

자료구조

103분반

2010.11.25

학번 : 200611516

이름 : 김성현

자료구조

프로그래밍 과제 보고서

1. 과제 명 : 자료구조 프로그래밍 과제 #2 "One Card"

2. 제출자: 분반 200611516 김성현

3. 프로그램의 완성도

(1) 컴파일 오류 여부

a. 오류가 발생하지 않는다 

b. 오류가 발생한다

c. 기타의 경우 ( )

(2) 실행 오류 여부

a. 항상 오류가 발생한다 

b. 대부분의 경우 오류가 발생한다 

c. 일부 예외적인 데이터에 대해서만 오류가 발생한다 

d. 내가 아는 한 오류가 발생하지 않는다 

e. 기타의 경우 ( )

(3) 기능적 요구사항의 만족도

a. 요구된 모든 기능을 충족한다

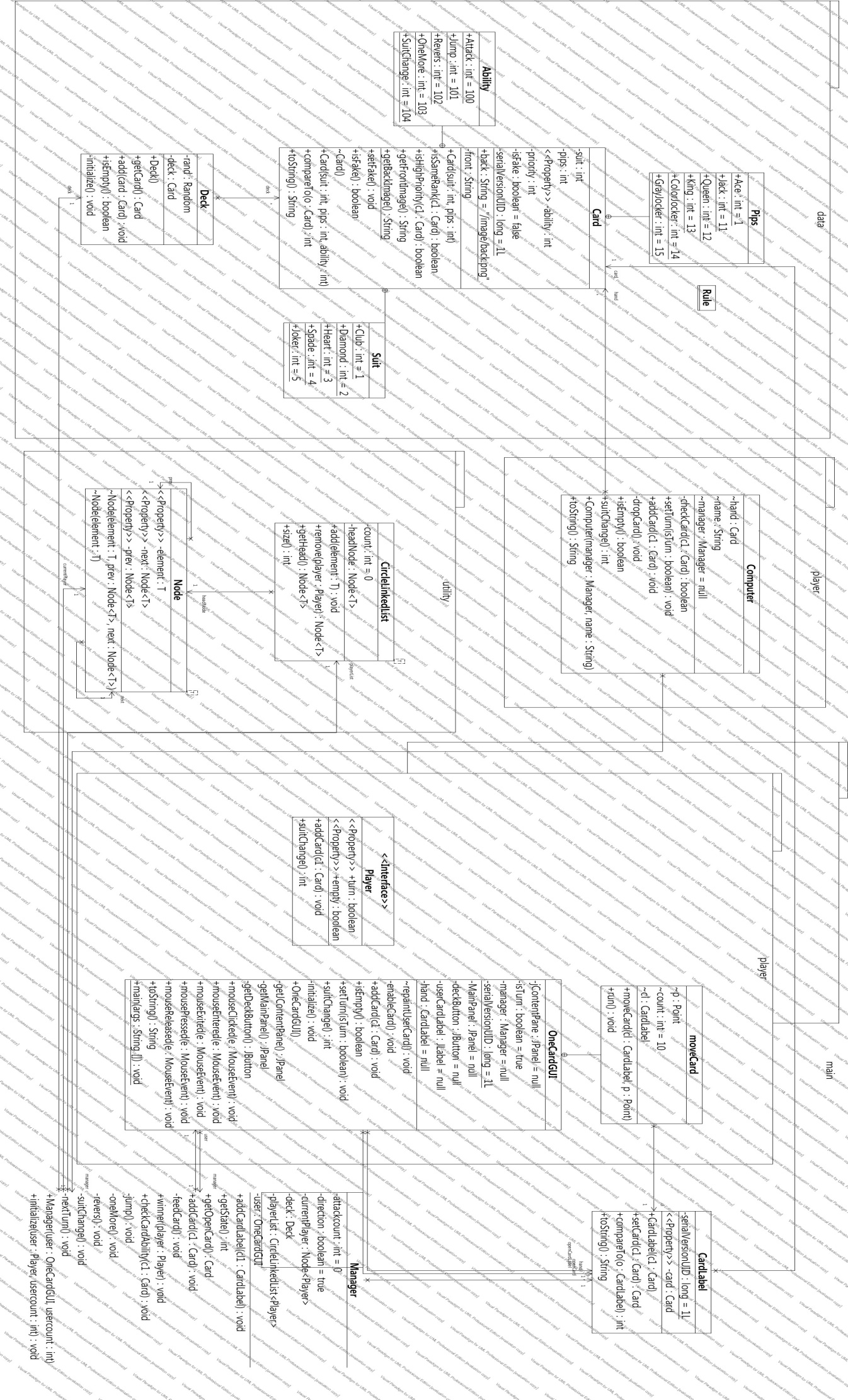
b. 요구된 대부분의 기능을 충족한다 

c. 일부의 기능만을 충족한다 

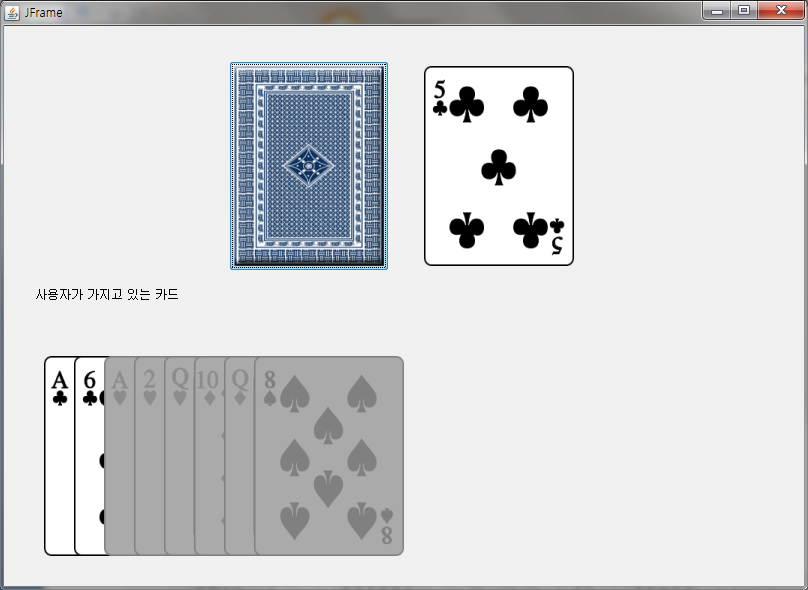
d. 사실 거의 한 게 없다 

e. 기타의 경우 ( )

(4) 미완성인 경우 어떤 부분까지 구현하였는지 구체적으로 정확하게 기술하시오.

현제 프로그램을 진행하면 할수록 프로그램이 시스템에서 점유하는 메모리의 양이 증가하는데 이것은 프로그램의 구조상 메소드를 스택으로 호출해서 실행을 하면 할수록 스택에 저장되는 내용이 커지기 때문인 것으로 추측됩니다.

6. 프로그램 실행 예

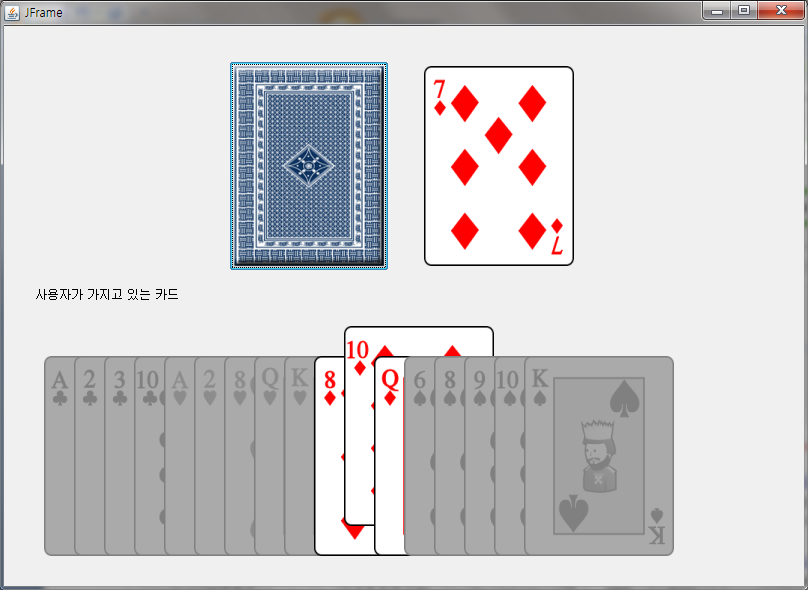


윗쪽에 있는 카드 뒷면과 앞면은

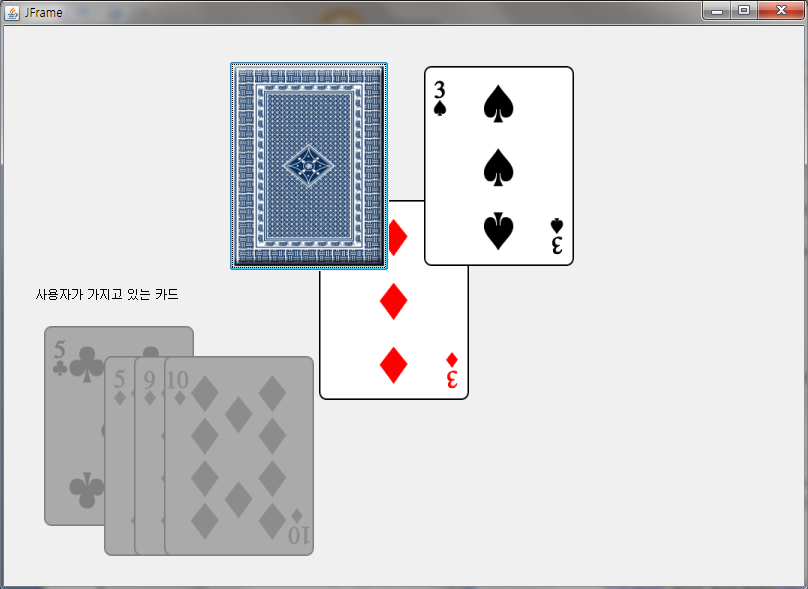
뒷면은 쌓여있는 덱을 의미하고, 앞면은 현재 바닥에 놓여있는 카드를 의미합니다.

바닥에 놓여있는 카드와 상황에 따라서 내려놓을 수 없는 카드 들은 disable 상태로 만듭니다.

회색으로 표시되는 것은 JLabel 의 기본적인 옵션입니다.

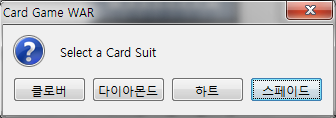


카드 위에 마우스를 올리면 해당 카드가 약간 위로 올라가서 어느 카드가 선택 되었는지 알기 쉽게 합니다.



카드를 선택하면 일정 속도로 움직여서 바닥에 놓여진 카드로 움직이는 에니메이션을 보여줍니다.

7번 카드를 내려놓을 경우



의 옵션 페널을 이용하여 모양을 바꿀 수 있게 합니다.

7. [Optional] 본 과제를 하면서 느낀 점을 기술하시오

포커나 기타 게임과 달리 카드 마다 특정한 역활이 있어서 생각보다 어려웠습니다.

카드가 움직이는 에니메이션을 만들 때 쓰레드를 안쓰고 그냥 했다가 동작을 안해서 애를 먹었습니다.

메소드가 계속해서 호출되는 이유로 시간이 지날수록 메모리 점유율이 늘어나는데 이것을 해결하려고 Player 을 상속 받는 클레스에 Thread를 상속시켜서 해결하려 했으나 시간 관계상 구현은 못했습니다.

PS. 교수님 저기 지난번 Simple Dictionary 에서 컴파일 되는데 컴파일 안됨 에 체크 해놨더라구요.. 죄송합니다

8. 프로그램 소스 코드

Data Package

-Class Card.java

**package** data;

**import** java.io.Serializable;

**public** **class** Card **implements** Comparable<Card>, Serializable{

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **static** **class** Pips{

**public** **final** **static** **int** *Ace* = 1;

/\*

public final static int Two = 2;

public final static int Three = 3;

public final static int Four = 4;

public final static int Five = 5;

public final static int Six = 6;

public final static int Seven= 7;

public final static int Eight = 8;

public final static int Nine = 9;

public final static int Ten = 10;

\*/

**public** **final** **static** **int** *Jack*=11;

**public** **final** **static** **int** *Queen* = 12;

**public** **final** **static** **int** *King* = 13;

**public** **final** **static** **int** *ColorJocker* = 14;

**public** **final** **static** **int** *GrayJocker* = 15;

}

**public** **static** **class** Suit{

//♠스페이드>♥하트>◆다이아몬드>♣클로버 // 포커

**public** **final** **static** **int** *Club* =1;

**public** **final** **static** **int** *Diamond* =2;

**public** **final** **static** **int** *Heart*=3;

**public** **final** **static** **int** *Spade* =4;

**public** **final** **static** **int** *Joker*=5;

}

**public** **static** **class** Ability{

//특수능력 //1이상 100이하 공격력

**public** **final** **static** **int** *Attack*=100;

**public** **final** **static** **int** *Jump* =101;

**public** **final** **static** **int** *Revers* =102;

**public** **final** **static** **int** *OneMore*=103;

**public** **final** **static** **int** *SuitChange*=104;

}

//카드 정보

**private** **int** suit;

**private** **int** pips;

**private** **int** ability;

**private** **int** priority;

**private** **boolean** isFake =**false**;

**public** **static** **final** String *back*="/image/back.png";

**private** String front;

Card()

{

}

**public** Card(**int** suit, **int** pips)

{

**this**.suit=suit;

**this**.pips=pips;

priority=0;

ability=0;

front="/image/"+suit+"\_"+pips+".png";

**switch**(pips)

{

**case** Card.Pips.*Ace*:

**if** (suit==Card.Suit.*Spade*)

{

priority=3;

ability=5;

}**else**{

priority=2;

ability=3;

}

**break**;

**case** 2:

priority=1;

ability=2;

**break**;

**case** 7:

ability=Card.Ability.*SuitChange*;

**break**;

**case** Card.Pips.*Jack*:

ability=Card.Ability.*Jump*;

**break**;

**case** Card.Pips.*Queen*:

ability=Card.Ability.*Revers*;

**break**;

**case** Card.Pips.*King*:

ability=Card.Ability.*OneMore*;

**break**;

**case** Card.Pips.*ColorJocker*:

ability=10;

priority=3;

**break**;

**case** Card.Pips.*GrayJocker*:

ability=7;

priority=3;

**break**;

}

}

**public** Card(**int** suit, **int** pips,**int** ability)

{

**this**.suit=suit;

**this**.pips=pips;

**this**.ability=ability;

front="/image/"+suit+"\_"+pips+".png";

**switch**(pips)

{

**case** 1:

**if** (suit==Card.Suit.*Spade*)

priority=3;

**else**

priority=2;

**break**;

**case** 2:

priority=1;

**break**;

**case** 14:

**case** 15:

priority=3;

**break**;

**default** :

priority=0;

}

}

**public** **boolean** isSameRank(Card c1)

{

**if**(suit == c1.suit || pips==c1.pips||c1.suit==Card.Suit.*Joker*||suit==Card.Suit.*Joker*)

**return** **true**;

**else**

**return** **false**;

}

**public** **boolean** isHighPriority(Card c1)

{

//c1이 this 보다 높거나 같은 우선순위라면 true

//같은 글자 또는 높은 우선순위

//c1==opencard

**if**(**this**.pips==c1.pips){

//같은 글자(A,2..)일때

**return** **true**;

}**else** **if**( (**this**.priority>=c1.priority) ){

//System.out.println(this+", " +c1+" : "+(this.priority>=c1.priority)+", "+( this.suit == c1.suit || this.suit==Card.Suit.Joker ));

//우선순위가 높으면서 모양이 같을때

System.*out*.println(( **this**.suit == c1.suit) || (c1.suit==Card.Suit.*Joker* ));

**return** ( **this**.suit == c1.suit) || (**this**.suit==Card.Suit.*Joker* );

}

**return** **false**;

}

**public** **int** getAbility()

{

**return** ability;

}

**public** String getFrontImage()

{

**return** front;

}

**public** **static** String getBackImage()

{

**return** *back*;

}

@Override

**public** **int** compareTo(Card o) {

**return** suit\*1000+pips-o.suit\*1000-o.pips;

}

**public** String toString()

{

//♠스페이드>♥하트>◆다이아몬드>♣클로버 // 포커

String temp="";

**switch**(suit)

{

**case** 1:

temp+="♣"; **break**;

**case** 2:

temp+="◆"; **break**;

**case** 3:

temp+="♥"; **break**;

**case** 4:

temp+="♠"; **break**;

**case** 5:

temp+="Joker"; **break**;

}

**switch**(pips)

{

**case** Card.Pips.*Ace*:

temp+="A"; **break**;

**case** Card.Pips.*Jack*:

temp+="J"; **break**;

**case** Card.Pips.*Queen*:

temp+="Q"; **break**;

**case** Card.Pips.*King*:

temp+="K"; **break**;

**case** Card.Pips.*ColorJocker*:

temp+=" Color"; **break**;

**case** Card.Pips.*GrayJocker*:

temp+=" Gray"; **break**;

**default**:

temp+=pips; **break**;

}

**return** temp;

}

**public** **void** setFake()

{

**this**.isFake=**true**;

}

**public** **boolean** isFake()

{

**return** isFake;

}

}

-Class Deck.java

**package** data;

**import** java.util.ArrayList;

**import** java.util.Random;

**public** **class** Deck{

**private** ArrayList <Card> deck;

**private** Random rand;

**public** Deck(){

deck=**new** ArrayList<Card>();

rand=**new** Random();

initialize();

}

**private** **void** initialize()

{

**for**(**int** suit=1;suit<=4;suit++)

{

**for**(**int** pips=1;pips<=13;pips++)

{

add(**new** Card(suit,pips));

}

}

add(**new** Card(Card.Suit.*Joker*,Card.Pips.*ColorJocker*));

add(**new** Card(Card.Suit.*Joker*,Card.Pips.*GrayJocker*));

//System.out.println(deck.size());

}

**public** Card getCard()

{

**if**(!isEmpty())

{

//System.out.println(deck.size());

**return** deck.remove(rand.nextInt(deck.size()-1));

}**else** {

**return** **null**;

}

}

**public** **void** add(Card card)

{

**if**(!card.isFake())

deck.add(card);

}

**public** **boolean** isEmpty()

{

**return** deck.isEmpty();

}

}

Main Package

-Class CardLabel.java

**package** main;

**import** java.awt.Image;

**import** javax.swing.ImageIcon;

**import** javax.swing.JLabel;

**import** data.\*;

**public** **class** CardLabel **extends** JLabel **implements** Comparable<CardLabel> {

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** Card card;

**public** CardLabel(Card c1)

{

**super**();

card=c1;

ImageIcon icon=**new** ImageIcon(getClass().getResource(card.getFrontImage()));

icon.setImage(icon.getImage().getScaledInstance(150, 200, Image.*SCALE\_SMOOTH*));

setIcon(icon);

}

**public** Card setCard(Card c1)

{

Card temp=card;

card=c1;

ImageIcon icon=**new** ImageIcon(getClass().getResource(card.getFrontImage()));

icon.setImage(icon.getImage().getScaledInstance(150, 200, Image.*SCALE\_SMOOTH*));

setIcon(icon);

**this**.repaint();

**return** temp;

}

**public** Card getCard()

{

**return** card;

}

**public** **int** compareTo(CardLabel o) {

// **TODO** Auto-generated method stub

**return** card.compareTo(o.card);

}

**public** String toString()

{

**return** getCard().toString();

}

}

-Class Manager.java

**package** main;

**import** utility.\*;

**import** player.\*;

**import** data.\*;

**public** **class** Manager {

**private** Node<Player> currentPlayer;

**private** Deck deck;

**private** CircleLinkedList <Player>playerList;

**private** **int** attackcount=0;

**private** **boolean** direction=**true**;

**private** CardLabel openCard;

**private** OneCardGUI user;

//true next, false prev

**public** Manager(OneCardGUI user,**int** usercount)

{

**this**.user=user;

**this**.openCard=user.getOpenCardLabel();

initialize(user,usercount);

}

**public** **void** initialize(Player user,**int** usercount)

{

deck=**new** Deck();

playerList=**new** CircleLinkedList<Player>();

playerList.add(user);

currentPlayer=playerList.getHead();

**for** (**int** i=1;i<usercount;i++)

{

Computer temp=**new** Computer(**this**,"컴퓨터"+i);

//temp.start();

playerList.add(temp);

}

**for**(**int** k=0;k<usercount;k++)

**for**(**int** i=0;i<7;i++)

{

currentPlayer.getElement().addCard(deck.getCard());

currentPlayer=currentPlayer.getNext();

}

}

**public** **void** addCardLabel(CardLabel cl1) {

user.repaintUserCard();

**if**(cl1==**null**)

addCard(**null**);

**else**

addCard(cl1.getCard());

}

**public** **int** getState()

{

**return** attackcount;

}

**public** Card getOpenCard()

{

**return** openCard.getCard();

}

**public** **void** addCard(Card c1)

{

**if**(playerList.size()==1)

{

winner(currentPlayer.getElement());

}

currentPlayer.getElement().setTurn(**false**);

System.*out*.println(c1+" : "+ currentPlayer.getElement());

**if**(c1!=**null**){

deck.add(openCard.getCard());

openCard.setCard(c1);

System.*out*.println("OpenCard : "+ openCard.getCard());

**if**(currentPlayer.getElement().isEmpty())

{

winner(currentPlayer.getElement());

}

checkCardAbility(c1);

}**else**{

System.*out*.println("카드를 먹습니다. 처묵처묵");

feedCard();

}

}

**private** **void** feedCard()

{

**if**(attackcount==0)

attackcount=1;

**for**(**int** i=0;i<attackcount;i++)

currentPlayer.getElement().addCard(deck.getCard());

attackcount=0;

**if**(currentPlayer.getElement().equals(user))

user.repaintUserCard();

nextTurn();

}

**public** **void** winner(Player player)

{

System.*out*.println("Winner is"+player);

System.*exit*(0);

}

**public** **void** checkCardAbility(Card c1)

{

System.*out*.println(c1.getAbility());

**if**(c1.getAbility()<1)

{

nextTurn();

}**else** **if**(c1.getAbility()< Card.Ability.*Attack*){

**this**.attackcount+=c1.getAbility();

System.*out*.println("공격점수"+attackcount);

nextTurn();

}**else** **if**(c1.getAbility()==Card.Ability.*Jump*){

jump();

}**else** **if**(c1.getAbility()==Card.Ability.*Revers*){

revers();

}**else** **if**(c1.getAbility()==Card.Ability.*OneMore*){

oneMore();

}**else** **if**(c1.getAbility()==Card.Ability.*SuitChange*){

suitChange();

}

}

**private** **void** jump()

{

System.*out*.println("점프");

**if**(direction){

currentPlayer=currentPlayer.getNext();

}**else**{

currentPlayer=currentPlayer.getPrev();

}

nextTurn();

}

**private** **void** oneMore()

{

System.*out*.println("한번더");

currentPlayer.getElement().setTurn(**true**);

}

**private** **void** revers()

{

System.*out*.println("반대로");

direction=!direction;

nextTurn();

}

**private** **void** suitChange()

{

deck.add(openCard.getCard());

**int** k=currentPlayer.getElement().suitChange();

System.*out*.println("SuitChange : "+ k);

Card temp=**new** Card(k,7);

temp.setFake();

openCard.setCard(temp);

System.*out*.println("SuitChange : "+ openCard.getCard());

nextTurn();

}

**private** **void** nextTurn()

{

**if**(direction){

currentPlayer=currentPlayer.getNext();

}**else**{

currentPlayer=currentPlayer.getPrev();

}

System.*out*.println("턴 : "+currentPlayer.getElement());

currentPlayer.getElement().setTurn(**true**);

}

}

-Class OneCardGUI.java

**package** main;

**import** javax.swing.\*;

**import** data.Card;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.util.\*;

**import** player.Player;

**import** java.awt.Dimension;

**import** java.awt.Rectangle;

**public** **class** OneCardGUI **extends** JFrame **implements** MouseListener,Player{

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** JPanel jContentPane = **null**;

**private** ArrayList<CardLabel> hand=**null**; // @jve:decl-index=0:

**private** JPanel MainPanel = **null**;

**private** JButton deckButton = **null**;

**private** CardLabel openCardLabel = **null**;

**private** JLabel userCardLabel = **null**;

**private** Manager manager=**null**;

**private** **boolean** isTurn=**true**;

**private** **void** initialize() {

**this**.setContentPane(getJContentPane());

**this**.setTitle("JFrame");

**this**.addWindowListener(**new** java.awt.event.WindowAdapter() {

**public** **void** windowClosing(java.awt.event.WindowEvent e) {

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

System.*exit*(0);

}

});

**this**.pack();

hand=**new** ArrayList<CardLabel>();

manager=**new** Manager(**this**,3);

enableCard();

}

**public** OneCardGUI() {

**super**();

initialize();

//Card c=new Card(0,14);

//card.setIcon(c.getFrontImage());

}

**private** JPanel getJContentPane() {

**if** (jContentPane == **null**) {

jContentPane = **new** JPanel();

jContentPane.setLayout(**new** BorderLayout());

jContentPane.add(getMainPanel(), BorderLayout.*CENTER*);

}

**return** jContentPane;

}

**private** JPanel getMainPanel() {

**if** (MainPanel == **null**) {

userCardLabel = **new** JLabel();

userCardLabel.setBounds(**new** Rectangle(31, 257, 175, 23));

userCardLabel.setText("사용자가 가지고 있는 카드");

MainPanel = **new** JPanel();

MainPanel.setLayout(**null**);

MainPanel.setPreferredSize(**new** Dimension(800, 560));

MainPanel.add(getDeckButton(), **null**);

MainPanel.add(getOpenCardLabel(), **null**);

MainPanel.add(userCardLabel, **null**);

}

**return** MainPanel;

}

**public** CardLabel getOpenCardLabel()

{

**if**(openCardLabel==**null**){

openCardLabel = **new** CardLabel(**new** Card(1,2));

openCardLabel.setText("JLabel");

openCardLabel.setSize(**new** Dimension(150, 200));

openCardLabel.setLocation(**new** Point(420, 40));

}

**return** openCardLabel;

}

**private** JButton getDeckButton() {

**if** (deckButton == **null**) {

deckButton = **new** JButton();

deckButton.setBounds(**new** Rectangle(225, 35, 160, 210));

ImageIcon icon=**new** ImageIcon(getClass().getResource("/image/back\_up.png"));

icon.setImage(icon.getImage().getScaledInstance(150, 200, Image.*SCALE\_SMOOTH*));

deckButton.setIcon(icon);

icon=**new** ImageIcon(getClass().getResource("/image/back\_dn.png"));

icon.setImage(icon.getImage().getScaledInstance(150, 200, Image.*SCALE\_SMOOTH*));

deckButton.setSelectedIcon(icon);

icon=**new** ImageIcon(getClass().getResource("/image/back.png"));

icon.setImage(icon.getImage().getScaledInstance(150, 200, Image.*SCALE\_SMOOTH*));

deckButton.setRolloverIcon(icon);

deckButton.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** actionPerformed(java.awt.event.ActionEvent e) {

manager.addCard(**null**);

}

});

}

**return** deckButton;

}

**void** repaintUserCard() {

//System.out.println(hand.size());

Collections.*sort*(hand);

Rectangle r=**new** Rectangle(10+30\*hand.size(),330,150,200);

**for**(**int** i=hand.size()-1;i>=0;i--)

{

CardLabel temp=hand.get(i);

**if**(MainPanel.getComponentZOrder(temp)!=-1)

MainPanel.remove(temp);

temp.setBounds(r);

r.translate(-30, 0);

MainPanel.add(temp,**null**);

**if**(temp.getMouseListeners().length==0)

temp.addMouseListener(**this**);

}

MainPanel.repaint();

//enableCard();

}

**class** moveCard **extends** Thread

{

CardLabel cl;

Point p;

**int** count =10;

**public** moveCard(CardLabel cl,Point p)

{

**this**.cl=cl;

**this**.p=**new** Point((p.x-cl.getLocation().x)/count ,(p.y-cl.getLocation().y)/count);

}

**public** **void** run()

{

**while**(p.equals(cl.getLocation()))

{

Point p1=cl.getLocation();

p1.translate(p.x, p.y);

cl.setLocation(p1);

}

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

{

Point p1=cl.getLocation();

p1.translate(p.x, p.y);

cl.setLocation(p1);

**try** {

*sleep*(10);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

MainPanel.remove(cl);

manager.addCardLabel(hand.remove(hand.indexOf(cl)));

//repaintUserCard();

//openCardLabel.setCard(cl.getCard());

//repaint();

}

}

**public** **void** mouseClicked(MouseEvent e) {

}

**public** **void** mouseEntered(MouseEvent e) {

Point p =((JLabel)e.getSource()).getLocation();

p.y=300;

((JLabel)e.getSource()).setLocation(p);

}

**public** **void** mouseExited(MouseEvent e) {

Point p =((JLabel)e.getSource()).getLocation();

p.y=330;

((JLabel)e.getSource()).setLocation(p);

}

@Override

**public** **void** mousePressed(MouseEvent e) {

}

@Override

**public** **void** mouseReleased(MouseEvent e) {

System.*out*.println(isTurn);

**if**(isTurn&&((CardLabel)e.getSource()).isEnabled()){

**new** moveCard((CardLabel)e.getSource(),openCardLabel.getLocation()).start();

//isTurn=false;

}

}

**private** **void** enableCard()

{

Card temp\_card;

**if**(manager!=**null**){

temp\_card=manager.getOpenCard();

**int** state=manager.getState();

Iterator<CardLabel> ite=hand.iterator();

**while**(ite.hasNext())

{

CardLabel temp=ite.next();

**if**(state>1){

temp.setEnabled(temp.getCard().isHighPriority(temp\_card));

}**else**{

temp.setEnabled(temp.getCard().isSameRank(temp\_card));

}

}

}

}

@Override

**public** **void** addCard(Card c1) {

hand.add(**new** CardLabel(c1));

repaintUserCard();

}

@Override

**public** **boolean** isEmpty() {

**return** hand.isEmpty();

}

@Override

**public** **void** setTurn(**boolean** isTurn) {

**this**.isTurn=isTurn;

**if**(isTurn){

enableCard();

}

}

**public** String toString()

{

**return** "유저";

}

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

SwingUtilities.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**try** {

UIManager.*setLookAndFeel*("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (InstantiationException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IllegalAccessException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (UnsupportedLookAndFeelException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

OneCardGUI thisClass = **new** OneCardGUI();

thisClass.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

thisClass.setVisible(**true**);

}

});

}

@Override

**public** **int** suitChange() {

**return** JOptionPane.*showOptionDialog*(**null**, "Select a Card Suit",

"Card Game WAR",

JOptionPane.*YES\_NO\_CANCEL\_OPTION*,

JOptionPane.*QUESTION\_MESSAGE*, **null**,

**new** String[]{"클로버","다이아몬드","하트","스페이드"}, "스페이드")+1;

}

}

-Interface Player.java

**package** player;

**import** data.\*;

**public** **interface** Player {

**public** **void** addCard(Card c1);

//public boolean checkCard(Card c1);

**public** **void** setTurn(**boolean** isTurn);

**public** **boolean** isEmpty();

**public** **int** suitChange();

}

-Class Computer.java

**package** player;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.Random;

**import** main.Manager;

**import** data.Card;

**public** **class** Computer **implements** Player {

Manager manager=**null**;

ArrayList<Card> hand;

String name;

**public** Computer(Manager manager,String name)

{

**this**.manager=manager;

hand=**new** ArrayList<Card>();

**this**.name=name;

}

**private** **boolean** checkCard(Card c1) {

Card temp=manager.getOpenCard();

**int** state=manager.getState();

**if**(state>1)

{

**return** c1.isHighPriority(temp);

}**else**{

**return** c1.isSameRank(temp);

}

}

**public** **void** setTurn(**boolean** isTurn) {

**if**(isTurn)

{

dropCard();

}

// **TODO** Auto-generated method stub

}

**public** **void** addCard(Card c1) {

hand.add(c1);

// **TODO** Auto-generated method stub

}

**private** **void** dropCard()

{

/\*try {

sleep(1000);

} catch (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}\*/

ArrayList<Card> temp=**new** ArrayList<Card>();

Iterator <Card>ite=hand.iterator();

**while**(ite.hasNext())

{

Card c1=ite.next();

//System.out.println(c1);

**if**(checkCard(c1))

{

System.*out*.print(c1+", ");

temp.add(c1);

}

}

System.*out*.println();

**if**(temp.size()>1)

{

manager.addCard(hand.remove(hand.indexOf(temp.get(**new** Random().nextInt(temp.size()-1)))));

}**else** **if**( temp.size()==1){

manager.addCard(hand.remove(hand.indexOf(temp.get(0))));

}**else**{

manager.addCard(**null**);

}

}

**public** **boolean** isEmpty() {

// **TODO** Auto-generated method stub

**return** **false**;

}

**public** String toString()

{

**return** name;

}

@Override

**public** **int** suitChange() {

// **TODO** Auto-generated method stub

**return** **new** Random().nextInt(3)+1;

}

}

Utility Package

-Class CircleLinkedList.java

**package** utility;

**import** player.Player;

**public** **class** CircleLinkedList <T>{

**private** Node<T> headNode;

**private** **int** count=0;

**public** **void** add(T element)

{

**if**(count==0)

{

headNode=**new** Node<T>(element);

}**else** **if**(count ==1) {

**new** Node<T>(element, headNode, headNode);

}**else**{

**new** Node<T>(element,headNode.getPrev(),headNode);

}

count++;

}

**public** Node<T> remove(Player player)

{

**return** **null**;

}

**public** Node<T> getHead()

{

**return** headNode;

}

**public** **int** size() {

// **TODO** Auto-generated method stub

**return** count;

}

}

-Class Node.java

**package** utility;

**public** **class** Node <T>{

**private** Node <T> next;

**private** Node <T> prev;

**private** T element;

Node(T element,Node <T> prev,Node <T> next)

{

prev.setNext(**this**);

**this**.prev=prev;

next.setPrev(**this**);

**this**.next=next;

**this**.element=element;

}

Node(T element)

{

**this**.element=element;

}

**public** T getElement()

{

**return** element;

}

**public** **void** setElement(T element)

{

**this**.element=element;

}

**public** Node <T> getNext()

{

**return** next;

}

**public** Node <T> getPrev()

{**return** prev; }

**public** **void** setNext(Node <T> node)

{next=node;}

**public** **void** setPrev(Node <T> node)

{

prev=node;

}

}