# Project 1

## Memory Puzzle

CSC-18C Spring 2015 42030

Taught by Dr. Mark Lehr

Guthrie Price

May 20th, 2015

# Introduction

**Game Title:** Memory Puzzle

**Description:** The player is presented with an even number of cards that look identical. The player repeatedly picks two cards to flip over, trying to match cards until all cards are matched, or the player's score reaches zero.

## Rules

- **Objectives**
  - Match all cards successfully.
- **Constraints**
  - Only two cards that are unmatched can be chosen.
- **Game Ending Conditions**
  - The player matches all cards successfully—the player wins.
  - The player's score reaches zero—the player loses.

# Summary

## Outline of Implementation Goals

Implement a basic console version of Memory Puzzle using data structures used in class. Implement the board with a hash map and generate random cards using a queue. Allow the user to choose different sizes for the board.

## Basic Project Size Information

Total lines: 666
Lines of code: 456
Comment lines: 113
Blank lines: 97
Classes: 11

# Concept Checklist

## Data Structures

- Linked lists (though use of the stack)
- Stack (used in the card generator
- Hashing (used in the base class CordHashMap class, and its child class Board)
- Recursion (used in the power function)

# Class List

- **Board**
  - A class you can initialize based on the abstract class CordHashMap. Represents the board for the game.
  - Instance variables: None
  - Methods: None
- **Card**
  - Extends the CordObj class, represents a card for the game.
  - Instance variables(variable name(type)): hidden(*boolean*)
  - Methods(method name(*input type*)[*output type*]): isHidden(*None*)[*boolean*], hide(*none*)[*void*], show(*none*)[*void*], isEqual(*Card*)[*boolean*]
- **CardGenerator**
  - Extends the Stack class, responsible for generating random cards for the board.
  - Instance variables(variable name(type)): size(*int*), generator(*Random*), cardSymbols(*char[]*), cordSpace(*ArrayList<Coordinate>*), spaceIter(*Iterator<Coordinate>*)
  - Methods(method name(*input type*)[*output type*]): fillStack(*None*)[*void*], setSpace(*int*)[*void*], getRandCard(*None*)[*Card*]

- **Coordinate**
  - Represents a two-dimensional coordinate in space.
  - Instance variables(variable name(type)): x(*int*), y(*int*)
  - Methods(method name(*input type*)[*output type*]): setX(*Integer*)[*void*], setY(*Integer*)[*void*], getX(*None*)[*Integer*], getY(*None*)[*Integer*], isEqual(*Coordinate*)[*Boolean*], toString(*None*)[*String*]
- **CordHashMap**
  - Abstract class for hashing CordObjs.
  - Instance variables(variable name(type)): dimSize(*int*), space(*CordObj[]*)
  - Methods(function name(*input type*)[*output type*]): getSpace(*None*)[*CordObj[]*], getSize(*None*)[*int*], fillSpace(*Vector<CordObj>*)[*void*], add(*CordObj*)[*void*], hash(*Coordinate*)[*int*], toString(*None*)[*String*]
- **CordObj**
  - An abstract class representing an object in space that has a position. Has a class variable <T>.
  - Instance variables(variable name(type)): cord(*Coordinate*), data(*T*)
  - Methods(function name(*input type*)[*output type*]): getCord(*None*)[*Coordinate*], getData(*None*)[*T*], setCord(*Coordinate*)[*void*], setCord(*Coordinate*)[*void*], setData(*T*)[*void*], toString(*None*)[*String*]
- **Driver**
  - Driver for the game.
  - Instance variables(variable name(type)): None
  - Methods(function name(*input type*)[*output type*]): main(*String[]*)[*void*]

- **LinkedList**
  - Implementation of a generic linked list data structure.
  - Instance variables(variable name(type)): head(*Node<T>*), length(*int*)
  - Methods(function name(*input type*)[*output type*]): append(*T*)[*void*], insert(*T, index*)[*void*], remove(*int*)[*boolean*], get(*int*)[T], size(*None*)[*int*], toString(*None*)[*String*]
- **Node**
  - Inner class of the Linked List data structure. Represents a single node of data.
  - Instance variables(variable name(type)): data(*T*), next(*Node<T>*)
  - Methods(function name(*input type*)[*output type*]): getData(*None*)[*T*], getNext(*None*)[*Node*], setData(*T*)[*void*], setNext(*Node<T>*)[*void*]
- **MemoryPuzzle**
  - Main class for the game.
  - Instance variables(variable name(type)): cards(*CardGenerator*), boardSize(*int*), board(*Board*), input(*Scanner*)
  - Methods(function name(*input type*)[*output type*]): fillBoard(*None*)[*void*], printBoard(*None*)[*void*], getCord(*None*)[*Coordinate*], getNumber(*String, int, int*)[*int*], waitForInput(*None*)[*void*], gameOver(*int*)[*boolean*], getCard(*None*)[*Card*], printMainMenu(*None*)[*void*], validBoardSize(*None*)[*boolean*], run(*None*)[*void*], toString(*None*)[*String*]
- **Stack**
  - Implementation of a generic stack data structure using a linked list. Has a class variable <T>.
  - Instance variables(variable name(type)): container(*LinkedList<T>*)
  - Methods(function name(*input type*)[*output type*]): push(*T*)[*void*], pop(*None*)[*T*], peek(*None*)[*T*], empty(*None*)[*boolean*], size(*None*)[*int*], toString(*None*)[*String*]

- **Utility**
  - A utility class for useful functions that don't belong anywhere else
  - Instance variables(variable name(type)): None
  - Methods(function name(*input type*)[*output type*]): pow(*int*, *int*)[*int*]

# Code from select classes

## Memory Puzzle

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class MemoryPuzzle {

        private CardGenerator cards; //The card generator to ensure random cards and even amounts
of card types
        private int boardSize=0; //The board size
        private Board board; //Container for cards
        private Scanner input;

        public MemoryPuzzle(char[] symbols){
         input = new Scanner(System.in);
                cards = new CardGenerator(boardSize,symbols);
        }

    //Fills the board with the cards generated by the card generator
    private void fillBoard(){
        board = new Board(boardSize); //Initialize the board
        cards.fillStack(); //Generate a stack of cards
        //Dump the stack of cards into the board
        for(int i=0;i<boardSize*boardSize;i++){
                board.add(cards.pop());
        }
    }

    //Prints the board to the screen
    public void printBoard(){
        int spaceSize = boardSize*boardSize;
        //Print out the column numbers
        for(int i=0;i<boardSize;i++)
            System.out.print(" "+(i+1)+" ");
        System.out.println();
        //Print out each card followed by the row number
        for(int i=0;i<spaceSize;i++){
            Card card = (Card)board.getSpace()[i];
            if(i%board.getSize() == 0 && i != 0)
                System.out.println(" "+(i/boardSize));
            //If the card is hidden, print out a blank card
            if(card.isHidden())
                System.out.print("[ ]");
            //If the card isn't hidden, print out the symbol on the card
            else
                System.out.print("["+card.getData()+"]");
        }
        //Print out the final row number
        System.out.print(" "+boardSize);
        System.out.println("\n");
    }

    //Gets a coordinate from the user
    public Coordinate getCord(){
        int x;
        int y;
        x = getNumber("Enter x coordinate",1,boardSize);
        y = getNumber("Enter y coordinate",1,boardSize);
```

```
    return new Coordinate(x,y);
}

//Gets a number from the user within the lower and upper bounds
public int getNumber(String prompt, int lowBnd, int upBnd){
    int choice=0;
    do{
        try{
            System.out.print(prompt+"("+lowBnd+"-"+upBnd+"): ");
            choice = input.nextInt();
        }
        catch(InputMismatchException e){
            choice=lowBnd-1;
            input.next();
        }
        finally{
            if(choice>upBnd || choice<lowBnd){
                System.out.println("Oops, invalid choice, try again.");
            }
        }
    }while(choice>upBnd || choice<lowBnd);

    return choice;
}

//Waits for the user to press enter
public void waitForInput(){
    System.out.print("Press enter to continue ");
    input.nextLine();
}

//Checks to see if the game is over
//The game is over if the all the cards on the board are not hidden
//or if the user ran out of guesses
public boolean gameOver(int guesses){
    if(guesses==0) return true;
    for(CordObj card : board.getSpace()){
        if(((Card)card).isHidden())
            return false;
    }
    return true;
}

//Returns a card based on the coordinate chosen by the user
public Card getCard(){
    Card choice;
    choice = (Card)board.get(getCord());
    while(!(choice.isHidden())){
        System.out.println("Oops, try again");
        choice = (Card)board.get(getCord());
    }
    return choice;
}

//Prints the main menu
public void printMainMenu(){
    System.out.println("Main Menu");
    System.out.println("-------------------");
    System.out.println("1. Start game");
    System.out.println("2. Quit\n");
};

//Validates board size. I.E. must be divisible by 2
public boolean validBoardSize(){
    if(boardSize % 2 == 0)
        return true;
    System.out.println("Board size must be divisible by 2");
    return false;
}

//Runs the game
public void run(){
    int gameChoice;
    Card choice1;
    Card choice2;
```

```
        do{
            printMainMenu();
            gameChoice = getNumber("Enter menu choice",1,2);
            switch(gameChoice){
                case 1:
                    do{
                        boardSize = getNumber("Enter board size (must be divisible by 2)",4,32);
                    }while(!validBoardSize());

                    cards.setSpace(boardSize);
                    fillBoard();
                    int guesses = boardSize*boardSize/2;
                    while(!gameOver(guesses)){
                        //While the game isn't over get guesses from the user
                        System.out.println();
                        printBoard();
                        System.out.println("Number of incorrect guesses remaining: "+guesses);
                        System.out.println("Enter first coordinate");
                        choice1 = getCard();
                        System.out.println("Enter second coordinate");
                        choice2 = getCard();

                        //If the symbols on the cards are the same, then the user got a match
                        //Set the cards to not be hidden
                        if(choice1.isEqual(choice2)){
                            System.out.println("\nYou got a match!");
                            choice1.show();
                            choice2.show();
                        }

                        //In the other case the user didn't get a match
                        //Show the cards and then flip them back over
                        else{
                            System.out.println("\nYou missed!\n");
                            choice1.show();
                            choice2.show();
                            printBoard();
                            choice1.hide();
                            choice2.hide();
                            guesses-=1;
                        }
                    System.out.println();
                    }

                    //Print out the ending messages based on win or loss
                    System.out.println();
                    printBoard();
                    if(guesses == 0)
                        System.out.println("\nYou ran out of guesses, you lose...");

                    else{
                        System.out.println("\nCongratulations, you won!");
                        System.out.println("Your score for this game is: "+guesses+"\n");
                    }
                    System.out.println();
                    break;
                case 2:
                    System.out.println("Ending game");
                    break;
                default:
                    System.out.println("Error, magic happened");
                    System.exit(1);
            }
        }while(gameChoice!=2);

    }

    //Prints the board out as a string
    public String toString(){
       return board.toString();
    }
}
```

# CodeHashMap

```
        import java.util.Vector;

        public abstract class CordHashMap{

            private int dimSize; //Size of one dimension of the space
            private CordObj[] space; //The space of coordinate objects

            public CordHashMap(int size){
                dimSize = size;
                space = new CordObj[size*size];
            }

            //Returns the coordinate space
            public CordObj[] getSpace(){
                return space;
            }

            //Returns the dimension size
            public int getSize(){
                return dimSize;
            }

            //Fills the coordinate space with coordinate objects
            public void fillSpace(Vector<CordObj> objs){
                //TODO fix error for objects with coordinates outside the space size
                for(CordObj obj : objs){
                    add(obj);
                }
            }

            //Sets a specific coordinate to a coordinate object
            public void add(CordObj obj){
                getSpace()[hash(obj.getCord())] = obj;
            }

            //Returns the coordinate object at a coordinate
            public CordObj get(Coordinate c){
                return getSpace()[hash(c)];
            }

            //Maps a 2D coordinate to the coordinate space
            public int hash(Coordinate c){
                return ((c.getX()-1)+((c.getY()-1)*dimSize));
            }

            public String toString(){
                String result = "";
                for(int i=0;i<dimSize*dimSize;i++){
                    result = result+space[i].toString()+" ";
                }
                return result;
            }
}
```