parameter length. Use a Random object to generate

// a number between 0 and 3. Do this length amount of times and if a zero is generated, // then an open plain terrain is added to racetrack, if a one is generated, then a forest

// terrain is added to racetrack, if a two is generated, then a desert is added to racetrack, // and lastly, if a 3 is generated, then a lake is added to racetrack. The last spot of

// racetrack will always be a finish line.

// Method advanceOneTurn: Instantiate an Ostrich object. Instantiate a Turtle object.

// Set currentSpeed to output of the randomSpeed method for each racer. Choose if

// each racer will move their position on the racetrack by calling the currentSpeed method // in either the **Ostrich** class or the **Turtle** class depending on which animal the racer is.

// After choosing if a racer will move their position on the racetrack, this method check // if what the currentSpeed method from either class is negative, if it is, then

// the current position will not be moved on the racetrack and will stay the same.

// Method randomSpeed: Takes one parameter that is the counter (int) that is used in the

// advanceOneTurn method. Creates a Random object. Generates a random integer from // one to max speed of the racer. Returns the random integer to the advanceOneTurn

// method.

// Method readFile: Initialize animal, initialize speed, and initialize name. Create a

// FileInputStream object and a DataInputStream object. Read a character into the

// animal ArrayList, an integer into the speed ArrayList, and a String into the name

// ArrayList until the end of the binary file.

// Method advanceWholeRace: Using a loop, call the advanceOneTurn method and then loop

// through the getRacerIsWinner method until the getRacerIsWinner method returns true, // which means that a racer has won.

// When this method finds a winner then this method ends.

// Method print: Print the type of animals, the names of animals, the serial numbers of the

// animals, and the position of the animals on the racetrack along with the entire

// racetrack for each racer. Then print each the winning racer’s names.

**2.2.2. Pseudocode for** Animal

2a) **Animal** Pseudocode refinement #1

// This class is an abstract class.

// An abstract method that takes an index (int), the speed or amount of movement points all

// the animals have (ArrayList<Integer>), the racetrack (character array), and the position // on the racetrack of all the animals(ArrayList<Integer>).

2b) **Animal** Pseudocode refinement #2

// This class is an abstract class.

// This class is extended by the **Ostrich** class and the **Turtle** class.

// An abstract method that takes an index (int), the speed or amount of movement points all

// the animals have (ArrayList<Integer>), the racetrack (character array), and the position // on the racetrack of all the animals(ArrayList<Integer>).

**2.2.3.** **Pseudocode for** Ostrich

3a) **Ostrich** Pseudocode refinement #1

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

3b) **Ostrich** Pseudocode refinement #2

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

// If the returned value is negative the advanceOneTurn method in the **Race** class will

// check for this. If the next terrain matches the current one, then currentSpeed of // the racer will lose one. If the next terrain is a forest, then the currentSpeed of // the racer will lose three. If the next terrain is a lake, then the currentSpeed of

// the racer will lose two. If the next terrain is an open plain, then the

// currentSpeed of the racer will gain one. Lastly, if the next terrain is a desert,

// then the currentSpeed of the racer will gain one.

**2.2.4. Pseudocode for** Turtle

4a) **Turtle** Pseudocode refinement #1

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

4b) **Turtle** Pseudocode refinement #2

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

// If the returned value is negative the advanceOneTurn method in the **Race** class will

// check for this. If the next terrain matches the current one and the next terrain // is not a lake, then currentSpeed of the racer will lose one. If the next terrain is a // forest, then the currentSpeed of the racer will lose one. If the next terrain is an // open plain, then the currentSpeed of the racer will lose one. Lastly, if the next // terrain is a desert, then the currentSpeed of the racer will lose one. The terrain // of a lake for the **Turtle** class will not be checked because it costs zero

// movement points to move into a lake.

**2.2.5. Pseudocode for** AnimalRace

5a) **AnimalRace** Pseudocode refinement #1

// Method main: Instantiate a Race object. Call createRace, advanceWholeRace, and print.

**PHASE 3: RISK ANALYSIS (“What can go wrong, and how bad can it be?”)**

No risks (to timetable, cost, human health, etc.) are identified by me.

# PHASE 4: VERIFICATION (“Are the algorithms correct?”)

The algorithm has only one execution path (a sequential execution). Correctness of the path has been verified by me by analyzing its steps, and their completeness w.r.t. the Specification.

# PHASE 5: CODING

## 5a) Code Refinement #1 (class structure with pseudocode only; pseudocode is used as comments)

**File Race.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Race class

**Public class Race implements IRace{**

// This class implements the IRace interface.

// Private Fields: name (ArrayList<String>), animal (ArrayList<Character>),

// speed (ArrayList<Integer>), currentSpeed (<Integer>), currentPos (ArrayList<Integer>),

// racetrack (character array

// Constructor: Accepts one parameter: the number of racers (int). Initialize currentPos,

// fill currentPos with zeros to account for every racer starting at position zero

// call the readFile method, and copy the speed ArrayList to currentSpeed.

// Methods: --------------------------------------------------------------------------------------------------------------

// Method getRacerIsWinner: Take one parameter: racerIndex (int). Check if the current

// position of the racerIndex has won. Return true if the racer has won and return false if

// the racer has not won.

// If a racer has won, then set their current position on the racetrack to the spot of the // finish line for printing uses.

// Method createRace: Takes two parameters: length (int) and numRacers (int). Initialize

// racetrack with the size of the parameter length. Use a Random object to generate

// a number between 0 and 3. Do this length amount of times and if a zero is generated, // then an open plain terrain is added to racetrack, if a one is generated, then a forest

// terrain is added to racetrack, if a two is generated, then a desert is added to racetrack, // and lastly, if a 3 is generated, then a lake is added to racetrack. The last spot of

// racetrack will always be a finish line.

// Method advanceOneTurn: Instantiate an Ostrich object. Instantiate a Turtle object.

// Set currentSpeed to output of the randomSpeed method for each racer. Choose if

// each racer will move their position on the racetrack by calling the currentSpeed method // in either the **Ostrich** class or the **Turtle** class depending on which animal the racer is.

// After choosing if a racer will move their position on the racetrack, this method check // if what the currentSpeed method from either class is negative, if it is, then

// the current position will not be moved on the racetrack and will stay the same.

// Method randomSpeed: Takes one parameter that is the counter (int) that is used in the

// advanceOneTurn method. Creates a Random object. Generates a random integer from // one to max speed of the racer. Returns the random integer to the advanceOneTurn

// method.

// Method readFile: Initialize animal, initialize speed, and initialize name. Create a

// FileInputStream object and a DataInputStream object. Read a character into the

// animal ArrayList, an integer into the speed ArrayList, and a String into the name

// ArrayList until the end of the binary file.

// Method advanceWholeRace: Using a loop, call the advanceOneTurn method and then loop

// through the getRacerIsWinner method until the getRacerIsWinner method returns true, // which means that a racer has won.

// When this method finds a winner then this method ends.

// Method print: Print the type of animals, the names of animals, the serial numbers of the

// animals, and the position of the animals on the racetrack along with the entire

// racetrack for each racer. Then print each the winning racer’s names.

**}** // end of Race class

**File Animal.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Animal class

**Public abstract class Animal {**

// This class is an abstract class.

// This class is extended by the **Ostrich** class and the **Turtle** class.

// An abstract method that takes an index (int), the speed or amount of movement points all

// the animals have (ArrayList<Integer>), the racetrack (character array), and the position // on the racetrack of all the animals(ArrayList<Integer>).

**}** // end of Animal class

**File Ostrich.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Ostrich class

**Public class Ostrich extends Animal{**

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

// If the returned value is negative the advanceOneTurn method in the **Race** class will

// check for this. If the next terrain matches the current one, then currentSpeed of // the racer will lose one. If the next terrain is a forest, then the currentSpeed of // the racer will lose three. If the next terrain is a lake, then the currentSpeed of

// the racer will lose two. If the next terrain is an open plain, then the

// currentSpeed of the racer will gain one. Lastly, if the next terrain is a desert,

// then the currentSpeed of the racer will gain one.

**}** // end of Ostrich class

**File Turtle.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Turtle class

**Public class Turtle extends Animal{**

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

// If the returned value is negative the advanceOneTurn method in the **Race** class will

// check for this. If the next terrain matches the current one and the next terrain // is not a lake, then currentSpeed of the racer will lose one. If the next terrain is a // forest, then the currentSpeed of the racer will lose one. If the next terrain is an // open plain, then the currentSpeed of the racer will lose one. Lastly, if the next // terrain is a desert, then the currentSpeed of the racer will lose one. The terrain // of a lake for the **Turtle** class will not be checked because it costs zero

// movement points to move into a lake.

**}** // end of Turtle class

**File AnimalRace.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the AnimalRace class

**Public class AnimalRace {**

// Method main: Instantiate a Race object. Call createRace, advanceWholeRace, and print.

**}** // end of AnimalRace class

## 5b) Code Refinement #2 (still incomplete program: class and constructor/method structure with pseudocode only; pseudocode is used as comments)

**File Race.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Race class

**Public class Race implements IRace{**

// This class implements the IRace interface.

// Private Fields: name (ArrayList<String>), animal (ArrayList<Character>),

// speed (ArrayList<Integer>), currentSpeed (<Integer>), currentPos (ArrayList<Integer>),

// racetrack (character array

// Constructor: Accepts one parameter: the number of racers (int). Initialize currentPos,

// fill currentPos with zeros to account for every racer starting at position zero

// call the readFile method, and copy the speed ArrayList to currentSpeed.

// Methods: --------------------------------------------------------------------------------------------------------------

// Method getRacerIsWinner: Take one parameter: racerIndex (int). Check if the current

// position of the racerIndex has won. Return true if the racer has won and return false if

// the racer has not won.

**public boolean getRacerIsWinner(/\*int racerIndex\*/) {**

// If a racer has won, then set their current position on the racetrack to the spot of the // finish line for printing uses.

**}** //end of gestRacerIsWinner method

// Method createRace: Initialize racetrack with the size of the parameter length. Use a Random // object to generate a number between 0 and 3. Do this length amount of times and if a // zero is generated, then an open plain terrain is added to racetrack, if a one is

// generated, then a forest terrain is added to racetrack, if a two is generated, then a

// desert is added to racetrack, and lastly, if a 3 is generated, then a lake is added to

// racetrack. The last spot of racetrack will always be a finish line.

**public void createRace(/\*int length, int numRacers\*/){**

**}** // end of createRace method

// Method advanceOneTurn: Instantiate an Ostrich object. Instantiate a Turtle object.

// Set currentSpeed to output of the randomSpeed method for each racer. Choose if

// each racer will move their position on the racetrack by calling the currentSpeed method // in either the **Ostrich** class or the **Turtle** class depending on which animal the racer is.

**public void advanceOneTurn() {**

// After choosing if a racer will move their position on the racetrack, this method check // if what the currentSpeed method from either class is negative, if it is, then

// the current position will not be moved on the racetrack and will stay the same.

**}** // end of advanceOneTurn method

// Method randomSpeed: Takes one parameter that is the counter (int) that is used in the

// advanceOneTurn method. Creates a Random object. Generates a random integer from // one to max speed of the racer. Returns the random integer to the advanceOneTurn

// method.

**public int randomSpeed(/\*int i\*/) {**

**}** // end of randomSpeed method

// Method readFile: Initialize the fields: animal, speed, and name. Create a

// FileInputStream object and a DataInputStream object. Read a character into the

// animal ArrayList, an integer into the speed ArrayList, and a String into the name

// ArrayList until the end of the binary file.

**public void readFile() {**

**}** // end of readFile method

// Method advanceWholeRace: Using a loop, call the advanceOneTurn method and then loop

// through the getRacerIsWinner method until the getRacerIsWinner method returns true, // which means that a racer has won.

**public void advanceWholeRace() {**

// When this method finds a winner then this method ends.

**}** // end of adcanceWholeRace method

// Method print: Print the type of animals, the names of animals, the serial numbers of the

// animals, and the position of the animals on the racetrack along with the entire

// racetrack for each racer. Then print each the winning racer’s names.

**public void print() {**

**}** // end of print method

**}** // end of Race class

**File Animal.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Animal class

**Public abstract class Animal {**

// This class is an abstract class.

// This class is extended by the **Ostrich** class and the **Turtle** class.

// An abstract method that takes an index (int), the speed or amount of movement points all

// the animals have (ArrayList<Integer>), the racetrack (character array), and the position // on the racetrack of all the animals(ArrayList<Integer>).

**public abstract int currentSpeed(/\*int index, ArrayList<Integer> currentSpeed, char[] raceTrack, ArrayList<Integer> currentPos\*/);**

**}** // end of Animal class

**File Ostrich.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Ostrich class

**Public class Ostrich extends Animal{**

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

**public int currentSpeed(/\*int index, ArrayList<Integer> currentSpeed, char[] raceTrack, ArrayList<Integer> currentPos\*/) {**

// If the returned value is negative the advanceOneTurn method in the **Race** class will

// check for this.

// If the next terrain matches the current one, then currentSpeed of the racer will lose // one.

// If the next terrain is a forest, then the currentSpeed of the racer will lose three.

// If the next terrain is a lake, then the currentSpeed of the racer will lose two.

// If the next terrain is an open plain, then the currentSpeed of the racer will gain one.

// If the next terrain is a desert, then the currentSpeed of the racer will gain one.

**}** //end of currentSpeed method

**}** // end of Ostrich class

**File Turtle.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Turtle class

**Public class Turtle extends Animal{**

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

**public int currentSpeed(/\*int index, ArrayList<Integer> currentSpeed, char[] raceTrack, ArrayList<Integer> currentPos\*/) {**

// If the returned value is negative the advanceOneTurn method in the **Race** class will

// check for this.

// If the next terrain matches the current one and the next terrain is not a lake, then

// currentSpeed of the racer will lose one.

// If the next terrain is a forest, then the currentSpeed of the racer will lose one.

// If the next terrain is an open plain, then the currentSpeed of the racer will lose one.

// Lastly, if the next terrain is a desert, then the currentSpeed of the racer will lose one.

// The terrain of a lake for the **Turtle** class will not be checked because it costs zero

// movement points to move into a lake.

**}** // end of currentSpeed method

**}** // end of Turtle class

**File AnimalRace.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the AnimalRace class

**Public class AnimalRace {**

// Method main:

**Public static void main(String[] args) {**

// Instantiate a Race object. Call createRace, advanceWholeRace, and print.

**}** // end of main method

**}** // end of AnimalRace class

## 5c) Code Refinement #3 (complete program--with complete fields/properties, code for constructor/methods)

**File IRace.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

**public** **interface** IRace {

//returns an array of characters, representing the racetrack

**char**[] **getRacetrack**();

//Returns the name of the racer with the given index

**String** **getRacerName**(**int** racerIndex);

//Returns the position of the racer with the given index

**int** **getRacerPosition**(**int** racerIndex);

//Returns whether or not the racer with the given index is a winner (crossed // the finish line)

**boolean** **getRacerIsWinner**(**int** racerIndex);

//Creates the racetrack of a given length and instantiates the specified

// number of racers

//Racetrack should be a random racetrack using the four different types of terrain, and ending // with the finish line.

**void** **createRace** (**int** length, **int** numRacers)) **throws** **IOException**;

//Causes each racer to take one turn, moving a number of spaces based on their

// movement speed and the terrain

**void** **advanceOneTurn**();

**}** // end of IRace

**File Race.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Race class

**Public class Race implements IRace{**

// This class implements the IRace interface.

// Private Fields: name (ArrayList<String>), animal (ArrayList<Character>),

// speed (ArrayList<Integer>), currentSpeed (<Integer>), currentPos (ArrayList<Integer>),

// racetrack (character array

// Constructor: Accepts one parameter: the number of racers (int). Initialize currentPos,

// fill currentPos with zeros to account for every racer starting at position zero

// call the readFile method, and copy the speed ArrayList to currentSpeed.

**public** **Race** (**int** numRacers) {

currentPos = **new** ArrayList<Integer>();

**for** (**int** **i**=0;i<numRacers;i++) {

currentPos.add(0);

}

readFile();

currentSpeed = **new** ArrayList<Integer>(speed);

**}** // end Race constructor

// Methods: --------------------------------------------------------------------------------------------------------------

// Method getRacerIsWinner: Take one parameter: racerIndex (int). Check if the current

// position of the racerIndex has won. Return true if the racer has won and return false if

// the racer has not won.

**public boolean getRacerIsWinner(int racerIndex) {**

**if** (currentPos.get(racerIndex) >= racetrack.length-1) {

// If a racer has won, then set their current position on the racetrack to the spot of the // finish line for printing uses.

currentPos.set(racerIndex, racetrack.length);

**return** **true**;

} **else** {

**return** **false**;

}

**}** //end of getRacerIsWinner method

// Method createRace: Initialize racetrack with the size of the parameter length. Use a Random // object to generate a number between 0 and 3. Do this length amount of times and if a // zero is generated, then an open plain terrain is added to racetrack, if a one is

// generated, then a forest terrain is added to racetrack, if a two is generated, then a

// desert is added to racetrack, and lastly, if a 3 is generated, then a lake is added to

// racetrack. The last spot of racetrack will always be a finish line.

**public void createRace(int length, int numRacers) throws IOException {**

racetrack = **new** **char**[length];

**Random** **r** = **new** Random();

**int** **rand**;

**for** (**int** **i** = 0; i < length - 1; i++) {

rand = r.nextInt(4);

**switch** (rand) {

**case** 0:

racetrack[i] = '.';

**break**;//open plains

**case** 1:

racetrack[i] = '#';

**break**;//forest

**case** 2:

racetrack[i] = 'O';

**break**;//desert

**case** 3:

racetrack[i] = '~';

**break**;//lake

}

}

racetrack[length-1] = '|';//finish line

**}** // end of createRace method

// Method advanceOneTurn: Instantiate an Ostrich object. Instantiate a Turtle object.

// Set currentSpeed to output of the randomSpeed method for each racer. Choose if

// each racer will move their position on the racetrack by calling the currentSpeed method // in either the **Ostrich** class or the **Turtle** class depending on which animal the racer is.

**public void advanceOneTurn() {**

**Ostrich** **callO** = **new** Ostrich();

**Turtle** **callT** = **new** Turtle();

**for** (**int** **i** = 0; i < 10; i++) {

currentSpeed.set(i, randomSpeed(i));

**if** (getRacerIsWinner(i) == **false**) {

**if** (animal.get(i) == 'O') {

// After choosing if a racer will move their position on the

// racetrack, this method check if what the currentSpeed // method from either class is negative, if it is, then the

// current position will not be moved on the racetrack and // will stay the same.

**if** (callO.currentSpeed(i, currentSpeed, racetrack, currentPos) > -1) {

currentPos.set(i, currentPos.get(i)+1);

}

} **else** {

**if** (callT.currentSpeed(i, currentSpeed, racetrack, currentPos) > -1) {

currentPos.set(i, currentPos.get(i) + 1);

}

}

}

}

**}** // end of advanceOneTurn method

// Method randomSpeed: Takes one parameter that is the counter (int) that is used in the

// advanceOneTurn method. Creates a Random object. Generates a random integer from // one to max speed of the racer. Returns the random integer to the advanceOneTurn

// method.

**public int randomSpeed(int i) {**

**Random** **r** = **new** Random();

**int** **random** = speed.get(i);

**int** **rand** = r.nextInt(random) + 1;

**return** rand;

**}** // end of randomSpeed method

// Method readFile: Initialize the fields: animal, speed, and name. Create a

// FileInputStream object and a DataInputStream object. Read a character into the

// animal ArrayList, an integer into the speed ArrayList, and a String into the name

// ArrayList until the end of the binary file.

**public void readFile() {**

animal = **new** ArrayList<Character>();

speed = **new** ArrayList<Integer>();

name = **new** ArrayList<String>();

**boolean** **end** = **false**;

**try** {

**FileInputStream** **fStream** = **new** FileInputStream("Animal.dat");

**DataInputStream** **input** = **new** DataInputStream(fStream);

**while**(!end) {

animal.add(input.readChar());

speed.add(input.readInt());

name.add(input.readUTF());

}

input.close();

} **catch** (**EOFException** **e**) {

end = **true**;

} **catch** (**IOException** **e**) {

e.printStackTrace();

}

**}** // end of readFile method

// Method advanceWholeRace: Using a loop, call the advanceOneTurn method and then loop

// through the getRacerIsWinner method until the getRacerIsWinner method returns true, // which means that a racer has won.

**public void advanceWholeRace() {**

**boolean** **stop** = **false**;

**while** (!stop) {

advanceOneTurn();

**for** (**int** **i** = 0; i < 10; i++) {

**if** (getRacerIsWinner(i) == **true**) {

// When this method finds a winner then this method ends.

stop = **true**;

}

}

}

**}** // end of advanceWholeRace method

// Method print: Print the type of animals, the names of animals, the serial numbers of the

// animals, and the position of the animals on the racetrack along with the entire

// racetrack for each racer. Then print each the winning racer’s names.

**public void print() {**

//printing of the animal, name, serial number, and racetrack position

**for** (**int** **i** = 0; i < 10; i++) {

**if** (animal.get(i) == 'O') {

**System**.***out***.print("Ostrich ");

} **else** {

**System**.***out***.print("Turtle ");

}

**System**.***out***.printf("(" + getRacerName(i) + ") %d" + ": ",i+1);

**int** **count** = 1;

**for** (**int** **j** = 0; j < racetrack.length; j++) {

**if** (getRacerPosition(i) == count) {

**System**.***out***.printf("<%c>",racetrack[j]);

count++;

} **else** {

**System**.***out***.printf(" %c ",racetrack[j]);

count++;

}

}

**System**.***out***.println();

}

//printing of the winner(s)

**System**.***out***.println("\nWinner(s):");

**for** (**int** **i** = 0; i < 10; i++) {

**if** (getRacerIsWinner(i) == **true**) {

**System**.***out***.println(getRacerName(i));

}

}

**}** // end of print method

**}** // end of Race class

**File Animal.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Animal class

**Public abstract class Animal {**

// This class is an abstract class.

// This class is extended by the **Ostrich** class and the **Turtle** class.

// An abstract method that takes an index (int), the speed or amount of movement points all

// the animals have (ArrayList<Integer>), the racetrack (character array), and the position // on the racetrack of all the animals(ArrayList<Integer>).

**public abstract int currentSpeed(int index, ArrayList<Integer> currentSpeed, char[] raceTrack, ArrayList<Integer> currentPos);**

**}** // end of Animal class

**File Ostrich.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Ostrich class

**Public class Ostrich extends Animal{**

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

**public int currentSpeed(int index, ArrayList<Integer> currentSpeed, char[] raceTrack, ArrayList<Integer> currentPos) {**

**int** **returned** = currentSpeed.get(index);

**if** (currentPos.get(index) == 19) {

returned = -1;

} **else** {

// If the next terrain matches the current one, then currentSpeed of the racer

// will lose one.

**if** (raceTrack[currentPos.get(index)+1] == raceTrack[index]) {

returned = currentSpeed.get(index) - 1;

// If the next terrain is a forest, then the currentSpeed of the racer will lose

// three.

} **else** **if** (raceTrack[currentPos.get(index)+1] == '#') {

returned = currentSpeed.get(index)-3;

// If the next terrain is a lake, then the currentSpeed of the racer will lose two.

} **else** **if** (raceTrack[currentPos.get(index)+1] == '~') {

returned = currentSpeed.get(index)-2;

// If the next terrain is an open plain, then the currentSpeed of the racer will

// gain one.

} **else** **if** (raceTrack[currentPos.get(index)+1] == '.') {

returned = currentSpeed.get(index)+1;

// If the next terrain is a desert, then the currentSpeed of the racer will gain

// one.

} **else** **if** (raceTrack[currentPos.get(index)+1] == 'O') {

returned = currentSpeed.get(index)+1;

}

}

// If the returned value is negative the advanceOneTurn method in the **Race**

//class will check for this.

**return** returned;

**}** //end of currentSpeed method

**}** // end of Ostrich class

**File Turtle.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the Turtle class

**Public class Turtle extends Animal{**

// This class extends the Animal class.

// Method currentSpeed: Takes four parameters: index (int), currentSpeed (ArrayList<Integer>), // racetrack (character array), currentPos (ArrayList<Integer>). Compare racetrack terrains // to see which terrain is next, how many currentSpeed movementpoints a racer will lose // or gain, then return the currentSpeed of the racer.

**public int currentSpeed(int index, ArrayList<Integer> currentSpeed, char[] raceTrack, ArrayList<Integer> currentPos) {**

**int** **returned** = currentSpeed.get(index);

**if** (currentPos.get(index) == 19) {

returned = -1;

} **else** {

// If the next terrain matches the current one and the next terrain is not a lake, // then currentSpeed of the racer will lose one.

**if** (raceTrack[currentPos.get(index)+1] == raceTrack[currentPos.get(index)] && raceTrack[currentPos.get(index)+1] != '~') {

returned = currentSpeed.get(index) - 1;

// If the next terrain is a forest, then the currentSpeed of the racer will lose

// one.

} **else** **if** (raceTrack[currentPos.get(index)+1] == '#') {

returned = currentSpeed.get(index) - 1;

// If the next terrain is an open plain, then the currentSpeed of the racer will

// lose one.

} **else** **if** (raceTrack[currentPos.get(index)+1] == '.') {

returned = currentSpeed.get(index) - 1;

// If the next terrain is a desert, then the currentSpeed of the racer will lose

// one.

} **else** **if** (raceTrack[currentPos.get(index)+1] == 'O') {

returned = currentSpeed.get(index) - 1;

}

// The terrain of a lake for the **Turtle** class will not be checked because it costs // zero movement points to move into a lake.

}

// If the returned value is negative the advanceOneTurn method in the **Race** class will

// check for this.

**return** returned;

**}** // end of currentSpeed method

**}** // end of Turtle class

**File AnimalRace.java:**

**Package edu.wmich.CS1120.LA4.NMacklinCamel;**

// This is the AnimalRace class

**Public class AnimalRace {**

// Method main:

**Public static void main(String[] args) {**

**Race** **r** = **new** Race(10);

r.createRace(20,1);

r.advanceWholeRace();

r.print();

**}** // end of main method

**}** // end of AnimalRace class

# PHASE 6: TESTING (“Did we build it correctly?”)

This program has only a single execution path. Therefore, a single run tests it completely. Multiple tests produces the following output of the program:

**First test:**

Ostrich (Chacha) 1: O O O . # . # # . . ~ # O . O O . <O> . |

Turtle (Sissy) 2: O O O . # . # # . . ~ # O . O O . O . <|>

Turtle (Tata) 3: O O O . # . # # . . ~ # O . O O . O . <|>

Ostrich (Chooby) 4: O O O . # . # # . . ~ # <O> . O O . O . |

Ostrich (Lightning) 5: O O O . # . # # . . ~ # O . O O . O . <|>

Ostrich (Speedo) 6: O O O . # . # # . . ~ # O . O O . O . <|>

Turtle (Steady) 7: O O O . # . # # . . ~ # O . O O . O . <|>

Ostrich (Osha) 8: O O O . # . # # . . ~ # O . O O . <O> . |

Turtle (Swinger) 9: O O O . # . # # . . ~ # O . O O . O . <|>

Turtle (Coach) 10: O O O . # . # # . . ~ # O . O O . O . <|>

Winner(s):

Sissy

Tata

Lightning

Speedo

Steady

Swinger

Coach

**Second test:**

Ostrich (Chacha) 1: # ~ # # O # . # # # . ~ O ~ O ~ # . . <|>

Turtle (Sissy) 2: # ~ # # O # . # # # . ~ O ~ O ~ # . . <|>

Turtle (Tata) 3: # ~ # # O # . # # # . ~ O ~ O ~ # . . <|>

Ostrich (Chooby) 4: # ~ # # O # . # # # . ~ O ~ O ~ # . . <|>

Ostrich (Lightning) 5: # ~ # # O # . # # # . ~ O ~ O <~> # . . |

Ostrich (Speedo) 6: # ~ # # O # . # # # . ~ O ~ O ~ # . . <|>

Turtle (Steady) 7: # ~ # # O # . # # # . ~ O ~ O ~ # . . <|>

Ostrich (Osha) 8: # ~ # # O # . # # # . ~ O ~ O ~ # . . <|>

Turtle (Swinger) 9: # ~ # # O # . # # # . ~ O ~ O ~ # . . <|>

Turtle (Coach) 10: # ~ # # O # . # # # . ~ O ~ O ~ # . . <|>

Winner(s):

Chacha

Sissy

Tata

Chooby

Speedo

Steady

Osha

Swinger

Coach

**Third test:**

Ostrich (Chacha) 1: # O . # # # . O O . # # O # ~ ~ . # . <|>

Turtle (Sissy) 2: # O . # # # . O O . # # O # ~ ~ . # . <|>

Turtle (Tata) 3: # O . # # # . O O . # # O # ~ ~ . # . <|>

Ostrich (Chooby) 4: # O . # # # . O O . # # O # ~ ~ . # . <|>

Ostrich (Lightning) 5: # O . # # # . O O . # # O # ~ ~ . # . <|>

Ostrich (Speedo) 6: # O . # # # . O O . # # O # ~ ~ . # . <|>

Turtle (Steady) 7: # O . # # # . O O . # # O # ~ ~ . # . <|>

Ostrich (Osha) 8: # O . # # # . O O . # # O # ~ <~> . # . |

Turtle (Swinger) 9: # O . # # # . O O . # # O # ~ ~ . # . <|>

Turtle (Coach) 10: # O . # # # . O O . # # O # ~ ~ . # . <|>

Winner(s):

Chacha

Sissy

Tata

Chooby

Lightning

Speedo

Steady

Swinger

Coach

I verified that this output satisfies program requirements.

**PHASE 7: REFINING THE PROGRAM (“Add bells and whistles to the program”)**

No refinements are needed. In this program, I have already included all required features.

# PHASE 8: PRODUCTION

I prepared a copy of the entire program for Lab TA’s evaluation, as specified by the TA. Then, I sent electronically the copy to the Lab TA.

# PHASE 9: MAINTENANCE

To fully benefit from the program evaluation feedback received from the Lab TA, I will perform program maintenance. This means that I should use all TAs feedback to improve my program.