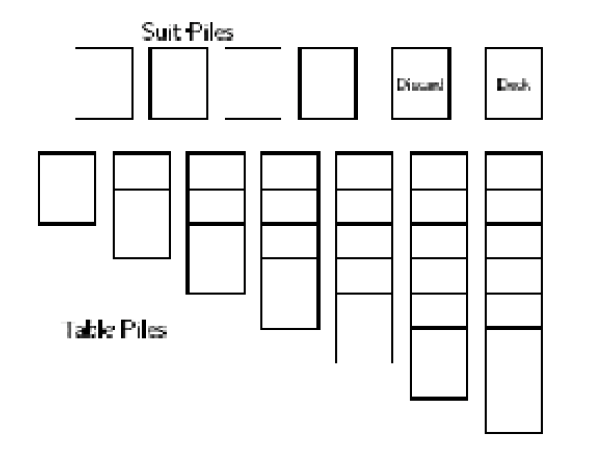
solitaire纸牌游戏实验报告

1. **实验目的：**

理解面向对象封装、继承、抽象类、抽象方法、多态、动态绑定等概念

1. **实验内容：**

单人纸牌游戏，牌桌上有7个堆共28张牌，第一堆1张牌，第二堆2张，。。。第7堆7张，每一堆的第一张牌朝上，其他朝下。牌桌上还有4个suitpiles，一个deck card堆和一个discard card堆，布局如下（参考windows的纸牌游戏）



设计一个简单的CardGames程序，运用面向对象封装、继承、抽象类、抽象方法、多态、动态绑定等概念。

1. **实验环境：**

Windows10

JDK1.8

IDEA

1. **实验步骤：**
2. 需求分析

单人纸牌游戏，牌桌上有7个堆共28张牌，第一堆1张牌，第二堆2张……第7堆7张，每一堆的第一张牌朝上，其他朝下。牌桌上还有4个suitpiles，一个deck card堆和一个discard card堆。初始时四个suitpiles和一个deck card堆为空，discard card堆中有24张牌且全部背面朝下。

若要操作方便，则需加入图形界面。

1. 类的设计（未界面说明）

*包CardMethod*

Card类：定义牌的一些属性，包括花色、数值、是否正面朝上等。

定义常量表示花色的角标，定义两个数组分别维护牌的花色和数值。

CardPile类：十三个牌堆的父类，用一个ArrayList数组维护各堆中的牌。

定义所有堆中的共同操作，如判断牌堆是否为空、添加牌，移除牌，获得堆顶牌等。

DeckPile类：继承CardPile，由于添加牌时需设置背面朝上，所以重写父类中的addCard方法。

由于父类中的getCardIndex方法为抽象方法，而该方法在该队中没有作用，故实现时只返回-1即可。

DisCardPile类：继承CardPile，由于添加牌和移除牌时需设置堆顶牌正面朝上，所以重写父类中的addCard方法和remove方法。

添加一个方法clear()，用于deckCard堆中的牌全部翻转到该堆中，下一轮翻转开始时，需将该堆中所有的牌移动至deckCard堆，而清空该堆。

实现getCardIndex方法，在数组中搜索传入的牌，若找到返回其index。

SuitPile类：继承CardPile，由于添加牌和移除牌时需设置堆顶牌正面朝上，所以重写父类中的addCard方法和remove方法。

实现getCardIndex方法

TablePile类：继承CardPile，由于移除牌时需设置堆顶牌正面朝上，所以重写父类中的remove方法。

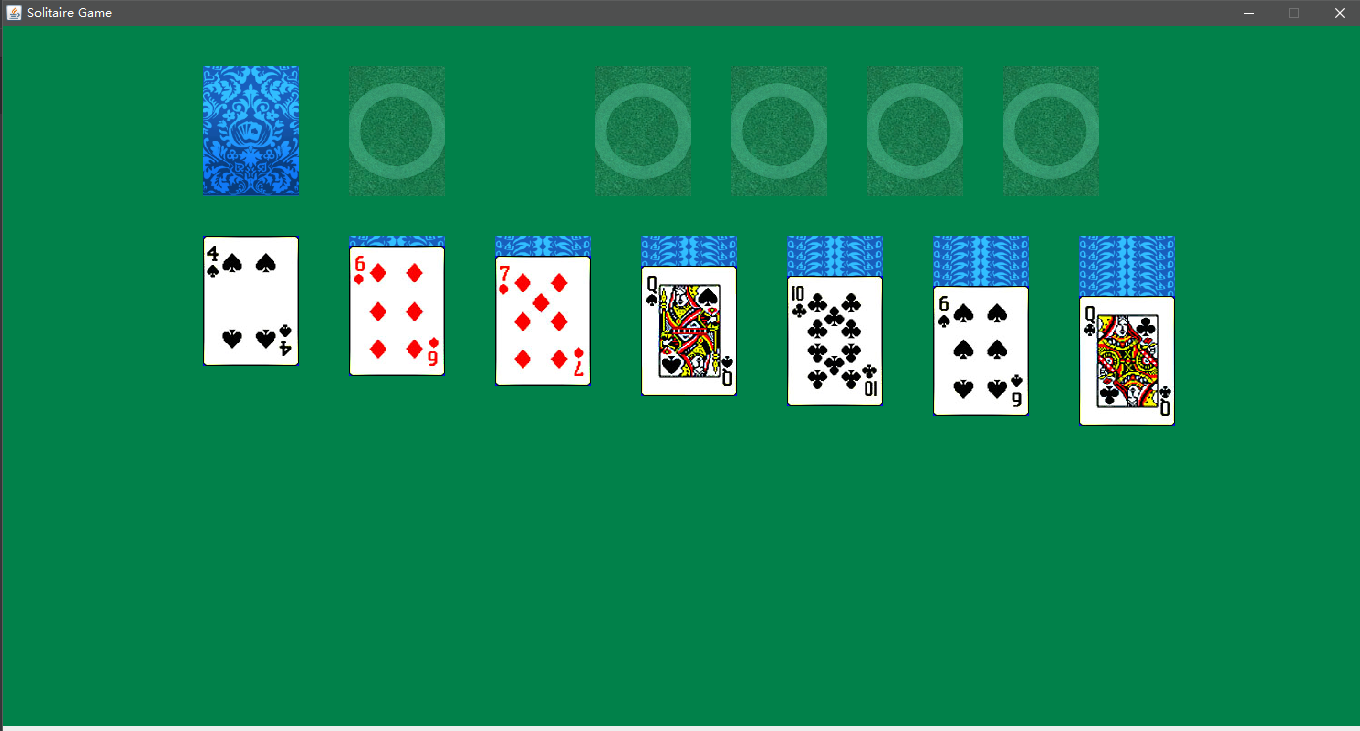
由于移向该堆的牌均为正面朝上，所以无需重写父类中的addCard方法。

实现第二个实现getCardIndex方法时加入for循环，获得多张牌的索引，用来移动多张已排好的牌。

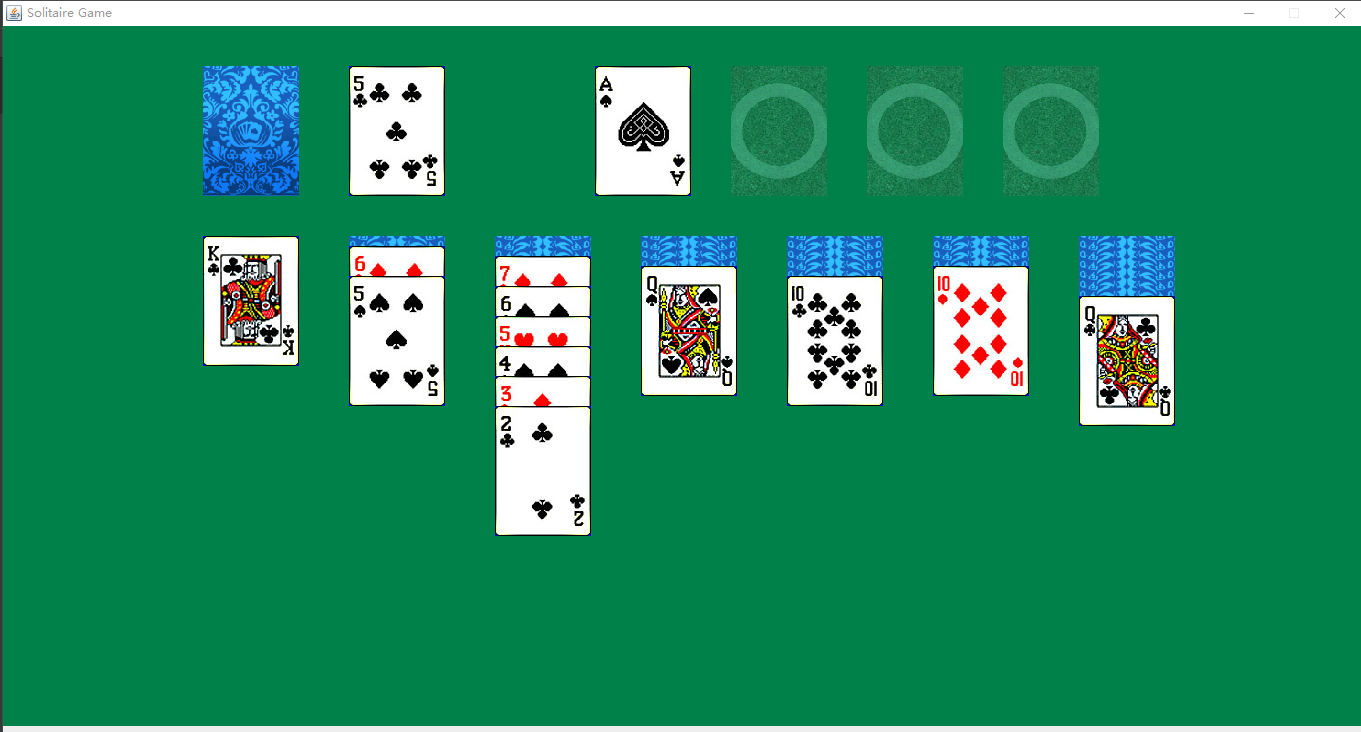
游戏界面的实现：将Card添加到Label，各堆运用JLayeredPane实现组件的重叠，为各组件安装mouseClicked鼠标监听。

1. **实验结果：**

程序初始界面：



移动几张牌之后



1. **遇到的问题：**

牌和堆都用ArrayList数组来维护，所以必须明确牌在数组中的索引。不论是添加牌还是移除牌，都是对牌堆第一个元素的操作，移动牌时，有时需要移动多张已经排好的牌，故采取一开始的那个getCardIndex方法不能实现，所以需重载此方法，然后TablePile在实现该方法时使用for循环以获得多张牌的索引。

由于各堆类中均有添加删除等方法，故可以添加一个父类以简化代码。

1. **心得体会：**

通过本实验基本对使用面向对象思想编程的基本思路有了一定的了解。首先要本着一切皆对象的基本原则。程序中能够分离出一个类的一定要分离出来，这对后来代码的修改与完善有很大的帮助。代码复用的一个很好很常见的解决办法是利用抽象，可以将重复代码抽象出来，形成抽象类，并将不同的部分利用延迟方法，延迟到具体的类进行实现。

1. **代码展示：**

**\*Filename：Card**

package solitaire.card;  
  
import java.awt.\*;  
import java.io.File;  
import java.io.IOException;  
  
import javax.imageio.ImageIO;  
  
public class Card {  
   
//final public static int width = 96;  
//final public static int height = 130;  
  
 final public static int *width* = 96;  
 final public static int *height* = 130;  
  
//四种花色  
final public static int *heart* = 0;  
final public static int *spade* = 1;  
final public static int *diamond* = 2;  
final public static int *club* = 3;  
   
private boolean faceup; //正反面  
private int num;  
private int type;  
private int x;  
private int y;  
  
public Card (int num, int type) {   
 this.num = num;   
 this.type = type;   
 this.faceup = false;   
 }  
public Color getColor() {  
 if (isFaceup()){  
 if (getType() == *heart* || getType() == *diamond*)  
 return Color.*red*;  
 else  
 return Color.*black*;  
 }  
 return Color.*yellow*;  
 }  
public void draw (Graphics g) {  
   
   
 Image image= null;  
 if (isFaceup()) {  
 try {   
// String picture = "picture/"+this.type+"-"+this.num+".png";  
 String picture = "poker/"+this.type+"-"+this.num+".jpg";  
 image = ImageIO.*read*(new File(picture));   
 } catch (IOException e) {  
  
 e.printStackTrace();  
 System.*out*.println("class-draw-if");  
 }   
 }  
 else {  
 try {   
 String picture = "poker/back.png";  
 image = ImageIO.*read*(new File(picture));  
 } catch (IOException e) {  
  
 e.printStackTrace();  
 System.*out*.println("class-draw-else");  
 }   
   
 }  
 g.drawImage(image, getX(), getY(), Card.*width*, Card.*height*, null);  
 }  
public boolean isFaceup() {  
 return faceup;  
}  
public void setFaceup(boolean faceup) {  
 this.faceup = faceup;  
}  
public int getNum() {  
 return num;  
}  
public void setNum(int num) {  
 this.num = num;  
}  
public int getType() {  
 return type;  
}  
public void setType(int type) {  
 this.type = type;  
}  
public int getX() {  
 return x;  
}  
public void setX(int x) {  
 this.x = x;  
}  
public int getY() {  
 return y;  
}  
public void setY(int y) {  
 this.y = y;  
}  
  
  
}

**\*Filename：Game**

package solitaire.game;  
  
  
import java.util.\*;  
  
import solitaire.pile.\*;  
import solitaire.card.\*;  
  
public class Game {  
 static public ArrayList<Card> *allCard*;  
 static public CardPile *allPiles*[];  
 static public DeckPile *deckPile*;//发牌堆  
 static public DiscardPile *discardPile*;//弃牌堆  
 static public TablePile *tablePile*[];//桌面上的牌堆  
 static public SuitPile *suitPile*[]; //4个存放可以匹配的牌堆  
 static public MoveCardPile *moveCard*;  
 static public Stack<UsedPile> *usedPile*;//用过的牌堆，用于撤销  
  
 static {  
 *init*();  
 }  
  
 static void init(){  
 //初始化所有牌  
 *allCard* = new ArrayList<Card>();  
 for (int i = 0; i < 4; i++)  
 for (int j = 0; j <= 12; j++)  
 *allCard*.add(new Card(j, i));  
 Random generator = new Random();  
 for (int i = 0; i < 52; i++) {  
 int j = Math.*abs*(generator.nextInt() % 52);  
 // 交换两张牌的值  
 Card temp = *allCard*.get(i);  
 *allCard*.set(i, *allCard*.get(j));  
 *allCard*.set(j, temp);  
 }  
 //初始化各个牌堆  
 *allPiles* = new CardPile[13];  
 *suitPile* = new SuitPile[4];  
 *tablePile* = new TablePile[7];  
  
 *allPiles*[0] = *deckPile* = new DeckPile(200, 40);  
 *allPiles*[1] = *discardPile* = new DiscardPile(200 + Card.*width* + 50, 40);  
 for (int i = 0; i < 4; i++)  
 *allPiles*[2 + i] = *suitPile*[i] = new SuitPile(200 + Card.*width* + 50 + Card.*width* + 150 + (40 + Card.*width*) \* i, 40);  
 for (int i = 0; i < 7; i++)  
 *allPiles*[6 + i] = *tablePile*[i] = new TablePile(200 + (50 + Card.*width*) \* i, 40 + Card.*height* + 40, i);  
 for (int i = 0; i < 7; i++) {  
 ArrayList<Card> al = new ArrayList<Card>();  
 for (int j = 0; j < *tablePile*[i].getCardNum(); j++) {  
 al.add(*allCard*.remove(*allCard*.size() - 1));  
 }  
 *tablePile*[i].addCard(al);  
 *tablePile*[i].setCardNum(*tablePile*[i].getNotFlipNum() + 1);  
 *tablePile*[i].top().setFaceup(true);  
 }  
 int rest = *allCard*.size();  
 for (int i = 0; i < rest; i++) {  
 *deckPile*.addCard(*allCard*.remove(*allCard*.size() - 1));  
 }  
 *moveCard* = new MoveCardPile();  
 *usedPile*=new Stack<>();  
 }  
  
 public static void transferFromDiscardToDeck() { //把弃牌堆重新转化为发牌堆  
  
 while (!(*discardPile*.isEmpty())) {  
 Card card = *discardPile*.pop();  
 card.setFaceup(false);  
 *deckPile*.addCard(card);  
  
 }  
 }  
  
 public static TablePile findTablePile(int x,int y){  
 for (int i=0;i<7;i++){  
 if (*tablePile*[i].includes(x,y))return *tablePile*[i];  
 }  
 return null;  
 }  
  
 public static SuitPile findSuitPile(int x,int y){  
 for (int i=0;i<4;i++){  
 if (*suitPile*[i].includes(x,y))return *suitPile*[i];  
 }  
 return null;  
 }  
  
 public static CardPile findLastPile(CardPile pile){  
 if (pile.getType()==CardPile.*TABLE\_PILE*){  
 for (int i=0;i<7;i++){  
 if (*tablePile*[i].isSame(pile))return *tablePile*[i];  
 }  
 }  
 else if(pile.getType()==CardPile.*SUIT\_PILE*){  
 for (int i=0;i<4;i++){  
 if (*suitPile*[i].isSame(pile))return *suitPile*[i];  
 }  
 }  
 else if(pile.getType()==CardPile.*DISCARD\_PILE*)return *discardPile*;  
 else if (pile.getType()==CardPile.*DECK\_PILE*)return *deckPile*;  
 return null;  
 }  
  
  
 public static void undo(){ //撤销上一步，可连续  
 if (*usedPile*.isEmpty()){  
// System.out.println("empty!!!!!!!");  
 return;  
 }  
  
 UsedPile pile=*usedPile*.pop();  
 if(pile.usedPile.getType()==CardPile.*TABLE\_PILE*){  
  
 TablePile tp=*findTablePile*(pile.nowX,pile.nowY);  
 if (tp!=null){  
 CardPile last=*findLastPile*(pile.lastPile);  
 if (last!=null){  
// System.out.println("从tablePile 撤回到"+last.getType()+" pile.size: "+pile.thePile.size());  
 int count=pile.thePile.size();  
 Stack<Card> stack=new Stack<>();  
 while (count>0){  
 count--;  
 Card t=tp.pop();  
// System.out.println(t.getNum()+" undo "+t.getType());  
 stack.push(t);  
  
 }  
 if (last.getType()==CardPile.*TABLE\_PILE*) {  
 if (!last.isEmpty()) {  
 if (((TablePile)last).getNotFlipNum()==((TablePile) last).getCardNum()-1&&!pile.isIsup()){  
 last.top().setFaceup(false);  
 ((TablePile) last).setNotFlipNum(((TablePile) last).getNotFlipNum() + 1);  
 }  
 }  
 }else if(last.getType()==CardPile.*DISCARD\_PILE*){  
 *discardPile*.addCard(stack.pop());  
// System.out.println("discard!!!!!!!!!!!!!");  
 return;  
 }  
  
  
 while (!stack.empty()){  
 last.addSingleCard(stack.pop());  
// System.out.println(last.top().getNum()+" top "+last.top().getType());  
 }  
 }  
  
  
 }  
  
  
 }  
 else if(pile.usedPile.getType()==CardPile.*DECK\_PILE*){  
// System.out.println("DeckPile!!!!");  
 }  
 else if(pile.usedPile.getType()==CardPile.*DISCARD\_PILE*){//撤销发牌  
 Card temp=*discardPile*.pop();  
 if (temp!=null) {  
 temp.setFaceup(false);  
 *deckPile*.addCard(temp);  
 }  
 }  
 else if(pile.usedPile.getType()==CardPile.*SUIT\_PILE*){  
 if (pile.lastPile.getType()==CardPile.*DISCARD\_PILE*){  
 *discardPile*.addCard(*findSuitPile*(pile.nowX,pile.nowY).pop());  
 }  
 else if(pile.lastPile.getType()==CardPile.*TABLE\_PILE*){  
// System.out.println("table!!!!!!!!!");  
 if(*findTablePile*(pile.x,pile.y).getNotFlipNum()==*findTablePile*(pile.x,pile.y).getCardNum()-1){  
 *findTablePile*(pile.x,pile.y).top().setFaceup(false);  
 *findTablePile*(pile.x,pile.y).setNotFlipNum(*findTablePile*(pile.x,pile.y).getNotFlipNum()+1);  
 }  
 *findTablePile*(pile.x,pile.y).addSingleCard(*findSuitPile*(pile.nowX,pile.nowY).pop());  
 }  
 else if(pile.lastPile.getType()==CardPile.SUIT\_PILE){  
 findSuitPile(pile.x,pile.y).addCard(findSuitPile(pile.nowX,pile.nowY).pop());  
 }  
  
 }  
  
 }  
  
  
  
 public static boolean testDeckPile(int x, int y) {  
 int selectNum = deckPile.select(x, y);//未知牌堆的数量  
 if (selectNum >= 0) {  
 Card c=deckPile.pop();  
 discardPile.addCard(c);//发牌一次  
  
 //记录到使用牌堆  
  
 UsedPile up=new UsedPile(x,y,deckPile);  
 up.setUsedPile(discardPile,discardPile.x,discardPile.y);  
 up.addCard(c);  
 usedPile.push(up);  
  
 return true;  
 } else if (selectNum == -2) {  
 Game.transferFromDiscardToDeck();  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 //选中结果牌堆  
 public static boolean testSuitPile(int x, int y) {//todo  
 for (int i = 0; i < 4; i++) {  
 int selectNum = suitPile[i].select(x, y);  
 if (selectNum >= 0) {  
// System.out.println("choose pile "+i);  
 moveCard.clear();  
 Card t = suitPile[i].pop();  
// System.out.println(t.getNum()+" pop "+t.getType());  
 moveCard.addCard(t);  
// SuitPile temp=new SuitPile(x,y);  
// temp.addCard(t);  
 moveCard.setFromPile(suitPile[i]);  
  
 //记录  
 UsedPile up=new UsedPile(x,y,suitPile[i]);  
 up.addCard(t);  
 usedPile.push(up);  
  
  
 return true;  
 }  
 }  
  
 return false;  
 }  
  
 public static boolean testDisCardPile(int x, int y) {  
 int selectNum = discardPile.select(x, y);  
 if (selectNum >= 0) {  
 moveCard.clear();  
 Card c=discardPile.pop();  
 moveCard.addCard(c);  
 moveCard.setFromPile(discardPile);  
  
 //记录  
 UsedPile up=new UsedPile(x,y,discardPile);  
 up.addCard(c);  
 usedPile.push(up);  
  
  
  
 return true;  
 } else if (selectNum == -2) {  
 } else {  
 }  
 return false;  
 }  
  
 public static boolean testTablePile(int x, int y) {  
 boolean isDrag = false;  
 for (int i = 0; i < tablePile.length; i++) {  
 int selectNum = tablePile[i].select(x, y);  
 if (selectNum >= 0) {  
 moveCard.clear();  
  
 int num = tablePile[i].getCardNum();  
 UsedPile up=new UsedPile(x,y,tablePile[i]);  
 for (int j = selectNum; j < num; j++) {  
  
 Card temp=tablePile[i].pop();  
 up.addCard(temp);  
 moveCard.addCard(temp);  
 }  
 //  
 if (selectNum+tablePile[i].getNotFlipNum()<tablePile[i].getCardNum())up.setIsup(true);//记录是否撤回时该朝上  
 moveCard.setFromPile(tablePile[i]);  
 usedPile.push(up);  
  
 //System.out.println("moveCard\_size:"+moveCard.size() );  
 return true;  
 } else {  
 // System.out.println("tablePile["+i+"]\_selectNum:"+-1);  
 }  
 //System.out.println("tablePile["+i+"]\_size:"+tablePile[i].thePile.size() );  
 }  
  
 return isDrag;  
 }  
  
 public static boolean isCanAddToSuitPile(int x, int y) {  
 if (moveCard.size() == 1) {  
 for (int i = 0; i < 4; i++) {  
 if (suitPile[i].includes(x, y)) {  
 if (suitPile[i].isCanAdd(moveCard.getCard())) {  
 suitPile[i].addCard(moveCard.removeCard());  
 usedPile.peek().setUsedPile(suitPile[i],x,y);//记录  
 return true;  
 }  
  
 }  
 }  
 }  
  
 return false;  
 }  
  
 public static boolean isCanAddtoTablePile(int x, int y) {  
 for (int i = 0; i < 7; i++) {  
 if (tablePile[i].includes(x, y)) {  
 if (tablePile[i].hashCode() != moveCard.getFromPile().hashCode()) {  
 if (tablePile[i].isCanAdd(moveCard.getCard())) {  
 tablePile[i].addCard(moveCard.clear());  
 usedPile.peek().setUsedPile(tablePile[i],x,y);//记录  
 return true;  
 }  
 }  
 }  
 }  
 return false;  
 }  
  
 public static boolean isWin() { //判断是否胜利  
 for (int i = 0; i < 4; i++) {  
 if (suitPile[i].size() < 13) return false;  
 }  
 return true;  
 }  
  
  
 public static void refreshTablePile() {  
  
 for (int i = 0; i < 7; i++) {  
 if (tablePile[i].top() != null)  
 if (!(tablePile[i].top().isFaceup())) {  
 tablePile[i].top().setFaceup(true);  
 tablePile[i].setNotFlipNum(tablePile[i].getNotFlipNum() - 1);  
 }  
 }  
 }  
  
 public static void returnToFromPile() {  
 if (moveCard.getFromPile() != null)  
 if (moveCard.getFromPile().hashCode() == discardPile.hashCode()) {  
 while (!(moveCard.isEmpty())) {  
 moveCard.getFromPile().addCard(moveCard.removeCard());  
 }  
 } else {  
 ArrayList<Card> temp = moveCard.clear();  
// System.out.println("else movecard:"+moveCard.getFromPile().getClass().toString()+" clear: "+temp.getClass().toString());  
 if (moveCard.getFromPile().getClass().equals(SuitPile.class)) {//回退结果牌堆的牌  
 Card t = temp.remove(0);  
// System.out.println(t.getNum()+" isnull? "+t.getType());  
 moveCard.getFromPile().addCard(t);  
  
 } else {  
 moveCard.getFromPile().addCard(temp);//  
// for (int i = 0; i < temp.size(); i++) {  
//// System.out.println("num:" + temp.get(i).getNum() + " type:" + temp.get(i).getType());  
// }  
 }  
 }  
  
  
 }  
}

**\*Filename：Main**

package solitaire.game;  
  
import javax.swing.\*;  
  
public class Main extends JFrame{  
 private Solitaire sp;  
 public Main(){  
 setSize(1366, 735);  
 setTitle("Solitaire Game");  
 setLayout(null);  
 setDefaultCloseOperation(*DISPOSE\_ON\_CLOSE*);  
 setVisible(true);  
 setResizable(false);  
  
 sp = new Solitaire();  
 add(sp);  
 }  
 public static void main(String[] args) {  
 Main main = new Main();  
 }  
  
}

**\*Filename：Solitaire**

package solitaire.game;  
  
import jdk.nashorn.internal.scripts.JO;  
  
import java.awt.\*;  
import java.awt.event.\*;  
import javax.swing.\*;  
  
public class Solitaire extends JPanel implements MouseListener, ActionListener,MouseMotionListener {  
   
 private boolean isDrag = false;  
 private int x;//mouse  
 private int y;//mouse  
 PopupMenu menu;  
  
  
 public Solitaire() {  
 setSize(1366, 700);  
 setLayout(null);  
 addMouseListener (this);  
 addMouseMotionListener(this);  
  
 menu=new PopupMenu();  
 MenuItem item=new MenuItem("undo");  
 MenuItem item2=new MenuItem("New game");  
 item.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 Game.*undo*();  
 repaint();  
 }  
 });  
  
 item2.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 Game.*init*();  
 repaint();  
  
 }  
 });  
 menu.add(item);  
 menu.add(item2);  
 add(menu);  
 }  
  
 @Override  
 protected void paintComponent(Graphics g) {  
  
 g.clearRect(0, 0, 1366, 768);   
 g.setColor(new Color(0x01814A));  
 g.fillRect(0, 0, 1366, 768);  
 for (int i = 0; i < 13; i++)   
 Game.*allPiles*[i].display(g);  
 Game.*moveCard*.display(g, x, y);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {}  
  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 this\_mousePressed(e);  
  
 }  
  
 void this\_mousePressed(MouseEvent e) {  
 int mods=e.getModifiers();  
//鼠标右键  
 if((mods&InputEvent.*BUTTON3\_MASK*)!=0){  
//弹出菜单  
 menu.show(this,e.getX(),e.getY());  
 }  
 }  
  
 @Override  
 public void mousePressed(MouseEvent e) {  
  
 x = e.getX();  
 y = e.getY();  
 isDrag = false;  
 boolean isSelect = false;  
 isSelect = Game.*testDeckPile*(x,y);  
 if(!isSelect){  
 isSelect = Game.*testDisCardPile*(x, y);  
 if(isSelect){  
// System.out.println("选中弃牌");  
 isDrag = true;  
 }  
 if(!isSelect){  
 isSelect = Game.*testTablePile*(x, y);  
 if(isSelect) {  
// System.out.println("选中桌面");  
 isDrag = true;  
 }  
 else{  
 Game.*testSuitPile*(x,y);  
// System.out.println("选中结果");  
 isDrag=true;  
 }  
 }  
   
 }  
 isDrag = false;  
 repaint();  
   
 }  
  
 @Override  
 public void mouseReleased(MouseEvent e) {  
  
  
  
 if(isDrag && Game.*moveCard*.size() > 0 ){  
 boolean isCanAdd =false;  
 isCanAdd = Game.*isCanAddToSuitPile*(x,y);  
 if(!isCanAdd)  
 isCanAdd = Game.*isCanAddtoTablePile*(x, y);  
   
 if(!isCanAdd ){  
 Game.*usedPile*.pop();  
 Game.*returnToFromPile*();  
 }  
 else  
 Game.*refreshTablePile*();  
 isDrag = false;  
 repaint();  
 }  
 else{  
 if(Game.*moveCard*.size() > 0) {  
 Game.*usedPile*.pop();//清除无效记录  
 Game.*returnToFromPile*();  
  
 }  
 repaint();  
 if (Game.*isWin*()){  
// int exi = JOptionPane.showConfirmDialog (null, "You Win !", null, JOptionPane.OK\_OPTION, JOptionPane.QUESTION\_MESSAGE);  
// if (exi == JOptionPane.YES\_OPTION)  
// {  
// System.exit (0);  
// }  
 Toolkit.*getDefaultToolkit*().beep();  
 JOptionPane.*showMessageDialog*(null,"You win !","",JOptionPane.*INFORMATION\_MESSAGE*);  
 System.*exit*(0);  
 }  
  
 }  
 }  
  
 @Override  
 public void mouseEntered(MouseEvent e) {}  
  
 @Override  
 public void mouseExited(MouseEvent e){}  
  
 @Override  
 public void mouseDragged(MouseEvent e){  
  
 isDrag = true;  
 x = e.getX();  
 y = e.getY();  
 repaint();  
   
 }  
  
 @Override  
 public void mouseMoved(MouseEvent e) {}  
   
  
}

**\*Filename：CardPile**

package solitaire.pile;  
  
import java.awt.\*;  
import java.io.File;  
import java.io.IOException;  
import java.util.Stack;  
import java.util.EmptyStackException;  
  
import javax.imageio.ImageIO;  
  
import solitaire.card.Card;  
  
public class CardPile {  
  
 public static final int *TABLE\_PILE*=1;  
 public static final int *SUIT\_PILE*=2;  
 public static final int *DECK\_PILE*=3;  
 public static final int *DISCARD\_PILE*=4;  
 public int type;  
 public int x;  
 public int y;  
 public Stack<Card> thePile;  
  
 public CardPile(int xl, int yl) {  
 x = xl;  
 y = yl;  
 thePile = new Stack<Card>();  
  
 }  
  
 public void setType(int type){  
 this.type=type;  
 }  
  
 public int getType(){  
 return this.type;  
 }  
  
 public Card top() { //牌堆的第一张牌  
 if (!(thePile.empty()))  
 return (Card) thePile.peek();  
 else  
 return null;  
 }  
  
 public boolean isEmpty() {  
 return thePile.empty();  
 }  
  
 public Card pop() {  
 try {  
 return (Card) thePile.pop();  
  
 } catch (EmptyStackException e) {  
 return null;  
 }  
 }  
  
 public Card peek(){  
 if (thePile.empty())return null;  
 return (Card)thePile.peek();  
 }  
  
  
 public boolean includes(int tx, int ty) { //判断位置是否在合适的区域内  
 return this.x <= tx && tx <= this.x + Card.*width* &&  
 this.y <= ty && ty <= this.y + Card.*height*;  
 }  
  
 public boolean isSame(CardPile pile){//判断两个牌堆是否同一个牌堆  
 return this.x<=pile.x&&pile.x<=this.x+Card.*width*&&this.y<=pile.y&&pile.y<=this.y+Card.*height*;  
 }  
  
 public int select(int tx, int ty) {  
 if (includes(tx, ty)) {  
 if (isEmpty())  
 return -2;  
 else  
 return thePile.size() - 1;  
 } else  
 return -1;  
 }  
  
 public void addCard(Object card) {  
 thePile.push((Card) card);  
 }  
  
 public void addSingleCard(Object card){  
  
 }  
  
 public boolean isCanAdd(Card card) {  
 return false;  
 }  
  
 public void display(Graphics g) {  
  
 if (isEmpty()) {  
 Image image = null;  
 try {  
 String picture = "poker/0.png";  
 image = ImageIO.*read*(new File(picture));  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*out*.println("class-CardPile-display-if(isEmpty())");  
 }  
 g.drawImage(image, this.x, this.y, Card.*width*, Card.*height*, null);  
 } else {  
if(top()==null){  
 System.*out*.println( this.getClass().toString());  
 System.*out*.println(isEmpty());  
}  
 top().setX(x);  
 top().setY(y);  
 top().draw(g);  
  
 }  
 }  
  
}

**\*Filename：DeckPile**

package solitaire.pile;  
  
  
public class DeckPile extends CardPile {  
  
 public DeckPile (int x, int y) {  
   
 super(x, y);  
 this.setType(*DECK\_PILE*);  
   
 }  
  
   
  
}

**\*Filename：DisCardPile**

package solitaire.pile;  
  
import solitaire.card.Card;  
  
  
  
public class DiscardPile extends CardPile {  
   
 public DiscardPile (int x, int y) {   
  
 super (x, y);  
 this.setType(*DISCARD\_PILE*);  
 }  
  
 public void addCard (Object card){  
 Card cards = (Card)card;  
 if (!(cards.isFaceup()))  
 cards.setFaceup(true);  
 thePile.push(cards);  
 }  
   
}

**\*Filename：MoveCardPile**

package solitaire.pile;  
  
import java.awt.Graphics;  
import java.util.ArrayList;  
  
import solitaire.card.Card;  
  
public class MoveCardPile { //当前选中的牌所在的牌堆  
 private ArrayList<Card> cardList ;  
 private CardPile fromPile;  
 private int x;  
 private int y;  
 final public static int *separation* = 30;  
 public MoveCardPile(){  
 cardList = new ArrayList<Card>();  
   
 }  
 public int size(){  
 return cardList.size();  
 }  
 public boolean isEmpty(){  
 return cardList.isEmpty();  
 }  
 public void addCard(Card card){  
 cardList.add(0, card);  
 }  
 public Card getCard(){  
 if(cardList.size()>0)  
 return cardList.get(0);  
 else  
 return null;  
 }  
 public Card removeCard(){  
 if(cardList.size()>0)  
 return cardList.remove(0);  
 else  
 return null;  
   
 }  
  
  
  
 public ArrayList<Card> clear(){  
 ArrayList<Card> list = cardList;  
 cardList = new ArrayList<Card>();  
 return list;  
 }  
 public void display(Graphics g,int tx,int ty){  
 x = tx - Card.*width*/2;  
 y = ty - Card.*height*/2;  
 int localy = y;  
 for (int i = 0;i < cardList.size();i++) {  
 Card aCard = (Card) cardList.get(i);  
 aCard.setX(x);  
 aCard.setY(localy);  
 if(!(aCard.isFaceup()))  
 aCard.setFaceup(true);  
 aCard.draw(g);  
 localy += *separation*;  
   
 }  
 }  
   
public ArrayList<Card> getCardList() {  
 return cardList;  
}  
public void setCardList(ArrayList<Card> cardList) {  
 this.cardList = cardList;  
}  
public CardPile getFromPile() {  
 return fromPile;  
}  
public void setFromPile(CardPile fromPile) {  
 this.fromPile = fromPile;  
}  
  
public int getX() {  
 return x;  
}  
public void setX(int x) {  
 this.x = x;  
}  
public int getY() {  
 return y;  
}  
public void setY(int y) {  
 this.y = y;  
}  
   
}

**\*Filename：SuitPile**

package solitaire.pile;  
  
import solitaire.card.Card;  
  
import java.util.ArrayList;  
  
  
public class SuitPile extends CardPile {  
  
 public SuitPile (int x, int y) {  
 super(x, y);  
 this.setType(*SUIT\_PILE*);  
 }  
  
 @Override  
 public boolean isCanAdd(Card card) {  
  
 if (isEmpty())  
 return card.getNum() == 0;  
 Card topCard = top();  
 return (card.getType() == topCard.getType()) &&  
 (card.getNum() == topCard.getNum() + 1);  
   
 }  
  
 public int size(){  
 return thePile.size();  
 }  
  
 @Override  
 public void addCard(Object card) {  
 //solitaire.card.Card cannot be cast to java.util.ArrayList  
  
// ArrayList<Card> cardList = (ArrayList<Card>)card;  
  
 thePile.push((Card)card);  
 }  
  
 public void addSingleCard(Object card){  
// if (!thePile.empty())  
// for (int i=0;i<thePile.size();i++){  
// Card t=thePile.pop();  
// System.out.println(t.getNum()+" thepile "+t.getType());  
// }  
 ArrayList<Card> cardList = (ArrayList<Card>)card;  
 for (int i=0;i<cardList.size();i++){  
 System.*out*.println(cardList.get(i).getNum()+" "+cardList.size()+" i:"+i+" "+cardList.get(i).getType());  
 }  
 thePile.push(cardList.get(0));  
 }  
  
  
}

**\*Filename：TablePile**

package solitaire.pile;  
  
  
import java.awt.Graphics;  
import java.awt.Image;  
import java.io.File;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.Enumeration;  
  
import javax.imageio.ImageIO;  
  
import solitaire.card.Card;  
  
public class TablePile extends CardPile {  
 private int notFlipNum;  
 private int cardNum;  
 final public static int *separation* = 30;  
 final public static int *unFlipCardSeparation* = 10;  
 public TablePile(int x, int y,int notFlipNum){  
 super(x, y);  
 this.notFlipNum = notFlipNum;  
 cardNum = notFlipNum+1;  
 this.setType(*TABLE\_PILE*);  
   
 }  
  
 public TablePile(int x,int y){  
 super(x,y);  
 cardNum=0;  
 this.notFlipNum=0;  
 this.setType(*TABLE\_PILE*);  
 }  
  
 @Override  
 public boolean includes(int tx, int ty) {  
 int beginX,beginY,endX,endY;  
 beginX = x;  
 beginY = y ;  
 endX = x + Card.*width*;  
 if(thePile.size() > 0)  
 endY = beginY + *unFlipCardSeparation* \* notFlipNum + *separation* \* (thePile.size() - 1 - notFlipNum) + Card.*height* ;  
 else  
 endY = beginY + Card.*height*;  
 boolean isInclude = beginX <= tx && tx <= endX && beginY <= ty && ty <= endY;  
 return isInclude;  
 }  
  
 @Override  
 public int select(int tx, int ty) {  
 if(!(isEmpty())){  
 int beginX,beginY,endX,endY;  
 //System.out.println(notFlipNum+" "+cardNum);  
 beginX = x ;  
 beginY = y + *unFlipCardSeparation* \* notFlipNum;  
 endX = x + Card.*width*;  
 endY = beginY + *unFlipCardSeparation* \* notFlipNum + *separation* \* (thePile.size() - 1 - notFlipNum) + Card.*height*;  
 boolean flip\_include = beginX <= tx && tx <= endX && beginY <= ty && ty <= endY;  
 //System.out.println(beginY+" "+endY);  
 if(flip\_include){  
   
 int c = (ty - beginY)/*separation* + notFlipNum;  
 if(c >= thePile.size()){  
 c = thePile.size() - 1;  
 }  
 return c;//  
 }  
 else  
 return -1;  
 }  
 else  
 return -1;  
 }  
  
 @Override  
 public void addCard(Object card) {  
  
 ArrayList<Card> cardList = (ArrayList<Card>)card;//*todo Exception in thread "AWT-EventQueue-0" java.lang.ClassCastException: solitaire.card.Card cannot be cast to java.util.ArrayList* cardNum += cardList.size();  
 for(int i = 0;i < cardList.size();i++){  
 thePile.push(cardList.get(i));  
 }  
 }  
  
 @Override  
 public void addSingleCard(Object card){  
 Card card1=(Card)card;  
 cardNum++;  
 thePile.push(card1);  
 }  
  
 @Override  
 public Card pop() {  
  
 cardNum--;  
 return super.pop();  
 }  
 @Override  
 public boolean isCanAdd(Card card){  
  
 if ( isEmpty())  
 return card.getNum() == 12;  
 Card topCard = top();  
 return (card.getColor() != topCard.getColor()) &&  
 (card.getNum() == topCard.getNum()-1 );  
 }  
  
 @Override  
 public void display(Graphics g) {  
 if (isEmpty()){  
 Image image= null;  
 try {   
// String picture = "picture/0.png";  
 String picture = "poker/0.png";  
 image = ImageIO.*read*(new File(picture));   
 } catch (IOException e) {  
 // *TODO Auto-generated catch block* e.printStackTrace();  
 System.*out*.println("class-CardPile-display-if(isEmpty())");  
 }   
 g.drawImage(image, this.x,this.y, Card.*width*, Card.*height*, null);  
 }  
 else{  
 int localy = y;  
 for (Enumeration e = thePile.elements(); e.hasMoreElements(); ) {  
   
 Card aCard = (Card) e.nextElement();  
 aCard.setX(x);  
 aCard.setY(localy);  
 aCard.draw(g);  
 if(aCard.isFaceup())  
 localy += *separation*;  
 else  
 localy += *unFlipCardSeparation*;  
   
 }  
 }  
 }  
 public int getNotFlipNum() {  
 return notFlipNum;  
 }  
 public void setNotFlipNum(int notFlipNum) {  
 this.notFlipNum = notFlipNum;  
 }  
 public int getCardNum() {  
 return cardNum;  
 }  
 public void setCardNum(int cardNum) {  
 this.cardNum = cardNum;  
 }  
 public static int getSeparation() {  
 return *separation*;  
 }  
 public static int getUnflipcardseparation() {  
 return *unFlipCardSeparation*;  
 }  
  
   
}

**\*Filename：UsedPile**

package solitaire.pile;  
  
  
public class UsedPile extends CardPile {  
  
 public int nowX;  
 public int nowY;  
 public boolean isup=false;//撤销时最后一张牌是否朝上  
  
  
 public boolean isIsup() {  
 return isup;  
 }  
  
 public void setIsup(boolean isup) {  
 this.isup = isup;  
 }  
  
  
 public CardPile lastPile;//上一个牌堆  
  
 public CardPile usedPile;//现在的牌堆  
  
 public void setUsedPile(CardPile usedPile,int nx,int ny) {  
 this.usedPile = usedPile;  
 nowX=nx;  
 nowY=ny;  
 }  
 public UsedPile(int x,int y){  
 super(x,y);  
 type=0;  
 }  
 public UsedPile(int x,int y,CardPile lastpile){ //x,y是lastpile的位置  
 super(x,y);  
 type=0;  
 lastPile=lastpile;  
 }  
  
 public void setLastPile(CardPile lastPile) {  
 this.lastPile = lastPile;  
 }  
}