ТЕХНИЧЕСКИ УНИВЕРСИТЕТ–СОФИЯ

ФАКУЛТЕТ "КОМПЮТЪРНИ СИСТЕМИ И УПРАВЛЕНИЕ"
Катедра „Програмиране и Компютърни Технологии"

Курсова Работа

по

Обектно-Ориентирано Програмиране

Игра с Карти

FreeCell

Изготвил Курсовата Работа:
Стефан Байчев
Email: sbaychev@gmail.com
ФКСУ - КТПП
Факултетен Номер 281611007
Магистър - изравнително обучение
I курс II семестър

Преподавател:……
/доц. д-р Иван Момчев/

София, 02.07.2012 г.

Съдържание

Въведение

По-надолу в този документ ще бъде описана програма тип „игра с карти“ (пасеанс), FreeCell, осъществена посредством обектно-ориентираният език JAVA (сегашната Версия е Java SE 7 Update 5) чрез IDE, Eclipse (Версия 1.4.2.20120213-0813).

Играта FreeCell има следният начин на работа и правила:

Обща постановка и разположение:

- Използва се стандартно 52-картово тесте с Карти
- Има четири отворени/ празни клетки и четири отворени/ празни основни клетки
- Отначало тестето с Карти бива разпределено на осем купчинки, четири от които имат седем карти и четири от които имат шест

Конструкцията при игра:

- Най-горната (изцяло видима) Карта на всяка купчинка може да започне нареждането
- Нареждането трябва да бъде построено от горе-надолу, чрез различни по цвят Карти

Извършвани действия:

- Коя да е карта от клетка (без основните) или такава горна от купчинка: може да бъде местена за да направи нареждане, да бъде поставена на празна клетка, празна основна клетка (отначало, важи само за карти тип Aco, но веднъж поставена там, неможе да бъде местена повече) или празна купчинка.
- Последователността на нареждане на местените карти, освен по цвят, е и по големина – като по-малката такава следва да бъде поставена над по-голямата и различна по цвят такава.
- Ако взетата карта неотговаря на някое от по-горе описаните условия за да бъде поставена, след нейното пускане тя се връща на позицията си

Край на Играта:

- Играта е спечелена, когато всички карти тип Aco са заели основните клетки и другите Карти от същият цвят и боя (Каро, Пики и тн.) са последователно разположени над тях – следвайки последователността Aco, Двойка….Десетка, Вале, Дама, Поп

Кратко Описание на Програмата

Програмата се визуализира посредством класът **UIFreeSell.java** - GUI интерфейсът е тук. Има последователно извикване и имплементиране на обекти от други класове, част от общият пакет на програмата (**package freecelll**), които осъществяват логическият модел на играта, разположението на Картовите елементи и отношението им едни към други спрямо подадените команди от потребителя.
Предстои да бъдат описани действията на всеки един от класовете използвани в **package freecelll** за създаване на играта.

Описание на използваните Класове

Класът Card.java

Целта му е да декларира, оформи и заложи базовите характеристики за елемента Карта, основна градивна единица за играта.

Public Member Functions

- Card (Face face, Suit suit)
- Face getFace ()
- Suit getSuit ()
- void setPosition (int x, int y)
- void draw (Graphics g)
- boolean isInside (int x, int y)

- int getX ()
- int getY ()
- void setX (int x)
- void setY (int y)
- String toString ()
- void turnFaceUp ()
- void turnFaceDown ()

Static Public Attributes

- static final int CARD_WIDTH
- static final int CARD_HEIGHT

В горната си част, прави import на използваните от нас библиотеки **import** java.awt.* и **import** javax.swing.*, следва обявяването на променливи нужни за дефиниране на една Карта (размер, самото изображение и др.), и за използване на рефлективност (нужно за коректното отразяване на пътища за достъп до ресурси и зареждане на този често използван Клас в JVM). Специфичното тук е декларирането им като **private static final**, като целта ни е те да са достъпни само от класовете част от пакета, за да се компилират по-бързо, за да са по „константни“ като стойности (целта е да ги предпазим от нежелана референция/ модификация) а и в някаква бъдеща употреба thread safe.
Следва, статичен метод, който извършва инициализацията на по-горе посочените статични променливи.
Имаме след това, декларация на няколко локални променливи (две от тях са **enum** референции), част от дефинирането на една Карта, нейното място на екрана, изображение и дали гледа с лицето си.

Constructor & Destructor Documentation

freecelll.Card.Card (Face face, Suit suit)

Дефиницията на метода се намира на ред **51**, от Card.java. Приема като параметри две enum променливи от тип **Face** и **Suit** (референции към по-долу описаните класове), където двете вече дефинираните локални за класът променливи приемат. Следва обработка на няколко променливи които извършват четенето и зареждането на изображението на Карта.

Member Function Documentation

Face getFace (), **Suit getSuit (), int getX ()**, **int getY ()**, **void setX (int x)**, **void setY (int y)**, **void turnFaceUp ()**, **void turnFaceDown ()** и **void setPosition (int x, int y)** методи говорят сами за себе си по имената и извършват точно това. За **X** и **Y** методите съответно **get** методите връщат дадената стойност, а **set** методите задават същите относно тях. Относно **Face** и **Suit** съдържащите методи, съответно тези с **get** връщат дадената стойност, а последните два с **Face** връщат съответната boolean стойност.

void freecelll.Card.draw (Graphics g)

Дефиницията на метода се намира на ред **99**, от Card.java. Извършва проверка дали Картата е с лице нагоре (правим под една форма проверка дали програмата се държи коректно):

- при положителен резултат се изрисува дадената Карта, с лицето си, на съответните вече зададени позиции x и y
- при отрицателен резултат се изрисува дадената Карта, с гърбът си, на съответните вече зададени позиции x и y

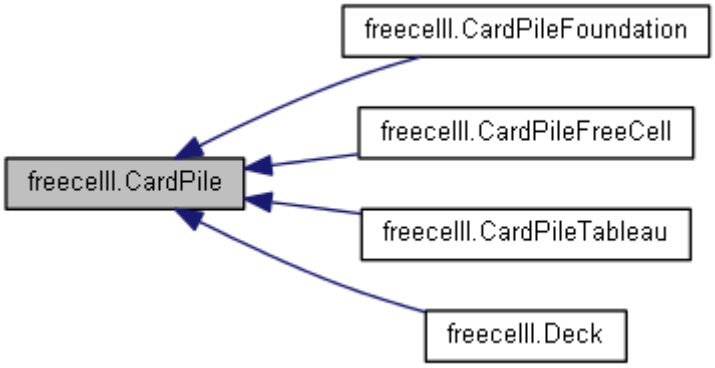
public boolean isInside (int x, int y)

Дефиницията на метода се намира на ред **112**, от Card.java. Приема като параметри две променливи от тип integer - координати на точка. Действието на метода е да извърши проверка дали тази точка, натисната/ избрана от мишката, е в Карта, и връща съответният за метода резултат.

public String toString ()

Дефиницията на метода се намира на ред **125**, от Card.java. Действието на метода е да пренапише за локално ползване **java.lang.Object.toString()** метода, чрез съответните нужни ни параметри.

Класът CardPile.java



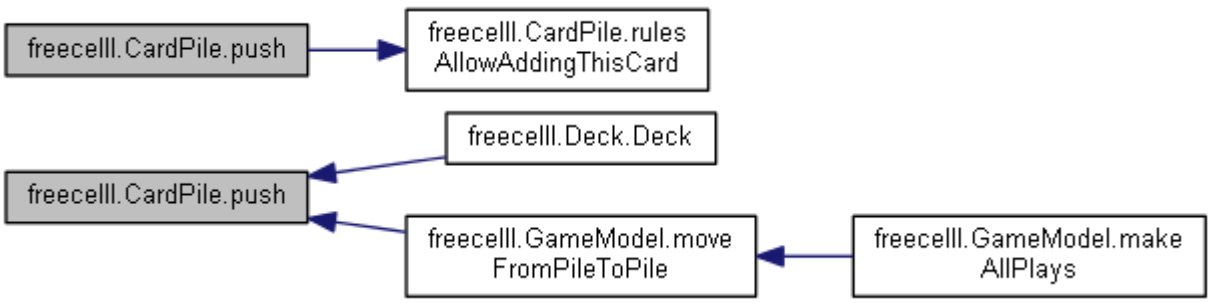
Диаграма на Унаследяването за CardPile

Public Member Functions

- Card pop ()
- void shuffle ()
- Card peekTop ()
- Iterator< Card > iterator ()
- ListIterator< Card > reverseIterator ()
- void clear ()
- void pushIgnoreRules (Card newCard)
- Card popIgnoreRules ()
- boolean push (Card newCard)
- boolean rulesAllowAddingThisCard (Card card)
- boolean isRemovable ()
- int size ()

Класът имплементира интерфейса **Iterable** върху референция от тип Card. Целта е да бъде използван „новият“ for цикъл (foreach) от другите класове ползващи този клас, а и метода асоцииран с **Iterable**. Идеята му е като цяла да дефинира специфични методи за обработката на Карта, нейното добавяне, махане, извикване/ взимане, а пък другите класове ползващи/ извикващи тези методи да си ги пренапишат според нуждите, но се запазва основата под формата на Карта част от цялото – част от една купчинка с Карти.
Първо имаме дефиниран ArrayList от тип **Card**, в който ще се съдържат всички Карти от едно тесте с **52** такива.
Следват дефинициите за два метода **public void pushIgnoreRules (Card newCard)** и **public Card popIgnoreRules()**, един след друг, от ред **13** до **25**, от CardPile.java. По имената може да се добие добра представа какво правят, засега няма да бъдат обяснявани, защото тяхното действие предстой да бъде обяснено по-долу.

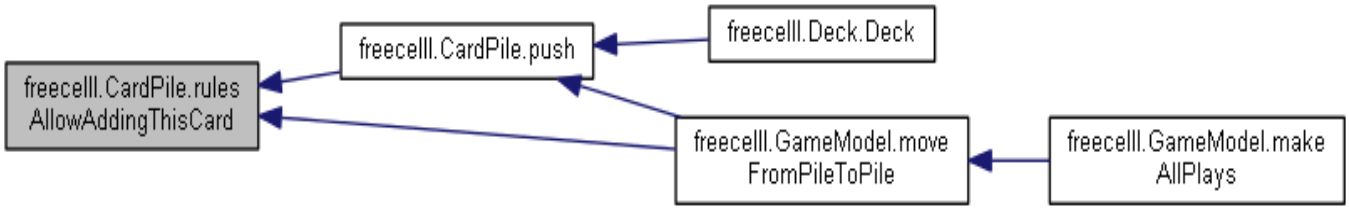
boolean freecelll.CardPile.push (Card newCard)



Дефиницията на метода се намира на ред **26**, от CardPile.java.
Приема като параметри променлива от тип Card. Действието на метода е първо да провери чрез друг метод с име **public boolean rulesAllowAddingThisCard(Card card)**, дали може да бъде добавена тази Карта към списъкът ArrayList:

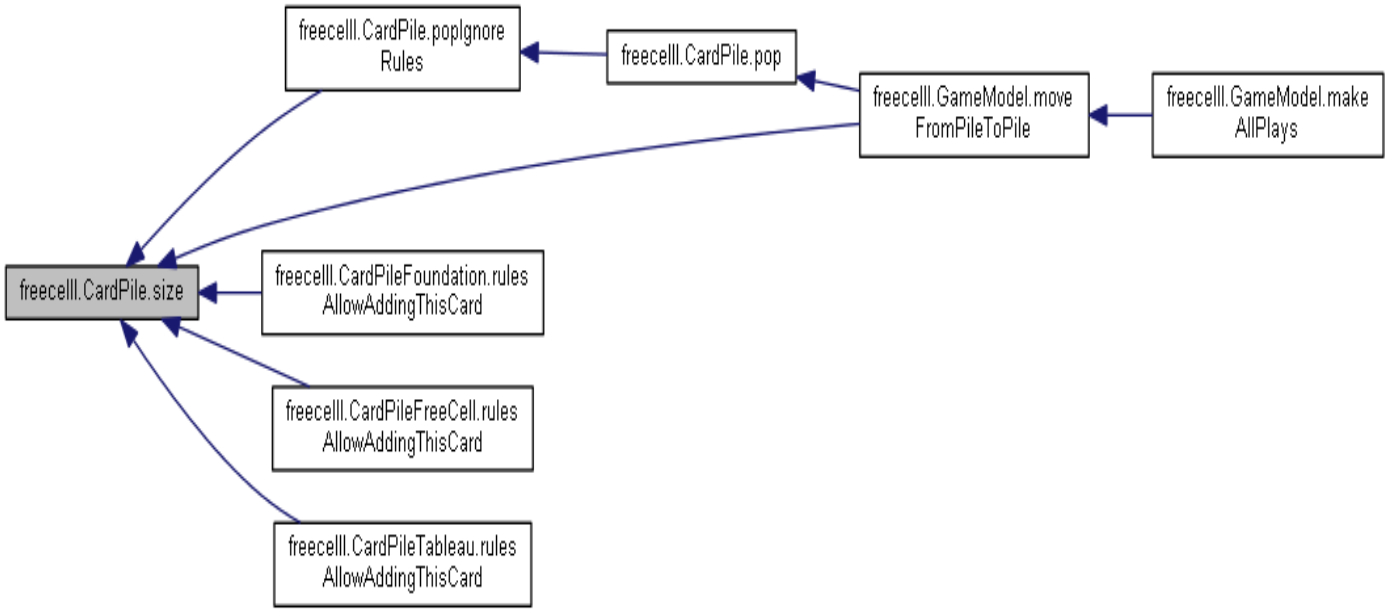
- при положителен резултат се добавя дадената Карта към списъкът ArrayList, и връща резултат **true**
- при отрицателен резултат се връща резултат **false**

public boolean rulesAllowAddingThisCard(Card card)



Дефиницията на метода се намира на ред **36**, от CardPile.java.
Приема като параметри променлива от тип **Card**. Метода връща резултат **true**. Той бива наново имплементиран в CardPileTableau.java, CardPileFoundation.java и CardPileFreeCell.java.

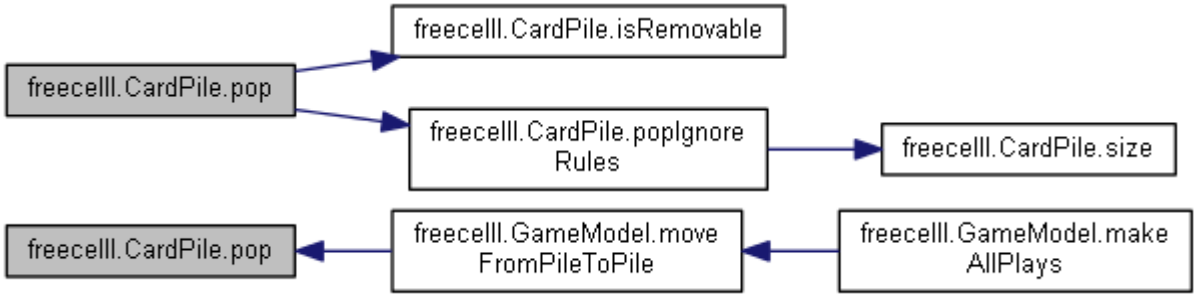
int freecelll.CardPile.size ( )



Графика на Извикване на Метода

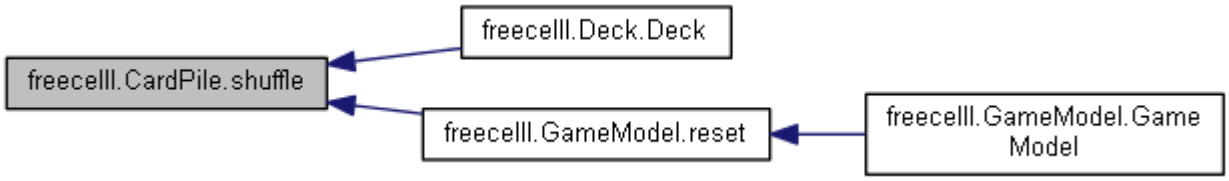
Дефиницията на метода се намира на ред **41**, от CardPile.java. Действието на метода е да върне като променлива от тип integer, големината на ArrayList-а от тип Card – иначе казано колко е броят на Картите.

Card freecelll.CardPile.pop ( )



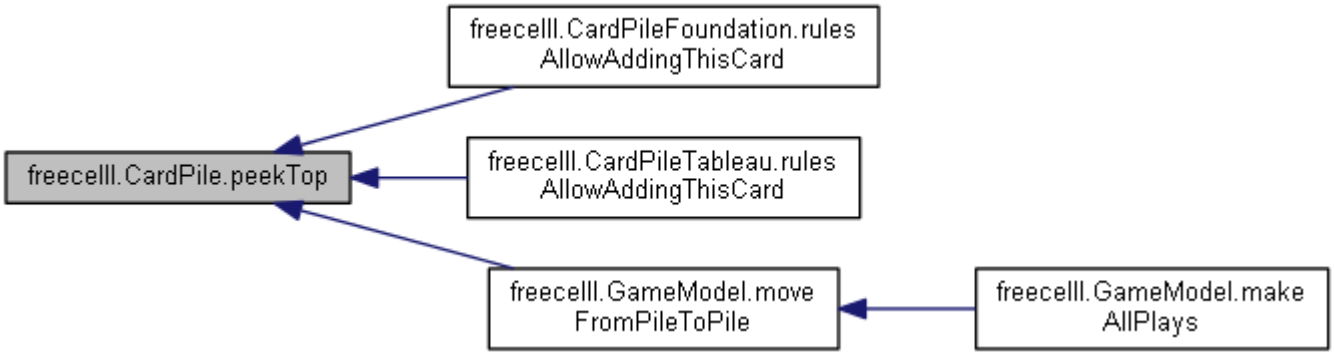
Дефиницията на метода се намира на ред **46**, от CardPile.java. Действието на метода е първо да провери дали резултата от ! **isRemovable()** метода и логическата операция, като при отрицателен резултат само от метода, ще изведе съобщение за грешка. След това, извиква метода **public Card popIgnoreRules()**.

void freecelll.CardPile.shuffle ( )



Дефиницията на метода се намира на ред 54, от CardPile.java. Действието на метода е да разбърка Картите подредени в списък от ArrayList, посредством **java.util.Collections.shuffle()** метода – той извършва пермутация чрез стандартен за JAVA източник на случайност.

Card freecelll.CardPile.peekTop ( )



Дефиницията на метода се намира на ред **59**, от CardPile.java. Действието на метода е да върне най-горната Карта от ArrayList списъкът с Карти.

Iterator<Card> freecelll.CardPile.iterator ( )

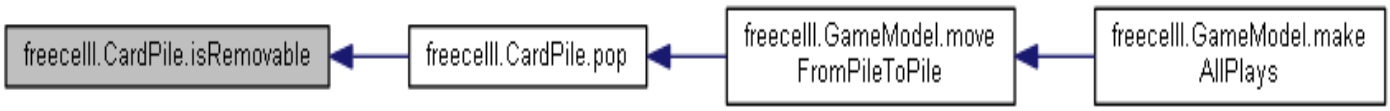
public ListIterator<Card> reverseIterator ( )

Дефиницията на методите се намира на ред **69** и **64**, от CardPile.java. Действието на двата метода е идентично, като по дефиниция те могат да обходят досега записаният списък от Карти, първият отпред – назад, а вторият отзад – напред.

void freecelll.CardPile.clear ( )

Дефиницията на метода се намира на ред **74**, от CardPile.java. Действието на метода е да изтрие ArrayList запаметеният списък с Карти посредством извикване на **void java.util.ArrayList.clear()** метода.

boolean freecelll.CardPile.isRemovable ( )



Дефиницията на метода се намира на ред **79**, от CardPile.java. Действието на метода е да върне резултат **true**.

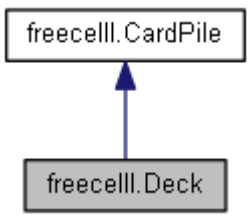
Класът Face.java

Този клас е от тип **enum** и целта му е да декларира **Face** типовете Карти използвани в играта, Асо, Двойка и тн.

Класът Suit.java

Този клас е от тип **enum** и целта му е да декларира **Suit** типовете Карти, а и техните Цветове използвани в играта: като за SPADES и CLUBS са съответно черно, а за HARTS и DIAMONDS са червено – поредността при декларирането в този и предният клас запазва последователността на дефинираните в Card.java стойности за зареждане/ дефиниране на Картите.
В класът има два метода, единият е от тип Конструктор който просто приема предаваната, под формата на променлива, стойност за цвят, и я съхранява в локално дефинираната такава. Последно споменатата локална стойност бива върната като резултат от вторият метод.

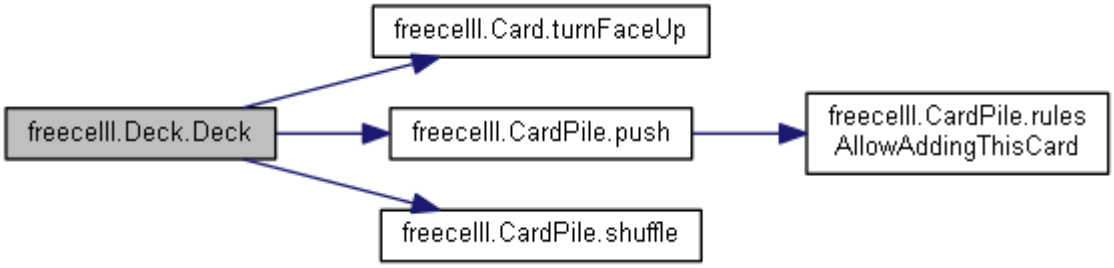
Класът Deck.java



Идеята на този клас е да унаследи Класът CardPile.java и посредством вече описаните по-горе Методи и променливи в класовете CardPile.java, Card.java, Suit.java и Face.java, да създаде тесте с карти.

Constructor & Destructor Documentation

freecelll.Deck.Deck ( )



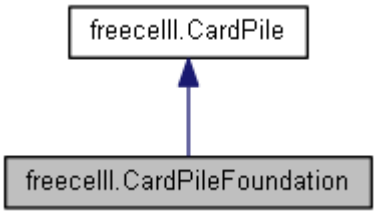
Дефиницията на метода се намира на ред **9**, от Deck.java. Действието на метода е чрез начално описаните да създаде чрез for-each цикъл, една по една Картите за едно тесте, да ги добави в списък от тип ArrayList и да ги разбърка накрая.

Следващите Класове, **CardPileFoundation.java**, **CardPileFreeCell.java** и **CardPileTableau.java**, които предстоят да бъдат описани дефинират правила и положения за, както следва от редът им на записване:

- **Празните Основни Клетки** 4 на брой
- **Купчинките с Карти** 8 на брой
- **Празните Клетки** 4 на брой

Класът CardPileFoundation.java

Идеята на този клас е да унаследи Класът CardPile.java и посредством вече описаните по-горе Методи и променливи в класовете CardPile.java, Card.java, Suit.java и Face.java, да определи правилата за действие в четирите основни клетки – там където се поставят първо и само карти тип Ace, а в последствие Двойки и тн. до Поп от еднакъв вид (SPADES и тн.).



Public Member Functions

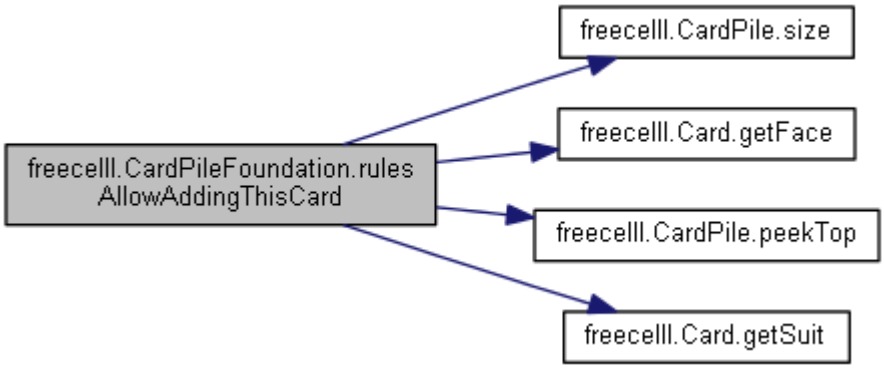
- boolean rulesAllowAddingThisCard (Card card)
- boolean isRemovable ()

Member Function Documentation

boolean freecelll.CardPileFoundation.isRemovable ( )

В този клас се пренаписват следните два Метода:

boolean freecelll.CardPileFoundation.rulesAllowAddingThisCard (Card card)



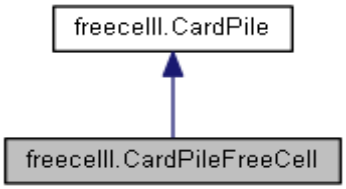
Дефиницията на метода се намира на ред **11**, от CardPileFoundation.java. Приема като параметър променлива от тип Card. Действието на метода е да направи две проверки една след друга, като:

- при първата проверка, при положителен резултат от нея, тоест ако няма Карти в дадената основна клетка и Картата която сме взели за да сложим е Ace – връща резултат **true** за изпълнението на метода и приключва дейността си

- при втората проверка, при положителен резултат от нея, тоест ако има вече карта в дадената основна клетка, следва да се направи още една проверка дали исканата да бъде сложена Карта е от същият вид (на вече сложената, SPADES и тн.) и е с поредност **Едно** по-голяма от дадената, ако и тук резултат е положителен връща резултат **true** за изпълнението на метода и приключва дейността си
- ако нито една от двете проверки не бъде реализирана, метода ще върне резултат **false**

boolean freecelll.CardPileFoundation.isRemovable ( )

Дефиницията на метода се намира на ред **31**, от CardPileFoundation.java. Действието на метода е да върне резултат **false** – карта поставена която и да е от основните клетки неможе да бъде местене повече.



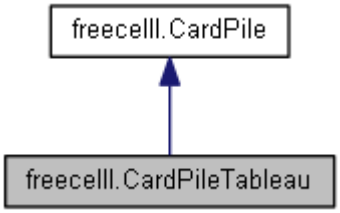
Класът CardPileFreeCell.java

Идеята на този клас е да унаследи Класът CardPile.java. Класът пренаписва по-долу даденият метод, като целта му е да определи правилото за поставяне на Карта в клетка:

boolean freecelll.CardPileFreeCell.rulesAllowAddingThisCard (Card card)

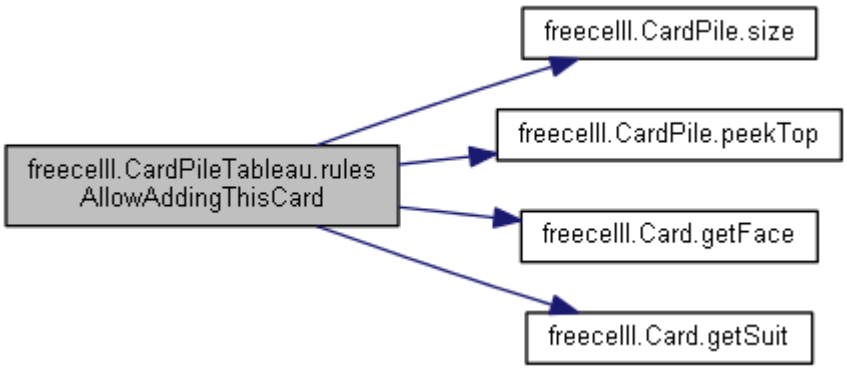
Дефиницията на метода се намира на ред **11**, от CardPileFreeCell.java. Действието на метода е да върне резултат **true** или **false** в зависимост от резултата на извикваният метод **size() == 0** и равнение с нула – дали има вече поставена Карта.

Класът CardPileTableau.java



Идеята на този клас е да унаследи Класът CardPile.java. Класът пренаписва по-долу даденият метод, като целта му е да определи правилото за поставяне на Карта в която и да е от купчинките с Карти:

boolean freecelll.CardPileTableau.rulesAllowAddingThisCard (Card card)



Дефиницията на метода се намира на ред **13**, от CardPileTableau.java. Приема като параметър променлива от тип Card. Действието на метода е да направи една проверка с няколко зададени условия за нея. Първото е, дали купчинката с Карти е празна, следва дали исканата да бъде поставена Карта е с **Едно** по-малка поредност от тази върху която се иска да бъде поставена, и дали цветът е различен от дадената. При положителен резултат от тази проверка, метода връща резултат **true** (Картата може да се постави), при отрицателен връща резултат **false** (Картата неможе да се постави)**.**

Класът GameModel.java

Целта на този клас е да постави логическите рамки на това как работи Free Cell програмата. Класът имплементира интерфейса **Iterable** върху референция от тип CardPile. Целта е да бъде използван „новият“ for цикъл (foreach), а и метода асоцииран с **Iterable**.
Първото нещо което извършва класът е дефинирането на променливи от тип CardPile, чрез масив, за да имаме разпределени трите полета за действие на Карти в играта (както е описано във **Въведение**). Следва дефинирането на две променливи от тип ArrayList, едната е от тип

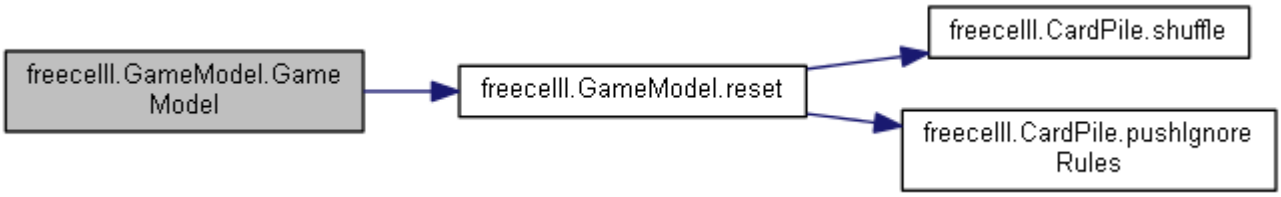
CardPile събирателна за всички купчинки с карти (включително и двете начално празни клетки), другата е от тип ChangeListener, събирателна за слушател (ще бъде обяснено по-надолу). Последната дефинирана променлива е от тип ArrayDeque, реферираща CardPile, като нейната нужда е за осъществяване на стековият принцип на купчините с Карти – тяхното добавяне една над друга преместване (изтриване от дадената) и тн.

Public Member Functions

- GameModel ()
- void reset ()
- Iterator< CardPile > iterator ()
- CardPile getTableauPile (int i)
- CardPile[] getTableauPiles ()
- CardPile[] getFreeCellPiles ()
- CardPile getFreeCellPile (int cellNum)
- CardPile[] getFoundationPiles ()
- CardPile getFoundationPile (int cellNum)
- boolean moveFromPileToPile (CardPile source, CardPile target)
- void makeAllPlays ()
- void addChangeListener (ChangeListener someoneWhoWantsToKnow)

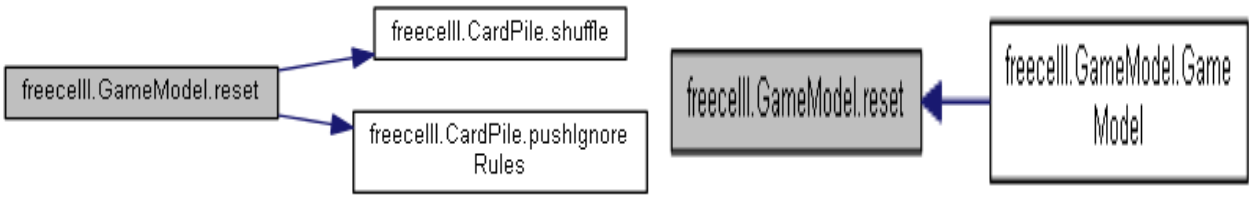
Constructor & Destructor Documentation

freecelll.GameModel.GameModel ( )



Дефиницията на метода се намира на ред **27**, от GameModel.java. Действието на метода е разделено на няколко етапа. Първо се обявява като такава вече заявената променлива от тип ArrayList, за всички купчинки с Карти, следва обявяването като такива на вече заявените променливи за трите региона на действие на Карти – по 4 позиции за основните клетки и другите клетки от тип CardPile и 8 позиции за Картите от купчинките от тип CardPileTableau. Следва чрез три последователни for-цикъла заявяване и запазването на празни места за тези обявени три региона, като всички те биват запазени в упоменатият ArrayList за всички Карти. Накрая създава вече декларираната променлива ArrayList, реферерирана към ChangeListener, след това се извиква метода **reset()**.

void freecelll.GameModel.reset ( )

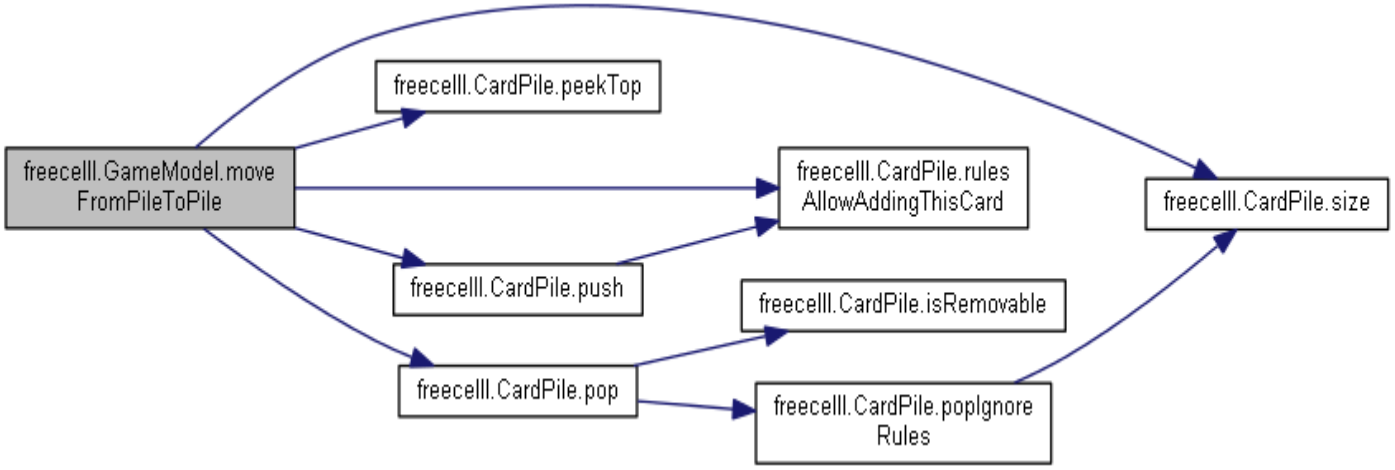


Дефиницията на метода се намира на ред **59**, от GameModel.java. Действието на метода е да създаде променлива от тип Deck и да използва, извика вече дефинираният метод shuffle() – създава тестето с карти и го разбърква. Чрез for-each цикъл за всички купчинки изчиства списъкът с запаметени там Карти (ако има някакъв пропуск при изпълнение на кода или просто защото е започната нова игра – не се изтрива това първо споменато, сега създадено тесте с карти). Следва добавяне на Картите от тестето по осемте по-горе създадени купчинки, следвайки посочените във **Въведение** правила. Накрая има извикване към метода **_notifyEveryoneOfChanges()** – който следва да бъде обяснен по-долу.

Методите, както следва, говорят сами за себе си относно дейността си по имената, по-подробно описание на тяхната дейност ще бъде дадено при употребата им:

Iterator <CardPile> iterator (), CardPile getTableauPile (int i), CardPile[] getTableauPiles (), CardPile[] getFreeCellPiles (), CardPile getFreeCellPile (int cellNum), CardPile[] getFoundationPiles (), CardPile getFoundationPile (int cellNum).

boolean freecelll.GameModel.moveFromPileToPile (CardPile source, CardPile target)



Дефиницията на метода се намира на ред **118**, от GameModel.java. Приема като параметри две променливи от тип CardPile. Действието на метода е да осъществи местенето на Карти от една купчинка към друга. Първо имаме декларирана променлива от тип boolean, със стойност равна на **false** (идеята е, ако по-долу описаното условие не е изпълнено да се върне към извикващият, че неможе да се осъществи). Следва проверка дали в дадената купчинка от която искаме да изместим Карта, има Карта:

- при положителен резултат, следва взимане на Картата и още една проверка дали същата може да бъде преместена върху друга такава (съответно друга купчинка) според дефинираните вече правила (**rulesAllowAddingThisCard (Card card)**).
  - ✓ при положителен резултат от последната проверка чрез вече дефинираните методи **push** и **pop** (в класът CardPile.java, първият добавя, а вторият изважда Карта) извършваме съответните действия над двете купчинки, извикваме метода **_notifyEveryoneOfChanges()**, отразяваме промените в начално създадената променлива от тип ArrayDeque и връщаме резултат **true**
- при отрицателен резултат връщаме стойността на по-горе дефинираната за метода променлива от тип Boolean - **false**

void addChangeListener (ChangeListener someoneWhoWantsToKnow)

Дефиницията на метода се намира на ред **174**, от GameModel.java. Приема като параметър променлива от тип ChangeListener. Действието на метода е да записва във създадената променлива от тип ArrayList (реферирана към ChangeListener) тези стойности.

private void _notifyEveryoneOfChanges()

Дефиницията на метода се намира на ред **179**, от GameModel.java. Действието на метода е да посредством for-each цикъл да добави към горе описаният ArrayList (реферирaн към ChangeListener) събитието дефинирано чрез
void javax.swing.event.ChangeListener.stateChanged(ChangeEvent e).

Класът UICardPanel.java

Дейността на този Клас е да окомплектова досега създадените от другите класове (част от пакета **freecelll**) обекти и специфични методи, за да изобрази картите и да поддържа действието на мишката. За тази цел той унаследява методите и атрибутите на JComponent класът, и имплементира различните слушатели за очакваните от потребителя действия - MouseListener, MouseMotionListener и ChangeListener.
Първо, UICardPanel.java, създава набор от променливи от тип **private static final**, чиято цел е да дефинират размерите за трите региона, един спрямо друг и разположението си в

компонента на програмата. Определяме също големината/ ширината на Картите за виждане на екрана, а и цветът на фона (зелен).
Задаваме после променливи за да определим при избиране с мишка откъде точно дадена Карта сме изтеглили, а също и две такива от тип Card и CardPile (за да можем да определим коя карта, къде се намира).
Биват създадени също: променлива от тип IdentityHashMap (за да знаем точното разположение на купчинките), и такава от тип GameModel.

UICardPanel(GameModel model)

Дефиницията на метода се намира на ред **82**, от UICardPanel.java. Приема като параметур променлива от тип GameModel. Действието на метода е да зададе, разчертае виртуално и запази в декларираната променлива от тип IdentityHashMap тези стойности.
Първо биват зададени границите на компонента в който ще се показват всички Карти и се инициализира цветът, а след това се добавят два слушателя за дейностите с мишката.
От тук нататък следва посредством първо дефинираните в началото на класът променливи (за размерите на Карта, специфична позиция за един от трите региона с Карти и съответната променлива за да биват разполагани Картите по **X** и **Y** координатите), посредством един for-цикъл (с дължина на изпълнение 8) и изменение на стойността на координатата **X** след всеки цикъл - всеки един от трите региона бива виртуално изрисуван и запаметен в IdentityHashMap променливата.
Накрая добавяме слушател от тип ChangeListener (вече дефиниранийт метод **addChangeListener** от класът GameModel) за тази инстанция на GameModel.

public void paintComponent(Graphics g)

Дефиницията на метода се намира на ред **126**, от UICardPanel.java. Приема като параметър променлива от тип Graphics. Действието на метода е да пренапише съответният стандартен за JAVA метод. Първо се задават няколко параметъра които запълват екрана със зелен цвят (за да се избегне виждането на „визуални артефакти“ при местене и тн.), като преди това сме взели и локално предали нужните променливи, а след тези, залагаме за употреба черен цвят.
Метода извиква рекурсивно чрез три for-цикъла, където се извикват запазените за трите региона **get** дефинирани метода за извличане на вече съхранени в тях обекти (Карти), метода **private void _drawPile(Graphics g, CardPile pile, boolean topOnly)** – той следва да бъде обяснен.
Накрая на метода се проверява променливата от тип Card не е **null** (има действие да бъде издърпана), ако е така се извиква метода **draw** от класът Card.

private void _drawPile(Graphics g, CardPile pile, boolean topOnly)

Дефиницията на метода се намира на ред **156**, от UICardPanel.java. Приема като параметри променливи от тип Graphics, CardPile и boolean. Действието на метода е да изрисува първо правоъгълниците в които се намират картите и след това прави проверка дали подадената купчинка има Карти за изрисуване, ако няма просто приключва дейнсота си, но ако има:

- Прави проверка дали става дума за Карта намираща се в един от горните два региона, ако да:
  - ✓ прави проверка дали изтеглената картата е останала **null** като стойност за влачене (не е пипана), ако е така я нарисува на съответното място с предадените от IdentityHashMap стойности и заредени от класът **Card** данни.

  ако не:
  - ✓ изрисува посредством горе посочените и for-цикъл, всички Карти (тези от дадена купчинка), но само ако изтеглената картата е останала **null** като стойност за влачене (не е пипана). Променя стойността на координата **Y** след всяко изпълнение (за да можем да се движим нагоре-надолу)

public void mousePressed(MouseEvent e)

Дефиницията на метода се намира на ред **188**, от UICardPanel.java. Приема като параметър променлива от тип MouseEvent. Действието на метода е да проверява дали се намираме с

мишката върху някоя Карта. Метода първо взима в локални променливи сегашните координати на мишката и занулява променливата която показва дали сме в някоя Карта. Правим посредством for-each цикъл проверка дали някой от купчинките с Карти (в трите региона) от сегашната инстанция на GameModel може да има местене на карти, а и дали не е празна. Чрез вече дефинираните методи **peekTop()** и **isInside(x, y)**, „поглеждаме“ дадената горна Карта от съответната купчинка, спрямо това къде се намира, и ако вторият метод върне положителен резултат записваме нужните във вече декларираните за класът UICardPanel.java променливи и прекъсваме дейността на метода – първият и вторият метод продължават докато не се обходят възможните купчинки с Карти.

public void stateChanged(ChangeEvent e)

Дефиницията на метода се намира на ред **216**, от UICardPanel.java. Приема като параметър променлива от тип ChangeEvent. Действието на метода е да извика метода _clearDrag() и след това метода repaint().

public void mouseDragged(MouseEvent e)

Дефиницията на метода се намира на ред **228**, от UICardPanel.java. Приема като параметър променлива от тип MouseEvent. Действието на метода е да засече позицията на мишката в момента и да прерисува даденото място. Метода, първо прави проверка дали въобще има нужда от прерисуване, дали мишката се намира в момента върху Карта – дали има влачене. Следва, ако проверката е отрицателна, дефиниране на локални променливи за моментната позицията на **X** и **Y** и тяхното ограничаване в рамките на компонента (ограничаване позицията на Картата). Накрая зареждаме чрез вече дефинираният метод **setPosition(int x, int y)**, последното местоположение на Картата, и след това прерисуваме компонента.

public void mouseReleased(MouseEvent e)

Дефиницията на метода се намира на ред **253**, от UICardPanel.java. Приема като параметър променлива от тип MouseEvent. Действието на метода е да провери дали нещо бива местено в момента, след това като чрез if-statement виждаме дали променлива от тип CardPile е различна от **null**:

- Ако да, взима координатите на сегашното местоположение на мишката и ги предава на метода **_findPileAt(int x, int y)** (който ще бъде описан по-долу), ако върнатият резултат е различен от **null**, значи Картата се намира над друга Карта и се извиква вече дефинираният метод **boolean moveFromPileToPile (CardPile source, CardPile target)** за да извърши евентуалното местене.

Накрая метода извиква метода **_clearDrag()** и **repaint()** на реферираната инстанция на компонента.

private void _clearDrag()

Дефиницията на метода се намира на ред **278**, от UICardPanel.java. Действието на метода е да занули променливите пазещи стойност за това дали е влачена Карта и от кое място.

private CardPile _findPileAt(int x, int y)

Дефиницията на метода се намира на ред 284, от UICardPanel.java.
Приема като параметри две променливи от тип integer. Действието на метода е да провери преминавайки през стойностите запазени в IdentityHashMap променливата, дали предаваните координати **X** и **Y** отговарят на местоположението на някоя от всички купчинки с карти (и двете клетки) – при положителен резултат връща коя е купчинката, ако ненамира такава връща **null**.

Класът UIFreeSell.java

Идеята на този клас е да унаследи JAVA дефинираният клас JFrame, за да се покажат досега дефинираните обекти в прозорец, тук се реализират дейностите на всички досега създадени класове от пакета **freecelll** нужни за работата на програмата Free Cell.
Класът първо обявява две променливи от тип UICardPanel и GameModel, последната я и декларира.

Constructor & Destructor Documentation

freecelll.UIFreeSell.UIFreeSell ( )

Дефиницията на метода се намира на ред **33**, от UIFreeSell.java.
Действието на Конструктора – метод, е да определи и зададе начините на разположение на нужните за Визуалната работа, обекти и компоненти. Посредством обявената, но недефинирана променлива от тип UICardPanel, осъществяваме чрез нея модела на работа на играта – чрез дефинираната GameModel променлива. Следва създаването и добавянето на бутон за „Нова Игра“, като му добавяме и слушател да следи дали е натиснат. Бива създаден компонент от JPanel тип, в който добавяме вече споменатия бутон. След това се създава друг компонент от JPanel тип, но с различно разположение за обектите в него. Добавяме в него първо-създаденият компонент (поставяйки го най-горе в неговата рамка), а също и променливата от тип UICardPanel **(**JComponent, която да се намира в центъра на неговата рамка). Следва задаването на характеристиките на прозореца в който са добавени досега изказаните, като задаваме за ContentPane да е вторият създаден JPanel, задаваме заглавие „Free Cell”, задаваме при изход (с „ **X** ” бутона на прозореца) програмата да приключи, и задаваме да бъде „пакетирано“ досега казаното, като се покаже в центъра на екрана, без възможност да бъде променяна големината на прозореца.

static void freecelll.UIFreeSell.main (String[] args)

Дефиницията на метода се намира на ред **21**, от UIFreeSell.java. Действието на този главен метод е да извика чрез JAVA методът **void javax.swing.SwingUtilities.invokeLater(Runnable doRun)**, нова инстанция на горе-описаният Конструктор.

class ActionNewGame implements ActionListener

Дефиницията на този вътрешен клас се намира на ред **66**, от UIFreeSell.java. Действието му е да имплементира ActionListener при натискане на бутона „New Game”. Целта е чрез неговият метод **public void actionPerformed(ActionEvent evt)**, който извиква метода (вече дефиниран в класът GameModel.java) **reset()** върху сегашната инстанция на променливата от тип GameModel, да рестартира играта/ програмата, да прави нова игра.

Source Code на програмата

Card.java

```
1  // Description: Represents a single Card
2  // Issues:
3  // * Fragile: This loads each card image from a file, and has
4  // the file-naming conventions for the card built into it
5  // To change to another set of card images, it's necessary to change the code
6  // * It gets the images
7  // * It keeps track of it's x,y coordinates
8  // * It can draw itself
9  package freecelll;
10
11 import java.awt.*;
12 import javax.swing.*;
13
15 public class Card
16 {
17  //============================================================ constants
18  public static final int CARD_WIDTH; // Initialized in static initializer.
19  public static final int CARD_HEIGHT; // Initialized in static initializer..
20  //doing the static final to make it more constant...faster compiling, "thread
    safe" in a way
```

```
21  private static final String IMAGE_PATH = "/cardimages/";
22  private static final ImageIcon BACK_IMAGE; // Image of back of a card
23
24  private static final Class <?> CLASS = Card.class; //using reflection to get
    the
25  private static final String PACKAGE_NAME; //Class object for Card
26  private static final ClassLoader CLSLDR; //loads the java class file into the
    JVM
27  //========================================================= static initializer
28  static // just a static method that does some of the initialization, the static
    ones
29  {
30  //... Get current classloader, and get the image resources
31  // This is broken down into small steps for debugging
32  PACKAGE_NAME = CLASS.getPackage().getName(); //identifying the class..
33  CLSLDR = CLASS.getClassLoader(); //returns the class loader for the class
34  String urlPath = PACKAGE_NAME + IMAGE_PATH + "b.gif";
35  //using the .net.URL as a resource locator via the classloader
36  java.net.URL imageURL = CLSLDR.getResource(urlPath);
37  BACK_IMAGE = new ImageIcon(imageURL);
38
39  //... These constants are assumed to work for all cards.
40  CARD_WIDTH = BACK_IMAGE.getIconWidth();
41  CARD_HEIGHT = BACK_IMAGE.getIconHeight();
42  }
43  //========================================================= instance variables
44  private Face _face;
45  private Suit _suit;
46  private ImageIcon _faceImage;
47  private int _x;
48  private int _y;
49  private boolean _faceUp = true;
50  //================================================================ constructor
51  public Card(Face face, Suit suit) {
52  //... Set the face and suit values.
53  _face = face;
54  _suit = suit;
55
56  //... Assume card is at 0,0
57  _x = 0;
58  _y = 0;
59
60  //... By default the cards are face up.
61  _faceUp = false;
62
63  //... Read in the image associated with the card.
64  // Each card is stored in a GIF file where the name has two chars,
65  // ex, 3h.gif for the three of hearts.
66
67  //... Create the file name from the face and suit. order through ordinal
68  char faceChar = "a23456789tjqk".charAt(_face.ordinal());
69  char suitChar = "shcd".charAt(_suit.ordinal());
70  String cardFilename = "" + faceChar + suitChar + ".gif";
71
72  // the class loader is be used... returns a URL of the card file.
73  //... Get current classloader, and get the image resources.
74  String path = PACKAGE_NAME + IMAGE_PATH + cardFilename;
75  java.net.URL imageURL = CLSLDR.getResource(path);
76
77  //... Load the image from the URL.
78  _faceImage = new ImageIcon(imageURL);
79  }
80  //================================================================== getFace
81  // Returns face value of card.
82  public Face getFace()
83  {
84  return _face;
```

```
 85  }
 86  //=========================================================================== getSuit
 87  public Suit getSuit()
 88  {
 89  return _suit;
 90  }
 91  //=========================================================================== setPosition
 92  public void setPosition(int x, int y)
 93  {
 94  _x = x;
 95  _y = y;
 96  }
 97  //=========================================================================== draw
 98  // Draws either face or back.
 99  public void draw(Graphics g)
100  {
101  if (_faceUp)
102  {
103  _faceImage.paintIcon(null, g, _x, _y);
104  } else
105  {
106  BACK_IMAGE.paintIcon(null, g, _x, _y);
107  }
108  }
109  //=========================================================================== isInside
110  // Given a point, it tells whether this is inside card image.
111  // Used to determine if mouse was pressed in card.
112  public boolean isInside(int x, int y)
113  {
114  return (x >= _x && x < _x + CARD_WIDTH) && (y >= _y && y < _y + CARD_HEIGHT);
115  }
116  //=========================================================================== getX, getY
117  public int getX() {return _x;}
118  public int getY() {return _y;}
119
120  //=========================================================================== getX, getY
121  public void setX(int x) {_x = x;}
122  public void setY(int y) {_y = y;}
123
124  //=========================================================================== toString
125  @Override public String toString()
126  {
127  return "" + _face + " of " + _suit;
128  }
129  //=========================================================================== turnFaceUp
130  public void turnFaceUp() {_faceUp = true;}
131
132  //=========================================================================== turnFaceDown
133  public void turnFaceDown() {_faceUp = false;}
134 }
```

Deck.java

```
  1 // Description: A Deck is a particular kind of CardPile with 52 Cards in it.
  2
  3 package freecelll;
  4
  5 public class Deck extends CardPile
  6 {
  7  //=========================================================== constructor
  9  public Deck()
 10  {
 11  for (Suit s : Suit.values())
 12  {
 13  for (Face f : Face.values())
 14  {
 15  Card c = new Card(f, s);
```

```
 c.turnFaceUp();
 this.push(c);
 }
 }
 shuffle();
 }
}
```

Face.java

```
// Description: Defines a card face value.

package freecelll;

enum Face
{
 ACE, DEUCE, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
 NINE, TEN, JACK, QUEEN, KING;
}
```

Suit.java

```
// Description: A card suit type.

package freecelll;

import java.awt.Color;

enum Suit
{
 //=========================================================== constants
 SPADES(Color.BLACK), HEARTS(Color.RED), CLUBS(Color.BLACK),
 DIAMONDS(Color.RED);
 //============================================================= field
 private final Color _color;
 //=========================================================== constructor
 Suit(Color color)
 {
 _color = color;
 }
 //============================================================== getColor
 public Color getColor()
 {
 return _color;
 }
}
```

CardPile.java

```
// Description: A pile of cards (can be used for a hand, deck, discard pile...)
// Subclasses: Deck (a CardPile of 52 Cards)
package freecelll;

import java.util.*;

public class CardPile implements Iterable<Card>
{
 //=========================================================== instance variables
 private ArrayList<Card> _cards = new ArrayList<Card>(); // All the cards.

 //=========================================================== pushIgnoreRules
 public void pushIgnoreRules(Card newCard)
 {
 _cards.add(newCard);
 }
 //=========================================================== popIgnoreRules
 public Card popIgnoreRules()
 {
```

```
20  int lastIndex = size()-1;
21  Card crd = _cards.get(lastIndex);
22  _cards.remove(lastIndex);
23  return crd;
24  }
25  //================================================================== push
26  public boolean push(Card newCard) {
27  if (rulesAllowAddingThisCard(newCard)) {
28  _cards.add(newCard);
29  return true;
30  } else {
31  return false;
32  }
33  }
34  //=============================================== rulesAllowAddingThisCard
35  //... Subclasses can override this to enforce their rules for adding.
36  public boolean rulesAllowAddingThisCard(Card card)
37  {
38  return true;
39  }
40  //================================================================== size
41  public int size()
42  {
43  return _cards.size();
44  }
45  //================================================================== pop
46  public Card pop()
47  {
48  if (!isRemovable()) {
49  throw new UnsupportedOperationException("Illegal attempt to remove.");
50  }
51  return popIgnoreRules();
52  }
53  //================================================================ shuffles
    the cards
54  public void shuffle()
55  {
56  Collections.shuffle(_cards);
57  }
58  //================================================================ peekTop
59  public Card peekTop() //gets the top card of the pile
60  {
61  return _cards.get(_cards.size() - 1);
62  }
63  //================================================================ iterator
64  public Iterator<Card> iterator()
65  {
66  return _cards.iterator();
67  }
68  //============================================================ reverseIterator
69  public ListIterator<Card> reverseIterator()
70  {
71  return _cards.listIterator(_cards.size());
72  }
73  //================================================================== clear
74  public void clear()
75  {
76  _cards.clear();
77  }
78  //============================================================ isRemovable
79  public boolean isRemovable()
80  {
81  return true;
82  }
83 }
```

CardPileFoundation.java

```
1  // Purpose: Represents a Foundation card pile.
2
3  package freecelll;
4
6  public class CardPileFoundation extends CardPile
7  {
8   //================================================ rulesAllowAddingThisCard
9   //... Accept card if pile is empty, or
10  // if face value is one lower and it's the opposite color.
11  @Override public boolean rulesAllowAddingThisCard(Card card)
12  {
13  //... Accept any ace on an empty pile.
14  if ((this.size() == 0) && (card.getFace() == Face.ACE))
15  {
16  return true;
17  }
18
19  if (size() > 0)
20  {
21  Card top = peekTop();
22  if ((top.getSuit() == card.getSuit() &&
23  (top.getFace().ordinal() + 1 == card.getFace().ordinal())))
24  {
25  return true;
26  }
27  }
28  return false;
29  }
30  //================================================================ isRemovable
31  @Override public boolean isRemovable()
32  {
33  return false;
34  }
35 }
```

CardPileFreeCell.java

```
1  // Purpose: CardPile specialized to add only one card.
2
3  package freecelll;
4
6  public class CardPileFreeCell extends CardPile
7  {
8
9   //================================================ rulesAllowAddingThisCard
10  //... Accepts a card only if a pile is empty.
11  @Override public boolean rulesAllowAddingThisCard(Card card)
12  {
13  //... Accept only if the current pile is empty.
14  return size() == 0;
15  }
16 }
```

CardPileTableau.java

```
1  // Purpose: Card pile with the initial cards.
2  // Only need to specify rules for adding cards.
3  // Default rules apply to remove them.
4
5  package freecelll;
6
8  public class CardPileTableau extends CardPile {
9
10  //================================================================== push
11  //... Accepts a card, if pile is empty, or
```

```
12  // if face value is one lower and it's the opposite color.
13  @Override public boolean rulesAllowAddingThisCard(Card card)
14  {
15  if ((this.size() == 0) ||
16  (this.peekTop().getFace().ordinal() - 1 == card.getFace().ordinal() &&
17  this.peekTop().getSuit().getColor() != card.getSuit().getColor())) {
18  return true;
19  }
20  return false;
21  }
22 }
```

GameModel.java

```
1  // Purpose: how things work
2
3  package freecelll;
4
5  import java.util.*;
6  import javax.swing.event.ChangeEvent;
7  import javax.swing.event.ChangeListener;
8
10 public class GameModel implements Iterable<CardPile>
11 {
12  //========================================================================= fields
13  private CardPile[] _freeCells;
14  private CardPile[] _tableau;
15  private CardPile[] _foundation;
16
17  private ArrayList<CardPile> _allPiles;
18
19  private ArrayList<ChangeListener> _changeListeners;
20
21  //... Using the Java Deque to implement a stack.
22  // Push the source and destination piles on, every time a move is made.
23  // Pop them off to do the undo. ...must suppress checking....
24  private ArrayDeque<CardPile> _undoStack = new ArrayDeque<CardPile>();
25
26  //============================================================== constructor
27  public GameModel()
28  {
29  _allPiles = new ArrayList<CardPile>();
30
31  _freeCells = new CardPile[4];
32  _tableau = new CardPileTableau[8];
33  _foundation = new CardPile[4];
34
35  //... Create empty piles to hold "foundation"
36  for (int pile = 0; pile < _foundation.length; pile++)
37  {
38  _foundation[pile] = new CardPileFoundation();
39  _allPiles.add(_foundation[pile]);
40  }
41  //... Create empty piles of Free Cells.
42  for (int pile = 0; pile < _freeCells.length; pile++)
43  {
44  _freeCells[pile] = new CardPileFreeCell();
45  _allPiles.add(_freeCells[pile]);
46  }
47  //... Arrange the cards into piles.
48  for (int pile = 0; pile < _tableau.length; pile++)
49  {
50  _tableau[pile] = new CardPileTableau();
51  _allPiles.add(_tableau[pile]);
52  }
53
54  _changeListeners = new ArrayList<ChangeListener>();
```

```
55
56  reset();
57  }
58  //=========================================================================== reset
59  public void reset()
60  {
61  Deck deck = new Deck();
62  deck.shuffle();
63
64  //... Empty all the piles.
65  for (CardPile p : _allPiles)
66  {
67  p.clear();
68  }
69  //... Deal the cards into the piles.
70  int whichPile = 0;
71  for (Card crd : deck)
72  {
73  _tableau[whichPile].pushIgnoreRules(crd);
74  whichPile = (whichPile + 1) % _tableau.length;
75  }
76  //... Tell interested parties (ex, the View) that things have changed.
77  _notifyEveryoneOfChanges();
78  }
79
80  //TODO: Needs to be simplified having methods that both
81  // return a pile by number, and the array of all piles.
82  //=========================================================================== iterator
83  public Iterator<CardPile> iterator()
84  {
85  return _allPiles.iterator();
86  }
87  //============================================================ getTableauPile
88  public CardPile getTableauPile(int i)
89  {
90  return _tableau[i];
91  }
92  //============================================================ getTableauPiles
93  public CardPile[] getTableauPiles()
94  {
95  return _tableau;
96  }
97  //============================================================ getFreeCellPiles
98  public CardPile[] getFreeCellPiles()
99  {
100 return _freeCells;
101 }
102 //============================================================ getFreeCellPile
103 public CardPile getFreeCellPile(int cellNum)
104 {
105 return _freeCells[cellNum];
106 }
107 //============================================================ getFoundationPiles
108 public CardPile[] getFoundationPiles()
109 {
110 return _foundation;
111 }
112 //============================================================ getFoundationPile
113 public CardPile getFoundationPile(int cellNum)
114 {
115 return _foundation[cellNum];
116 }
117 //============================================================ moveFromPileToPile
118 public boolean moveFromPileToPile(CardPile source, CardPile target)
119 {
120 boolean result = false;
121 if (source.size() > 0)
```

```
122  {
123  Card crd = source.peekTop();
124  if (target.rulesAllowAddingThisCard(crd))
125  {
126  target.push(crd);
127  source.pop();
128  _notifyEveryoneOfChanges();
129  //... Record on undo stack.
130  _undoStack.push(source);
131  _undoStack.push(target);
132  result = true;
133  }
134  }
135  return result;
136  }
137  //================================================= _forceMoveFromPileToPile
138  // No checking. Not recorded in undoStack. Used by undo.
139  private void _forceMoveFromPileToPile(CardPile source, CardPile target)
140  {
141  if (source.size() > 0)
142  {
143  target.push(source.pop());
144  _notifyEveryoneOfChanges();
145  }
146  }
147  //============================================================= makeAllPlays
148 /* public void makeAllPlays()
149  {
150  boolean worthTrying; // Set true if a move was made.
151  do
152  {
153  worthTrying = false; // Assume nothing is going to be moved.
154  //... Try moving each of the free cells to graveyard.
155  for (CardPile freePile : _freeCells)
156  {
157  for (CardPile gravePile : _foundation)
158  {
159  worthTrying |= moveFromPileToPile(freePile, gravePile);
160  }
161  }
162  //... Try moving each of the player piles to graveyard.
163  for (CardPile cardPile : _tableau)
164  {
165  for (CardPile gravePile : _foundation)
166  {
167  worthTrying |= moveFromPileToPile(cardPile, gravePile);
168  }
169  }
170
171  } while (worthTrying);
172  } */
173  //========================================================== addChangeListener
174  public void addChangeListener(ChangeListener someoneWhoWantsToKnow)
175  {
176  _changeListeners.add(someoneWhoWantsToKnow);
177  }
178  //================================================ _notifyEveryoneOfChanges
179  private void _notifyEveryoneOfChanges()
180  {
181  for (ChangeListener interestedParty : _changeListeners)
182  {
183  interestedParty.stateChanged(new ChangeEvent("Game state changed."));
184  }
185  }
186 }
```

UICardPanel.java

```
1  // Description: JPanel that displays cards, and manages the mouse.
2  // Cards are in three groups:
3  // * Tableau. The initial cards are in a "tableau" consisting of
4  // 8 piles, with 7 cards in the first four, and 6 in the second four.
5  // Cards can be removed from here. Cards from other tableau piles
6  // or from free cells can be played on either an empty tableau pile,
7  // or on a card with a one higher face value and of the opposite color.
8  // * Free cells. There are four "free cells" where single cards can
9  // be temporarily stored.
10 // * Foundation. Card suits are built up here. Only Aces can be
11 // placed on empty cells and successive cards must be one higher
12 // of the same suit. No cards can be removed from the foundation.
13 //
14 // Communication with the model:
15 // * The mouse can drag cards around. When a dragged card is
16 // dropped on a pile, the mouseReleased listener calls on the
17 // model to move the card from one pile to another.
18 // The "rules" implemented by the piles may prevent this, but
19 // that's not a problem, because it simply won't be moved, and
20 // when redrawn, will show up where it originally was.
21 // * The other interaction between the model and this "mod" of the
22 // model is that this class implements the ChangeListener interface,
23 // and registers itself with the model so that it's called whenever
24 // the model changes. The stateChanged method that is called when
25 // this happens only does a repaint and clear of the dragged card info.
26
27 package freecelll;
28
29 import java.awt.*;
30 import java.awt.event.*;
31 import javax.swing.*;
32 import java.util.*;
33 import javax.swing.event.*;
34
36
37 class UICardPanel extends JComponent implements
38  MouseListener,
39  MouseMotionListener,
40  ChangeListener {
41  //================================================================ constants
42  private static final int NUMBER_OF_PILES = 8;
43
44  //... Constants specifying position of display elements
45  private static final int GAP = 10;
46  private static final int FOUNDATION_TOP = GAP;
47  private static final int FOUNDATION_BOTTOM = FOUNDATION_TOP + Card.CARD_HEIGHT;
48
49  private static final int FREE_CELL_TOP = GAP + FOUNDATION_TOP;
50  private static final int FREE_CELL_BOTTOM = FREE_CELL_TOP + Card.CARD_HEIGHT;
51
52  private static final int TABLEAU_TOP = 2 * GAP +
53  Math.max(FOUNDATION BOTTOM, FREE CELL BOTTOM);
54  private static final int TABLEAU_INCR_Y = 15;
55  private static final int TABLEAU_START_X = GAP;
56  private static final int TABLEAU_INCR_X = Card.CARD_WIDTH + GAP;
57
58  private static final int DISPLAY_WIDTH = GAP + NUMBER_OF_PILES *
    TABLEAU_INCR_X;
59  private static final int DISPLAY_HEIGHT = TABLEAU_TOP + 3 * Card.CARD_HEIGHT +
    GAP;
60
61  private static final Color BACKGROUND_COLOR = new Color(0, 200, 0);
62
63  //================================================================ fields
65  //private int _initX = 0; // x coord - set from drag
```

```
66  // private int _initY = 0; // y coord - set from drag
67
69  private int _dragFromX = 0; // Displacement inside image of mouse press
70  private int _dragFromY = 0;
71
72  //... Selected card and its pile for dragging purposes.
73  private Card _draggedCard = null; // Current draggable card
74  private CardPile _draggedFromPile = null; // Which pile it came from
75
76  //... Remember where each pile is located.
77  private IdentityHashMap<CardPile, Rectangle> _whereIs =
78  new IdentityHashMap<CardPile, Rectangle>();
79
80  //private boolean _autoComplete = false;
81
82  private GameModel _model;
83  //============================================================== constructor
85  UICardPanel(GameModel model)
86  {
87  //... Save the model.
88  _model = model;
89
90  //... Initialize graphics
91  setPreferredSize(new Dimension(DISPLAY_WIDTH, DISPLAY_HEIGHT));
92  setBackground(Color.blue);
93
94  //... Add mouse listeners.
95  this.addMouseListener(this);
96  this.addMouseMotionListener(this);
97
98  //... Set location of all piles in model
99  int x = TABLEAU_START_X; // Initial x position.
100 for (int pileNum = 0; pileNum < NUMBER_OF_PILES; pileNum++)
101 {
102 CardPile p;
103 if (pileNum < 4)
104 {
105 p = _model.getFreeCellPile(pileNum);
106 _whereIs.put(p, new Rectangle(x, FREE_CELL_TOP, Card.CARD_WIDTH,
107 Card.CARD_HEIGHT));
108 } else {
109 p = _model.getFoundationPile(pileNum - 4);
110 _whereIs.put(p, new Rectangle(x, FOUNDATION_TOP, Card.CARD_WIDTH,
111 Card.CARD_HEIGHT));
112 }
113
114 p = _model.getTableauPile(pileNum);
115 _whereIs.put(p, new Rectangle(x, TABLEAU_TOP, Card.CARD_WIDTH,
116 3 * Card.CARD_HEIGHT));
117
118 x += TABLEAU_INCR_X;
119 }
120
121 //... Make sure model calls us whenever something changes
122 _model.addChangeListener(this);
123 }
124 //============================================================ paintComponent
126 @Override public void paintComponent(Graphics g)
127 {
128 //... Paint background.
129 int width = getWidth();
130 int height = getHeight();
131 g.setColor(BACKGROUND_COLOR); // in order not to see visual artifacts
132 g.fillRect(0, 0, width, height);//, because of the override
133 g.setColor(Color.BLACK); // Restore pen color.
134
135 //... Display each pile.
```

```
136 for (CardPile pile : _model.getFreeCellPiles())
137 {
138 _drawPile(g, pile, true);
139 }
140 for (CardPile pile : _model.getFoundationPiles())
141 {
142 _drawPile(g, pile, true);
143 }
144 for (CardPile pile : _model.getTableauPiles())
145 {
146 _drawPile(g, pile, false);
147 }
148
149 //... Draw the dragged card, if any
150 if (_draggedCard != null)
151 {
152 _draggedCard.draw(g);
153 }
154 }
155 //================================================================ _drawPile
156 private void _drawPile(Graphics g, CardPile pile, boolean topOnly)
157 {
158 Rectangle loc = _whereIs.get(pile);
159 g.drawRect(loc.x, loc.y, loc.width, loc.height);
160 int y = loc.y;
161 if (pile.size() > 0)
162 {
163 if (topOnly)
164 {
165 Card card = pile.peekTop();
166 if (card != _draggedCard)
167 {
168 //... Draw only non-dragged card.
169 card.setPosition(loc.x, y);
170 card.draw(g);
171 }
172 } else {
173 //... Draw all cards.
174 for (Card card : pile)
175 {
176 if (card != _draggedCard)
177 {
178 //... Draw only non-dragged card.
179 card.setPosition(loc.x, y);
180 card.draw(g);
181 y += TABLEAU_INCR_Y;
182 }
183 }
184 }
185 }
186 }
187 //================================================================ mousePressed
188 public void mousePressed(MouseEvent e)
189 {
190 int x = e.getX(); // Save the x coord of the click
191 int y = e.getY(); // Save the y coord of the click
192
193 //... Find card image this is in. Check top of every pile.
194 _draggedCard = null; // Assume not in any image.
195 for (CardPile pile : _model)
196 {
197 if (pile.isRemovable() && pile.size() > 0)
198 {
199 Card testCard = pile.peekTop();
200 if (testCard.isInside(x, y))
201 {
202 _dragFromX = x - testCard.getX(); // how far from left
```

```
203 _dragFromY = y - testCard.getY(); // how far from top
204 _draggedCard = testCard; // Remember what we're dragging.
205 _draggedFromPile = pile;
206 break; // Stop when we find the first match.
207 }
208 }
209 }
210 }
211 //============================================================ stateChanged
212 // Implementing ChangeListener means we had to define this.
213 // Because we added ourselves as a change listener in the model,
214 // This method will be called whenever anything changes in the model.
215 // All we have to do is repaint.
216 public void stateChanged(ChangeEvent e)
217 {
218 _clearDrag(); // Perhaps not needed, but just making sure.
219 this.repaint();
220 }
221 /* //============================================================ setAutoCompletion
222 // Called from other parts of user interface.
223 void setAutoCompletion(boolean autoComplete) {
224 _autoComplete = autoComplete;
225 }*/
226 //============================================================ mouseDragged
228 public void mouseDragged(MouseEvent e)
229 {
230 if (_draggedCard == null)
231 {
232 return; // Non-null if pressed inside card image.
233 }
234 int newX;
235 int newY;
236
237 newX = e.getX() - _dragFromX;
238 newY = e.getY() - _dragFromY;
239
240 //... Don't move the image off the screen sides
241 newX = Math.max(newX, 0);
242 newX = Math.min(newX, getWidth() - Card.CARD_WIDTH);
243
244 //... Don't move the image off top or bottom
245 newY = Math.max(newY, 0);
246 newY = Math.min(newY, getHeight() - Card.CARD_HEIGHT);
247
248 _draggedCard.setPosition(newX, newY);
249
250 this.repaint(); // Repaint because position changed.
251 }
252 //============================================================ mouseReleased
253 public void mouseReleased(MouseEvent e)
254 {
255 //... Check to see if something was being dragged.
256 if (_draggedFromPile != null)
257 {
258 int x = e.getX();
259 int y = e.getY();
260 CardPile targetPile = _findPileAt(x, y);
261 if (targetPile != null)
262 {
263 //... Move card. This may not move if illegal.
264 _model.moveFromPileToPile(_draggedFromPile, targetPile);
265 /*if (_autoComplete)
266 {
267 //... Check to see if any cards can go to foundation piles.
268 _model.makeAllPlays();
269 }*/
270 }
```

```
271  _clearDrag();
272  this.repaint();
273  }
274  }
275  //============================================================ _clearDrag
276  // After mouse button is released, clear the drag info, otherwise
277  // paintComponent will still try to display a dragged card.
278  private void _clearDrag()
279  {
280  _draggedCard = null;
281  _draggedFromPile = null;
282  }
283  //============================================================ _findPileAt
284  private CardPile _findPileAt(int x, int y)
285  {
286  for (CardPile pile : _model)
287  {
288  Rectangle loc = _whereIs.get(pile);
289  if (loc.contains(x, y))
290  {
291  return pile;
292  }
293  }
294
295  return null; // Not found.
296  }
297  //=============================================== Ignore other mouse events.
298  public void mouseMoved (MouseEvent e) {} // ignore these events
299  public void mouseEntered(MouseEvent e) {} // ignore these events
300  public void mouseClicked(MouseEvent e) {} // ignore these events
301  public void mouseExited(MouseEvent e) { ; }
302 }
```

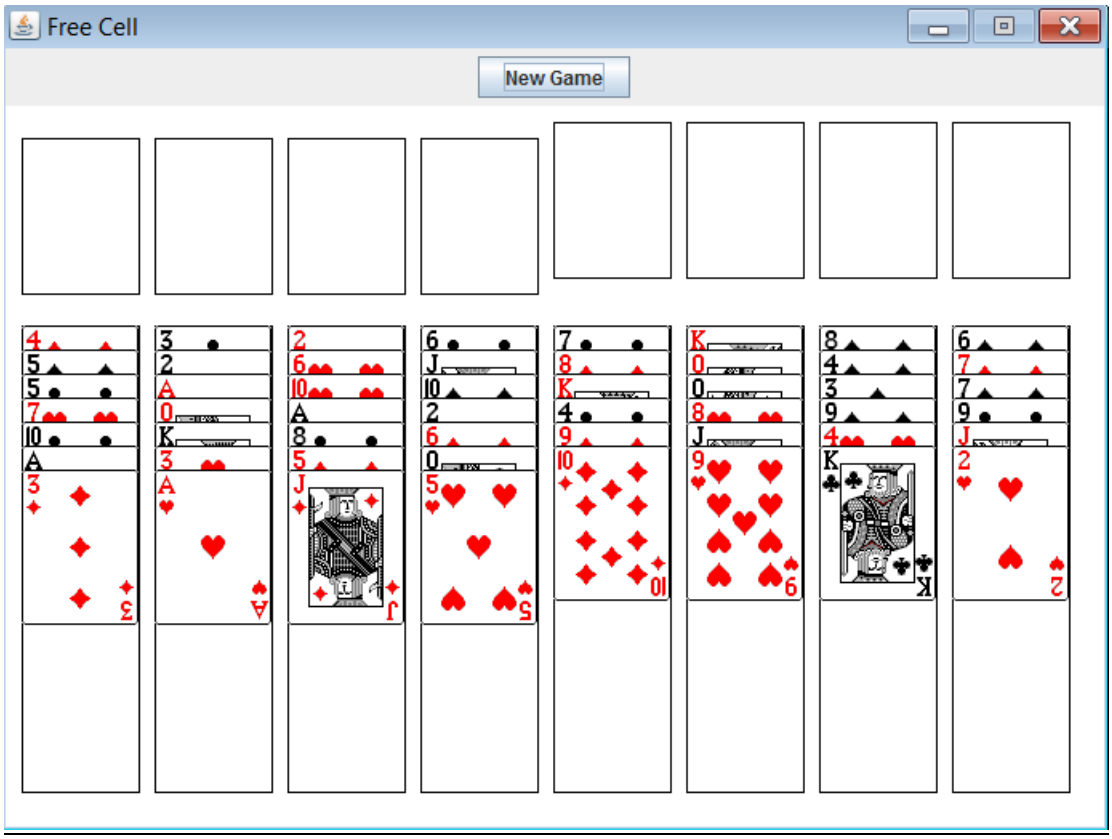
UIFreeSell.java

```
1  // Description: Free Cell solitaire program.
2  // Main program / JFrame. Adds a few components and the
3  // main graphics area, UICardPanel, that handles the mouse and painting.
4
5  package freecelll;
6
7  import java.awt.*;
8  import java.awt.event.*;
9  import javax.swing.*;
10
12 public class UIFreeSell extends JFrame
13 {
14  //============================================================== fields
15  private GameModel _model = new GameModel();
16
17  private UICardPanel _boardDisplay;
18
19 /* private JCheckBox _autoCompleteCB = new JCheckBox("Auto Complete");*/
20  //================================================================ main
21  public static void main(String[] args)
22  {
23  //... Doing all GUI initialization on EDT thread. This is the
24  // correct way.
25
26  SwingUtilities.invokeLater(new Runnable() {
27  public void run() {
28  new UIFreeSell();
29  }
30  });
31  }
32  //============================================================ constructor
33  public UIFreeSell()
```

```
34  {
35  _boardDisplay = new UICardPanel(_model);
36
37  //... Create button and check box.
38  JButton newGameBtn = new JButton("New Game");
39  newGameBtn.addActionListener(new ActionNewGame());
40
41 /* _autoCompleteCB.setSelected(true);
42  _autoCompleteCB.addActionListener(new ActionAutoComplete());
43  _boardDisplay.setAutoCompletion(_autoCompleteCB.isSelected());*/
44
45  //... Do layout
46  JPanel controlPanel = new JPanel(new FlowLayout());
47  controlPanel.add(newGameBtn);
48 /* controlPanel.add(_autoCompleteCB);*/
49
50  //... Create content pane with graphics area in center (so it expands)
51  JPanel content = new JPanel();
52  content.setLayout(new BorderLayout());
53  content.add(controlPanel, BorderLayout.NORTH);
54  content.add(_boardDisplay, BorderLayout.CENTER);
55
56  //... Set this window's characteristics.
57  setContentPane(content);
58  setTitle("Free Cell");
59  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
60  pack();
61  setLocationRelativeTo(null);
62  setResizable(false);
63  setVisible(true);
64  }
66  class ActionNewGame implements ActionListener
67  {
68  public void actionPerformed(ActionEvent evt)
69  {
70  _model.reset();
71  }
72  }
74 /* class ActionAutoComplete implements ActionListener {
75  public void actionPerformed(ActionEvent evt) {
76  _boardDisplay.setAutoCompletion(_autoCompleteCB.isSelected());
77  }
78  } */
79 }
```

Контролен Пример от изпълнението на програмата

Начало на Играта Free Cell



Край на Играта - Всички Карти са подредени в основните клетки