Alexandria University
Faculty of Engineering
Specialized Scientific Programs
Spring 2019

Computer and Communication
CS272:Programming 2
Assignment 3
Assigned: 12/04/2019
Due: 02/05/2019

River Crossing Puzzle

Objectives:

Upon completion of this assignment, you will be able to:

- Design an object-oriented model for a game.
- Draw a UML class diagram that represents your model.
- Apply different design patterns to your model.
- Apply the OOP concepts of inheritance and polymorphism to your design.
- Getting familiar with XML parsers.

1) Problem Description:

River Crossing IQ Logic Puzzles require you to get all of the characters(crossers) across a river in a small boat. The boat can cross the river many times to get anyone across. In our game each character(crosser) has some properties, the most important are:

- **Weight**: This property represents the weight of a crosser.
- **Eating Rank**: This property represents the eating rank of a crosser. Crossers with higher eating ranks can eat crossers with lower eating ranks.

We have three major categories of crossers in our game(farmers, animals, plants). Farmers don't eat any crosser, can't be eaten by any crosser and can raft the boat. Animals and plants cannot raft the boat. There are two major types of animals (carnivorous and herbivorous). Carnivorous can only eat herbivorous but can't eat plants. Herbivorous can only eat plants. This game has many challenging stories, for this assignment you have to implement at least 2 stories.

Story1:

"A farmer wants to cross a river and take with him a carnivorous, a herbivorous and plant."

Rules:

1. The farmer is the only one who can sail the boat. He can only take one passenger, in addition to himself.
2. You can not leave any two crossers on the same bank if they can harm(eat) each other

How can the farmer get across the river with all the 2 animals and the plant without any losses?

Story2:

"Four farmers and their animal need to cross a river in a small boat. The weights of the farmers are 90 kg, 80 kg,60 kg and 40 kg respectively, and the weight of the animal is 20 kg."

Rules:

1. The boat cannot bear a load heavier than 100 kg.
2. All farmers can raft, while the animal cannot.

How can they all get to the other side with their animal?

2) Tasks:

1) Implement the **ICrosser** interface to represent the entities which will cross the river e.g. Persons, Animals and Plants, and make sure you have logical assumptions. **You must create at least two animals of each category**.
2) Implement the **IGameController** interface to be the game engine
3) Each Level must implement the interface **ICrossingStrategy**
4) You must use the **bold** design patterns at least, for example:
    - **Singleton**
    - **Command**
    - **Strategy**
    - **MVC**
    - Observer
    - Factory
    - Snapshot (Memento)
    - Iterator
5) Design user friendly GUI for this game, which uses only the controller that implements the **IGameController** interface.

6) This user interface must have:
    - Levels menu at the beginning of the game, enables the user to choose which story to play
    - Label showing score for the current game, the score is the number of boat crosses
    - Two riverbanks with characters on them, addition to the boat
    - User can select any character to ride the boat
    - Undo & Redo for all user sails (i.e. complete sail from one bank to another)
    - User can save the game, and load it any time later
    - New game, reset, and exit game options
    - Show instructions of the current level
    - Showing alert when the user move is invalid
7) For saving & loading the game
    - Save your game state in XML files
    - You should use SAX or DOM, or StAX parsers to write & read the XML files
8) Bonus points:
    - (Bonus) Solve button, which solves the puzzle for the user, no need to solve it in minimum number of steps, but make sure that your algorithm finds a solution
    - Implementing more design patterns than the mandatory ones
    - Extra Bonus for implementing all the suggested design patterns
    - Implementing extra level(s) from your design
    - Good UI design for the game

3) References

- You can download this game from here for Android, and from here for iOS.
- This tutorial may help you get introduced to XML parsers.
- Demo on how to make sample GUI for this game will be posted on piazza as soon as possible. However, there are many online tutorials will help
- [Hint for the Bonus] Solving these puzzles can be done using dynamic programming where:
    - **Base case**(stopping condition): all crossers are on the other side of the river.
    - **State**: Animals on both sides and the position of the raft. You must save this state so that if this state is repeated you don't need to investigate it again.
    - You must try all possible combinations of crossers that can ride the boat.
    - This playlist may help you get introduced to dynamic programming.

4) Deliverables

- You should work in groups of three or four.
- You should implement this assignment in java.
- The implementation for the given interfaces is a must.
- You are free to use any GUI framework you like.
- You should submit this assignment online on Github, you should include the following:
    - The **code** folder
    - **A self-executable jar file (JRE 1.8)**: The program should be executable by simply double clicking the icon provided that you have a running JRE.
      (Make sure your jar file is running with no conflicts on different computers)
    - You should deliver a report that contains the required UML diagram, describes your design thoroughly, sequence diagram that shows the execution flow of your game, and contains snapshots of your GUI and a user guide that explains how to use your application. Any design decisions that you have made should be listed clearly.
    - A **readme.txt** file with any assumptions you need to run the program, and with the detailed division of labor among the team members..
- Delivering a copy will be **severely penalized** for both parties, so delivering nothing is so much better than delivering a copy.
  Late submission is accepted for only for 3 days after the deadline and will be graded from 70% of the full grade

5) Appendix:

1. ICrosser interface

```
public interface ICrosser {

	/**
	 * @return whether the crosser can sail the boat or not
	 */
	public boolean canSail();

	/**
	 * @return get the weight of the crosser
	 */
	public double getWeight();

	/**
	 * @return get the eating rank of the crosser
	 * this rank can be used to detect if one
	 * crosser can eat/harm another crosser
	 */
	public int getEatingRank();

	/**
	 *
	 * @return images of the crosser
	 * each crosser must have at least two images, each one
	 * is used on one bank of the river
	 */
	public BufferedImage[] getImages();

	/**
	 * @return exact copy of the crosser
	 */
	public ICrosser makeCopy();

	/**
	 *  this field is used by the game strategy to set the label which
```

```
         *  will be shown beside the crosser in the game view
         *  to inform the user about the criteria of the current level
         *  e.g. crosser eating rank
         */
        public void setLabelToBeShown(String label);

        /**
         * @return gets label of the crosser depending on the current game
         */
        public String getLabelToBeShown();
}
```

2. ICrossingStrategy

```
public interface ICrossingStrategy
        /**
         * @param boatRiders which the user had selected to be moved to the
other bank
         * @param rightBankCrossers the crosses on the right bank
         * @param leftBankCrossers the crosses on the left bank
         * @return whether this move is valid
         * or not according to the rules
         */
        public boolean isValid(List<ICrosser> rightBankCrossers,
                        List<ICrosser> leftBankCrossers, List<ICrosser>
boatRiders);
        /**
         * @return the crossers of the left bank initially
         * the right bank has no crossers initially
         */
        public List<ICrosser> getInitialCrossers();
        /**
         * @return return the rules and the instructions of the current
         * strategy, to be shown to the user
         */
        public String[] getInstructions();
}
```

3. IRiverCrossingController

```
public interface IRiverCrossingController {


	/**
	 * this method initialize the controller with game strategy according
	 * to the level
	 * @param gameStrategy
	 */
	public void newGame(ICrossingStrategy gameStrategy);

	/**
	 * resets the game without changing the strategy
	 */
	public void resetGame();

	/**
	 * @return the current strategy instructions if the user want to see
them
	 */
	public String[] getInstructions();

	/**
	 * @return list of crossers on the right bank of the river
	 */
	public List<ICrosser> getCrossersOnRightBank();

	/**
	 * @return list of crossers on the left bank of the river
	 */
	public List<ICrosser> getCrossersOnLeftBank();

	/**
	 * @return determines whether the boat is on the left or on the right
```

```
bank of the river
       */
      public boolean isBoatOnTheLeftBank();

      /**
       * @return returns the number of sails that the user have done so far
       */
      public int getNumberOfSails();

      /**
       * @param crossers which the user selected to move
       * @param fromLeftToRightBank boolean to inform the controller
       * with the direction of the current game
       * @return  boolean if it is a valid move or not
       */
      public boolean canMove(List<ICrosser> crossers, boolean
fromLeftToRightBank);

      /**
       * this method ysed to perform the move if it is valid
       * @param crossers
       * @param fromLeftToRightBank
       */
      public void doMove(List<ICrosser> crossers, boolean
fromLeftToRightBank);

      /**
       * @return boolean providing that the undo action can be done or not
       */
      public boolean canUndo();

      /**
       * @return boolean providing that the redo action can be done or not
       */
      public boolean canRedo();
```

```
	/**
	 * undo to the last game state
	 */
	public void undo();

	/**
	 * redo the undo actions
	 */
	public void redo();

	/**
	 * saves the game state
	 * (left bank crossers, right bank crossers, number of done sails,
position of the boat)
	 */
	public void saveGame();

	/**
	 * load the saved game state
	 */
	public void loadGame();

	/**
	 * this function is bonus
	 * it returns the boat riders starting from the beginning of the game
	 * until the final solution to show the user the solution
	 */
	public List<List<ICrosser>> solveGame();

 }
```