**1.6 Работа с Графикой.**

В компьютерном представлении рисунок – это набор чисел. Числа задают цвет каждого пиксела на экране компьютера и хранятся в буфере экрана . Видеокарта компьютера читает содержимое этого буфера с огромной скоростью (пробегая буфер несколько десятков сотен раз в секунду) и расцвечивает пикселы экрана в соответствии с числами, хранимыми в буфере. Всякий раз, когда возникает необходимость изменить цвет того или иного пиксела, компьютер вставляет новые числа в буфер экрана, и долю секунды спустя новая информация отображается на мониторе.

Набор чисел не обязательно должен хранится в экранном буфере, его можно хранить в произвольном месте, например, в виде файла на жестком диске. В Java существуют стандартные классы и процедуры копирования изображений из одной части памяти в другую, получения изображений из файлов, отображения изображений на экране компьютера.

Для представления изображений в Языке Java используется стандартный класс **java.awt.Image**. Каждый объект типа Image содержит информацию о конкретном изображении. Существует два типа объектов Image. Один тип - это изображения, хранимые в виде файлов. Другой тип – это изображения, хранимые в памяти компьютера.

Каждое изображение представлено с помощью набора чисел, но представление это может быть произведено не одним-единственным образом. Так, для файлов изображений существует два стандартных способа кодирования изображений, используемых в Java. Один способ используется для создания GIF изображений, другой – для создания JPEG-изображений. Соответствующие им файлы имеют расширения gif и jpg или jpeg. Тот и другой формат являются сжатыми, в них уменьшен объем памяти, требуемый для хранения изображения.

В классе Applet определен метод **getImage**, этот метод используется для загрузки изображения, хранимого в виде файла GIF или JPEG, например,

img = getImage(getCodeBase(), “ace.gif”);

Эта инструкция приведет к появлению объекта изображения. Второй аргумент в методе – это имя файла с изображением. Первый аргумент – это директория, в которой расположен файл с рисунком, значение getCodeBase() соответствует тому, что файл с рисунком находится в том же каталоге, что и содержащий его Апплет. После того как получен объект Рисунка, его можно отобразить в любом графическом контексте. Наиболее часто при этом используется **метод** **PaintComponent() в компоненте JPanel** **(или в каком-либо ином компоненте)** :

**d.drawImage ( img, x, y, this )**

Эта команда используется для отображения рисунка в компоненте. Параметры **x** и **y –** это положение верхнего левого угла **Рисунка**, а размер Прямоугольника будет таким, чтобы весь рисунок был отображен в натуральную величину. **Четвертый параметр –** это компонент, где рисунок будет отображен. При обращении к getImage() запрашиваемый файл не будет скачиваться немедленно. Файл будет запрошен тогда, когда он должен отобразиться в первый раз. Объект **Image** просто запоминает местонахождения файла. Этот метод начинает скачивание файла, но не дожидается его завершения. **Четвертый параметр –** это объект, который наблюдает за изображением. После того как рисунок будет полностью получен, система сообщает наблюдателю за изображением о том, что рисунок теперь стал доступным. В качестве наблюдателя может использоваться любой **Компонент JComponent**. Если Известно, что рисунок уже загружен, то четвертый параметр можно указать в виде null.

Существуют и другие методы Вызова **drawImage()**. **Например, задается размер изображения:**

**g.drawImage( img, x, y, width, height, this );**

Можно нарисовать лишь **часть изображения:**

g.drawImage( img, dest\_x1, dest\_y1, dest\_x2, dest\_y2,

source\_x1, source\_y1, source\_x2, source\_y2, this );

Здесь координаты **source\_x1, source\_y1, source\_x2 и source\_y2** задают Верхний Левый и Нижний Правый Углы той части Изображения, которая будет Показана. Координаты **dest\_x1, dest\_y1, dest\_x2, dest\_y2** – это Координаты Графического Контекста. При необходимости будет произведено соответствующее координатам сжатие или растяжение рисунка. Далее мы рассмотрим пример с картами. Этот пример будет использовать только один рисунок для отображения всех 52 карт (рис. 1.23).

Для отображения только одной карты следует указать координаты отображаемого фрагмента Рисунка. Метод этот Используется в классическом примере Игры На Угадывание ( Нужнр Угадать, какая придет следующая карта – Более Старшая или Более Младшая).

В этом Апплете карты отображаются с использованием следующего метода. В Переменной типа **Image** и Именем **cardImages** хранится рисунок с изображениями 52 карт (рис. 1.23). Размер каждой карты составляет 40 х 60 пикселов. На основе этого создаются изображения для каждой карты в отдельности (листинг 1.25).

**Листинг 1.25.** файл **drawCard()**

void drawCard(Graphics g, Card card,int x, int y) {  
 if (card == null) {  
 // Описание Карты рубашкой Вверх.  
 g.setColor(Color.blue);  
 g.fillRect(x, y, 40, 60);  
 g.setColor(Color.white);  
 g.drawRect(x+3, y+3, 33, 53);  
 g,drawRect(x+4, y+4, 31, 51);  
 }  
 else{  
 int row = 0; // Определение Ряда, в котором нарисована текущая карта.  
 switch (card.getSuit()) {  
 case Card.CLUBS: row = 0; break;  
 case Card.HEARTS: row = 1; break;  
 case Card.SPADES: row = 2; break;  
 case Card.DIAMOND: row = 3; break;  
 }  
 int sx, sy; // Координаты Верхнего Левого Угла Карты  
 // в заданном Рисунке.  
 sx = 40\*(card.getValue() - 1);  
 sy = 60 \* row;  
 g.drawImage(cardImages, x, y, x+40, y+60, sx, sy, sx+40, sy+60, this);  
 }  
}

Далее приводится программный код, состоящий из 4 файлов. Основная программа – Cards.java. Кроме нее потребуется наличие файлов Card.java, Hand.java, Deck.java (листинг 1.26, 1.27, и 1.28).

**Листинг 1.26.** Файл **Cards.java**

/\*  
Простая Карточная Игра "Угадай, что Дальше"  
 \*/  
import java.awt.\*;  
import java.awt.event.\*;  
import javax.swing.\*;  
public class Cards extends JApplet {  
 java.awt.Image cardsImages;  
 public void init() {  
 cardImages = getImage(getCodeBase(), "smallcards.gif");  
 setBackground(new Color(130, 50, 49));  
 HighLowCanvas board = new HighLowCanvas();  
 getContentPane().add(board, BorderLayout.*CENTER*);  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setBackground(new Color(220, 200, 180));  
 getContentPane().add(buttonPanel, BorderLayout.*SOUTH*);  
 JButton higher = new JButton("Starshe");  
 higher.addActionListener(board);  
 buttonPanel.add(lower);  
 JButton newGame = new JButton("New Geme");  
 newGame.addActionListener(board);  
 buttonPanel.add(newGame);  
 }  
 public Insets getInsets() {  
 return new Insets(3, 3, 3, 3);  
 }  
 class HighLowCanvas extends JPanel implements ActionListener {  
 Deck deck; // Колода Карт.  
 Hand hand; // Сбрасываемая карта.  
 String message; // Сообщение.  
 boolean gameInProgress; // Состояние Игры.  
 Font bigFont; // Шрифт Отображения Сообщения.  
 Font smallFont; // Шрифт Для Отображения Карт.  
 HighLowCanvas() {  
 setBackground(new Color(0, 120, 0));  
 setForeground(Color.*green*);  
 smallFont = new Font("SansSerif", Font.*PLAIN*, 12);  
 bigFont = new Font("Serif", Font.*BOLD*, 14);  
 doNewGame();  
 }  
 public void actionPerformed(ActionEvent evt) {  
 String command = evt.getActionCommand();  
 if (command.equals("Starshe"))  
 doHigher();  
 else if (command.equals("Mladshe"))  
 doLower();  
 else if (command.equals("NewGame"))  
 doNewGame();  
 }  
 void doHigher() {  
 if (gameInProgress == false) {  
 message = "Click \" New Game\" first!";  
 repaint();  
 return;  
 }  
 hand.addCard(deck.dealCard());  
 int cardCt = hand.getCardCount();  
 Card thisCard = hand.getCard(cardCt - 1);  
 // Описание предыдуәй Карты.  
 Card prevCard = hand.getCard(cardCt - 2);  
 if (thisCard.getValue() < prevCard.getValue()) {  
 gameInProgress = false;  
 message = "Ploho poluchilos!";  
 }  
 else if (cardCt == 4) {  
 gameInProgress = false;  
 message = "Ok! You Won!";  
 }  
 else {  
 message = "You are True! Try Again" + cardCt + ".";  
 }  
 repaint();  
 return;  
 }  
 void doLower() {  
 if (gameInProgress == false){  
 message = "At first Press \"New Game\" !";  
 repaint();  
 return;  
 }  
 hand.addCard(deck.dealCard());  
 int cardCt = hand.getCardCount();  
 Card thisCard = hand.getCard(cardCt - 1);  
 Card prevCard = hand.getCard(cardCt - 2);  
 if (thisCard.getValue() > prevCard.getValue()) {  
 gameInProgress = false;  
 message = "You Loss!!!"  
 }  
 else if (thisCard.gerValue() == prevCard.getValue()) {  
 gameInProgress = false;  
 message = "Not True.";  
 }  
 else if (cardCt == 4) {  
 gameInProgress = false;  
 message = "Excelent! You Won!";  
 }  
 else{  
 message = "True! Try again" + cardCt + ".";  
 }  
 repaint();  
 }  
 void doNewGame() {  
 if (gameInProgress) {  
 message = "Do End The Game!";  
 repaint();  
 return;  
 }  
 deck = new Deck();  
 hand = new Hand();  
 deck.shuffle();  
 hand.addCard(deck.dealCard());  
 message = "Guess When the Next Card, Higher or Lower?";  
 gameInProgress = true;  
 repaint();  
 }  
 public void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 g.setFont(bigFont);  
 g.drawString(message, 10, getSize().height-10);  
 g.setFont(smallFont);  
 int cardCt = hand.getCardCoumt();  
 for (int i = 0; i < cardCt; i++)  
 drawCard(g, hand.getCard(i), 30 + i \* 70, 10);  
 if (gameInProgress)  
 drawCard(g, null, 30 + cardCt \* 70, 10);  
 }  
 void drawCard(Graphics g, Card card, int x, int y){  
 if (card == null) {  
 g.setColor(Color.*blue*);  
 g.fillRect(x, y, 40, 60);  
 g.setColor(Color.*white*);  
 g.drawRect(x+3, y+3, 33,53);  
 g.drawRect(x+4, y+4, 31, 51);  
 }  
 else{  
 int row = 0;  
 switch(card.getSuit()) {  
 case Card.*CLUBS*: row = 0; break;  
 case Card.*HEARTS*: row = 1; break;  
 case Card.*SPADES*: row = 2; break;  
 case Card.*DIAMONDS*: row = 3; break;  
 }  
 int sx, sy;  
 sx = 40\*(card.getValue() - 1);  
 sy = 60\*row;  
 g.drawImage(cardImages, x, y, x+40, y+60,  
 sx, sy, sx+40, sy+60, this);  
 }  
 }  
 }  
}

**Листинг 1.27.** КодКласса **Card.java**

/\*  
 Отдельная Карта.  
 \*/  
  
public class Card {  
 public final static int  
 *SPADES* = 0, // Значение Мастей.  
 *HEARTS* = 1, // Значение Мастей.  
 *DIAMONDS* = 2, // Значение Мастей.  
 *CLUBS* = 3; // Значение Мастей.  
 public final static int  
 *ACE* = 1, // Старшинство Карт.  
 *JACK* = 11,  
 *QUEEN* = 12,  
 *KING* = 13;  
 private final int suit; // Один Из Вариантов:  
 // SPADES, HEARTS, DIAMONDS, CLUBS  
 private final int value;  
 public Card(int theValue, int theSuit) {  
 value = theValue;  
 suit = theSuit;  
 }  
 public int getSuit() {  
 return suit;  
 }  
 public int getValue() {  
 return value;  
 }  
 public String getSuitAsString() {  
 switch (suit) {  
 case *SPADES*: return "Spades";  
 case *HEARTS*: return "Hearts";  
 case *DIAMONDS*: return "Diamonds";  
 case *CLUBS*: return "Clubs";  
 default: return "??";  
 }  
 }  
 public String getValueAsString() {  
 switch (value) {  
 case 1: return "Ace";  
 case 2: return "2";  
 case 3: return "3";  
 case 4: return "4";  
 case 5: return "5";  
 case 6: return "6";  
 case 7: return "7";  
 case 8: return "8";  
 case 9: return "9";  
 case 10: return "10";  
 case 11: return "Jack";  
 case 12: return "Queen";  
 case 13: return "King";  
 default: return "??";  
 }  
 }  
 public String toString() {  
 return getValueAsString() + " of " + getSuitAsString();  
 }  
}

Код класса **Hand** приводится далее **( Листинг 1.28 )**

**Листинг 1.28** Файл **Hand.java**

import java.util.Vector;  
public class Hand {  
 private Vector hand; // Сдаваемые Карты.  
 public Hand() {  
 hand = new Vector();  
 }  
 public void clear() {  
 hand.removeAllElements();  
 }  
 public void addCard(Card c) {  
 if (c != null)  
 hand.addElement(c);  
 }  
 public void removeCard(Card c) {  
 hand.removeElement(c);  
 }  
 public void removeCard(int position) {  
 if (position >= 0 && position < hand.size())  
 hand.removeElementAt(position);  
 }  
 public int getCardCount() {  
 return hand.size();  
 }  
 public Card getCard(int position) {  
 if (position >= 0 && position < habd.size())  
 return (Card)hand.elementAt(position);  
 else  
 return null;  
 }  
 public void sortBySuit() {  
 Vector newHand = new Vector();  
 while (hand.size() > 0) {  
 int pos = 0;  
 Card c = (Card)hand.elementAt(0);  
 for (int i = 1; i < hand.size(); i++) {  
 Card c1 = (Card)hand.elementAt(i);  
 if (c1.getSuit() < c.getSuit() ||  
 (c1.getSuit() == c.getSuit() && c1.getValue() < c.getValue()))  
 {  
 pos = i;  
 c = c1;  
 }  
 }  
 hand.removeElementAt(pos);  
 newHand.addElement(c);  
 }  
 hand = newHand;  
 }  
 public void sortByValue() {  
 Vector newHand = new Vector();  
 while (hand.size() > 0) {  
 int pos = 0;  
 Card c = (Card)hand.elementAt(0);  
 for(int i = 1; i < hand.size(); i++) {  
 Card c1 = (Card)hand.elementAt(i);  
 if(c1.getValue() < c.getValue() ||  
 (c1.getValue() == c.getValue()&& c1.getSuit() < c.getSuit())) {  
 pos = i;  
 c = c1;  
 }  
 }  
 hand.removeElementAt(pos);  
 newHand.addElement(c);  
 }  
 hand = newHand;  
 }  
}

Кроме того необходим Класс, описывающий все 52 Карты ( листинг 1.29 ).

**Листинг 1.29.** файл **Deck.java.**

public class Deck {  
 private Card[] deck;  
 private int cardsUsed;  
 public Deck() {  
 deck = new Card[52];  
 int cardCt = 0;  
 for(int suit = 0; suit <= 3; suit++) {  
 for(int value = 1; value <= 13; value++) {  
 deck[cardCt] = new Card(value, suit);  
 cardCt++;  
 }  
 }  
 cardsUsed = 0;  
 }  
 public void shuffle() {  
 for(int i = 51; i > 0; i--) {  
 int rand = (int)(Math.*random*()\*(i+1));  
 Card temp = deck[i];  
 deck[i] = deck[rand];  
 deck[rand] = temp;  
 }  
 cardsUsed = 0;  
 }  
 public int cardsLeft() {  
 return 52 - cardsUsed;  
 }  
 public Card dealCard() {  
 if (cardsUsed == 52)  
 shuffle();  
 cardsUsed++;  
 return deck[cardsUsed - 1];  
 }  
}