

Snap Applet Specifications

Course:

CSC8416 Advanced Programming in Java

Applet Location:

<http://www.swansonblog.com/geekspeak/2009/01/robosnap/>

Author:

Paul Swanson, Q1220200

Introduction

The fundamental requirement of this assignment was create an applet that implemented threads and allowed the user to interact with them in a variety of ways. To fulfil these requirements I designed a program that would play the simple card game “Snap” on its own. While it seemed fairly a straight forward prospect at the outset of design and planning it soon became apparent that it was more involved than first expected. Nonetheless, the Snap Applet embodies all of the specified requirements and demonstrates the ability to run and manage threads in a Java Applet.

Class Structures

Key to the design and function of the Snap Applet is the various class structures that define it. They are as follows:

- SnapVocab
- PlayingCard
- Dealer
- Player (includes buttonAdapter)
- Snap (includes buttonAdapter)
- GameMonitor
- SnapApplet

The first drafts of this Applet included a number of abstract classes but these were removed after it was discovered that they added little value to the project and generated a substantial amount of difficulty. The remaining classes are themselves quite extensible and reusable. The following is a detailed description of the classes.

SnapVocab

SnapVocab is a simple *enum* class used for messaging between the *Snap* and *Player* classes.

GAME_OVER	Signal end of current game
YOUR_TURN	Signal turn of a given player
DUPLICATE_CARDS	Signal duplicate cards on pile, time to snap

PlayingCard

PlayingCard is a class representing a regular playing card, the sort used in homes and casinos. It contains the following members:

final static protected String[] frenchFaceValues	Array of face values
final static protected String[] frenchSuitValues	Array of suits
final protected int value	The face value of the card
final protected int suit	The suit of the card
PlayingCard(int value)	Default constructor, suit assigned zero

PlayingCard(int value, int suit)	Overridden constructor
public int getValue()	Returns face value
public int getSuit()	Returns suit
public String getStringValue()	Returns face value as a String
public String getStringSuit()	Returns suit as a String
public String getName()	Returns face value & suit as a String

Dealer

Dealer is a class that serves the function of generating, shuffling and dealing the playing cards to the players.

protected int numberOfPlayers;	The number of players to be dealt cards
protected int deckSize;	The size of the deck of cards
protected int handSize = 0;	The even number of cards per hand
protected int excessCards = 0;	The remaining cards to be dealt unevenly
protected LinkedList<PlayingCard> deckOfCards;	The deck of cards
Dealer(int numberOfPlayers)	Constructor that requires players numbers
protected void calcPortions()	Calculates how many cards each hand gets
protected void buildDeck()	Populates the deck with cards
public void shuffleDeck()	Shuffles the deck
public LinkedList<PlayingCard> dealCards()	Returns a single hand / portion of cards
public void say(String message)	Outputs messages
public void showDeck()	Displays the cards deck
public int getDeckSize()	Returns deck size
public int getNumberOfPlayers()	Returns the number of players

Player implements Runnable

Player class provides most of the functionality associated with the computer generated card player. This class provides the basis for the *Player* thread.

protected Snap game;	Reference to the current Snap game
protected String name;	Name of the player
protected int playerNumber;	Unique number of the player
protected LinkedList<PlayingCard> playersHand;	The cards the playing holds
final protected int agilityVariance = 200;	Amount of variance for randomised agility
protected int minReactionTime = 50;	The minimum reaction delay
protected int agility = 100;	Default agility of a player
protected boolean myTurn	Flag signalling players turn
protected boolean threadSuspended	Flag signalling the thread is suspended
protected boolean iQuit	Flag signalling time to quit the thread and game
protected boolean snapTime	Flag signalling time to attempt a "Snap"
protected Button btnSuspendResume;	Button for suspending the player's thread
Panel pnlMain	Panel for the player's components

Label lblPlayerName, lblCardCount, lblStatus	Labels used in the panel pnlMain
Player(String name, Snap game)	Constructor that sets player name and game
protected void say(String message)	Output a message
private void playCard()	Place a card onto the current card pile
public synchronized void playGame()	Deal with asynchronous events in the game
public synchronized void yourTurn()	Method for a synchronized turn
public synchronized void suspendResume()	Suspend & Resume the player
public void run()	Main loop of the player's thread
public void tell(SnapVocab message)	Used by Snap class for communicating
public void getCards(LinkedList<PlayingCard> cards)	Add cards to the players hand
public LinkedList<PlayingCard> showCards()	Show cards in the player's hand
public int numberOfCards()	Returns number of cards in the player's hand
private void showCardCount()	Updates the player's card count label
public void setStatus(String message)	Update the player's status label
class buttonAdapter implements ActionListener	Parses input from control buttons
-- public synchronized void actionPerformed(ActionEvent e)	Map mouse clicks to corresponding actions

Snap implements Runnable

Snap class is responsible for the mediating, or running, of the game. It generates the appropriate number of *Player* objects, their corresponding threads and manages them. It determines how the game start, plays and finishes.

Dealer snapDealer	The Dealer object for the game
Player[] snapPlayers	An array of players objects
Thread[] playerThreads	An array of player's threads
Thread gameMonitor	Connection to the current game monitor
Panel pnlMain	Panel divides the viewing area into 3 rows
Panel pnlPlayers	Panel contains a grid of player's components
Panel pnlCardPile	Panel displays the top 2 cards on the pile
Panel pnlControls	Panel displays the game control components
Button btnSuspendResume, btnSpeedUp, btnSlowDown, btnStopRestart;	Buttons for game controls
Label lblTopCard, lblSecondCard;	Labels for the top 2 cards on the pile
Checkbox chbSynchronized;	Checkbox for toggling synchronized mode
private int numberOfPlayers	Number of players in the game
private LinkedList<PlayingCard> cardPile	The card pile used in the game
private boolean duplicateCards	Flags if top two cards are of duplicate value
boolean gameOver	Flag for game over
private int turnDelay	Default pause between turns, regulates speed
private int snapDelay	Enables winning snapper to start next hand
private boolean threadSuspended	Flag for suspend / resume for the game
private boolean synchronizedMode	Flag for synchronized gameplay mode
Snap(int numberOfPlayers)	Constructor for new Snap game
private void doTurnDelay()	Sleep for a little while to regulate game speed
public synchronized void wakeUp()	Wakes up Snap thread gently

protected void say(String message)	Displays message
public synchronized LinkedList<PlayingCard> getHand()	Dealer provides Player with opening hand of cards
public synchronized void putCard(PlayingCard topCard)	Places a card face up on the pile
public synchronized LinkedList<PlayingCard> snapCards()	Called by snapping players
public synchronized void suspendResume()	Suspend / Resumes the Snap game
public void synchronizedMode() Toggle synchronizedMode	Toggles Suspend / Resume start of game
public void setTurnDelay(int delay)	Adjusts turn delay, controls speed of the game
public int getTurnDelay()	Returns current turn delay
public void announce(String message)	Makes a prominent announcement
public void announce(String message, String subtitle)	Makes a prominent announcement with subtitle
public boolean pileEmpty()	Checks if the card pile is empty
public void run()	Main loop for the Snap game thread
class buttonAdapter implements ActionListener, ItemListener	Parses input from control buttons
public synchronized void itemStateChanged(ItemEvent e)	Maps mouse clicks to corresponding actions

GameMonitor implements Runnable

GameMonitor runs parallel to *Snap* and enables the user to completely restart the game. It does its job by monitoring the Snap game and reinstantiating the appropriate objects when the user requests.

Snap snapGame	Reference to the current Snap game
Thread snapThread	Reference to the current Snap thread
SnapApplet hostApplet	Reference to the current Applet object
GameMonitor(SnapApplet hostApplet, Snap snapGame, Thread snapThread)	Constructor connects game monitor to the appropriate applet, game and thread
public void run()	Main loop for the game monitor's thread

SnapApplet extends Applet

SnapApplet is the class responsible for allowing the game to be executed as an Applet inside a web browser. It loads the Snap game, thread and its monitor.

StringBuffer buffer	A buffer for outputting to the Applet
Snap mySnapGame	The Snap game object
GameMonitor mySnapMonitor	The game monitor object
Thread snapThread	The Snap thread
Thread snapMonitor	The game monitor's thread
final static int numberOfPlayers	The number of players for the game
public void init()	First method of the Applet
public void start()	Starts everything running
public void stop()	Method called when Applet stopped
public void destroy()	Method called when Applet destroyed
void addItem(String newWord)	Adds text to the Applet
public void paint(Graphics g)	Paints the Applet

Program Design

The Snap Game

A game of “Snap” is played by a given number of players, each with a portion of the card deck. On their turn, each player places one card on the pile. If the top two cards on the pile are of identical face value the first player to call “Snap!” and place their hand on the deck wins those cards. The game ends when one player successfully “snaps” all cards in the deck, thus becoming the winner, or no player does, resulting in a draw.

Snap Applet Implementation

In this program the Snap game is represented by a threaded object which spawns a threaded object for each computerised player (Player). The Snap thread mediates gameplay and determines the outcome of the game and the conclusion of program execution. Both Player threads, and the Snap thread itself, can be controlled by the user. The speed of gameplay can also be manipulated by adjusting the delay on each player's turn. Notably, the entire Snap game can be reinstated (i.e. restarted) through the implementation of a monitor object and thread that runs in parallel to the Snap game.

To reduce processing overhead the Snap thread “trusts” the Player threads with certain determinations, such as who wins, this way it does not need to keep polling every Player for a detailed status. In general, whenever a thread is finished with a cycle of its main loop (i.e. run()) it puts itself, or is put, in a waiting state to ensure that it doesn't waste processor time. cursory examinations have shown that whilst this Applet is running very little load is actually applied to the underlying system.

While a game is in process the Snap threads call each Player in turn, so at this point interaction is fairly synchronized. But when the Snap object detects duplicate valued cards on the pile it informs all Player objects and wakes their threads. At this point Player threads will asynchronously attempt to “snap” the card pile, only one will be successful. Originally a system of messaging was used to notify Players of their turn, but as this involved waking threads and so on, it actually was capable of resulting in Players, or some of their functions, happening out of order. A synchronization option was added to the interface to enable this kind of behaviour as an alternate to the more synchronized and regulated process of Snap calling a method on each Player, which is the default.

The Snap Applet was written using Java Standard Edition 6 JDK, mainly because that is what was available, but it also provided some useful features such as Generics.

Reflections on the Design and Implementation Process

The design and implementation of the Snap Applet was an interesting learning experience. While the initial idea and design seemed simple enough the implementation showed up every weakness. Once the extent of the situation was realised the point-of-no-return had long been reached. Nonetheless, the Applet was successfully implemented and has no known bugs. During the process I learned a lot about building an entire application in Java and particularly about using threads and the AWT. I also learned that there is a lot I don't know and that in Java nothing should be taken for granted. Hopefully some of the lessons learned in this assignment will assist me in my work on Assignment 2.