|  |
| --- |
| Further Object-Orientated Programming : CE00527-2-190909-161209 |
| Alex J Davison DV003874 |
|  |

|  |
| --- |
| Alex J Davison  10/8/2009 |

Table of Contents

[Tutorial 1 4](#_Toc245564996)

[Code listing for your final version of the Pet class 4](#_Toc245564997)

[Code listing for your final version the PetShop class 5](#_Toc245564998)

[A sample main method testing each of these classes, with output (no user interaction is required, just hard-code the test data into the main method) 7](#_Toc245564999)

[A UML class diagram showing attributes, methods, and the relationship between the PetShop and Pet classes 9](#_Toc245565000)

[Tutorial 2 10](#_Toc245565001)

[The code listing for the new version of the Pet class, with code listing for its test class and test output 10](#_Toc245565002)

[Code 10](#_Toc245565003)

[Testing 11](#_Toc245565004)

[The design (class diagram, including associations) of the Card, Hand and Deck classes, and associated enums 12](#_Toc245565005)

[The code listing for each of these classes and enums 13](#_Toc245565006)

[Rank 13](#_Toc245565007)

[Suit 13](#_Toc245565008)

[Card 14](#_Toc245565009)

[Hand 15](#_Toc245565010)

[Deck 16](#_Toc245565011)

[A sample main method testing the Card class, with output 17](#_Toc245565012)

[A sample main method testing the Hand class, with output 19](#_Toc245565013)

[A sample main method testing the Deck class, with output 20](#_Toc245565014)

[Tutorial 3 22](#_Toc245565015)

[What is the relationship of the address object anotherAddress to the Customer object c1? 22](#_Toc245565016)

[Customer class, TestCustomer class, and their output, in your portfolio 22](#_Toc245565017)

[Customer class 22](#_Toc245565018)

[TestCustomer 23](#_Toc245565019)

[Testing 24](#_Toc245565020)

[Which of the statements in TestCustomer are no longer valid, and why? 24](#_Toc245565021)

[Which method of Address is no longer valid and why? 24](#_Toc245565022)

[Design and screenshot of the running GUI program 25](#_Toc245565023)

[Tutorial 4 26](#_Toc245565024)

[The design (class diagram, including inheritance relationships) of your new classes to represent types of pets 26](#_Toc245565025)

[The code listing for each of these classes 27](#_Toc245565026)

[Dog 27](#_Toc245565027)

[Cat 28](#_Toc245565028)

[Parrot 29](#_Toc245565029)

[Hamster 30](#_Toc245565030)

[Your design for the add Pet dialog 31](#_Toc245565031)

[The code listing for the add Pet dialog 32](#_Toc245565032)

[The additional code that you added to your application to allow Pets to be deleted 33](#_Toc245565033)

[Evidence of testing of your application 34](#_Toc245565034)

[Delete Sam Parrot 34](#_Toc245565035)

[Go to next pet 35](#_Toc245565036)

[Go to pervious pet 35](#_Toc245565037)

[Find Fluffy 36](#_Toc245565038)

[Add Pet Alex The Cat 37](#_Toc245565039)

[Tutorial 5 39](#_Toc245565040)

[The new code that you added to the Card, Hand and Deck classes to answer parts 1, 2 and 4 39](#_Toc245565041)

[Card 39](#_Toc245565042)

[Deck 41](#_Toc245565043)

[Hand 43](#_Toc245565044)

[A code listing of the class you wrote to implement Comparator for type Card 45](#_Toc245565045)

[Evidence of testing this new functionality 46](#_Toc245565046)

[Answers to the questions of Part 3 54](#_Toc245565047)

[What is the difference between a deep and a shallow copy? 54](#_Toc245565048)

[Which type of copy did you use in your clone methods and why? 54](#_Toc245565049)

[If you override the equals method which other method should you override? 54](#_Toc245565050)

[What relationship should the two methods have? 54](#_Toc245565051)

# Tutorial 1

## Code listing for your final version of the Pet class

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

import java.text.SimpleDateFormat;

/\*\*

\*

\* @author Alex J Davison

\*/

public class Pet {

private String name;

private boolean sold;

private Date dob;

private int age;

public Pet (String n, boolean s, String d) {

name = n;

sold = s;

dob = new Date(d);

}

public String toString() {

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

String s = "Name: " + name + " Sold: " + sold + " Date Of Birth: " + sdf.format(dob) + " Age: " + getAge();

return s;

}

public String getName() { return name; }

public boolean getSold() { return sold; }

public String getDOB() { return dob.toString(); }

public int getAge() {

Date now = new Date();

long diff = now.getTime() - dob.getTime();

long age = (diff / (1000L\*60\*60\*24\*365));

return (int)age; }

public void setSold (boolean sold){

this.sold = sold;

}

}

## Code listing for your final version the PetShop class

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author Alex J Davison

\*/

public class PetShop {

private ArrayList <Pet> thePets;

private String thePetShopName;

private int currentIndex;

public PetShop (String name)

{

thePetShopName = name;

thePets = new ArrayList <Pet> ();

}

public int getCurIndex() { return currentIndex;}

public void nextPet()

{

if (currentIndex< thePets.size()-1)

{

currentIndex = currentIndex + 1;

}

}

public void previousPet()

{

if (currentIndex > 0)

{

currentIndex = currentIndex - 1;

}

}

public boolean addPet( Pet newPet )

{

thePets.add(currentIndex,newPet);

return true;

}

public void outputPetShopPets()

{

System.out.println("Pet Shop Name: " + thePetShopName );

//output each player in turn

System.out.println("The players:");

for (Pet nextPet : thePets)

{

System.out.println(nextPet.toString());

}

}

public Pet findPet(String petName) {

for (Pet p : thePets) {

if (p.getName().equalsIgnoreCase(petName))

return p;

}

return null;

}

public boolean deletePet(String name) {

Pet p = findPet(name);

if (p != null) {

thePets.remove(p);

return true;

}

return false;

}

public Pet currentPet(){

return thePets.get(currentIndex);

}

}

## A sample main method testing each of these classes, with output (no user interaction is required, just hard-code the test data into the main method)

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author Alex J Davison

\*/

public class TestPetShop {

public static void main(String[] args) {

PetShop thePetShop = new PetShop("Stafford Octagons");

Pet p1 = new Pet("Alex", false,"01/29/1987");

Pet p2 = new Pet("Dainie", false,"07/07/1990");

Pet p3 = new Pet("Ed",false,"07/07/1990");

thePetShop.addPet(p1);

thePetShop.addPet(p2);

thePetShop.addPet(p3);

thePetShop.outputPetShopPets();

Pet p = thePetShop.findPet("Pete");

if (p != null) {

System.out.println(p + " found");

} else {

System.out.println("Pete is not in the pet shop");

}

p = thePetShop.findPet("Alex");

if (p != null) {

System.out.println(p + " found");

} else {

System.out.println("Pete is not in the pet shop");

}

thePetShop.deletePet("Ed");

System.out.println("Pet Shop after Ed has left: " );

thePetShop.outputPetShopPets();

thePetShop.nextPet();

thePetShop.nextPet();

thePetShop.nextPet();

thePetShop.nextPet();

thePetShop.nextPet();

thePetShop.nextPet();

System.out.println(thePetShop.currentPet());

thePetShop.previousPet();

thePetShop.previousPet();

thePetShop.previousPet();

thePetShop.previousPet();

thePetShop.previousPet();

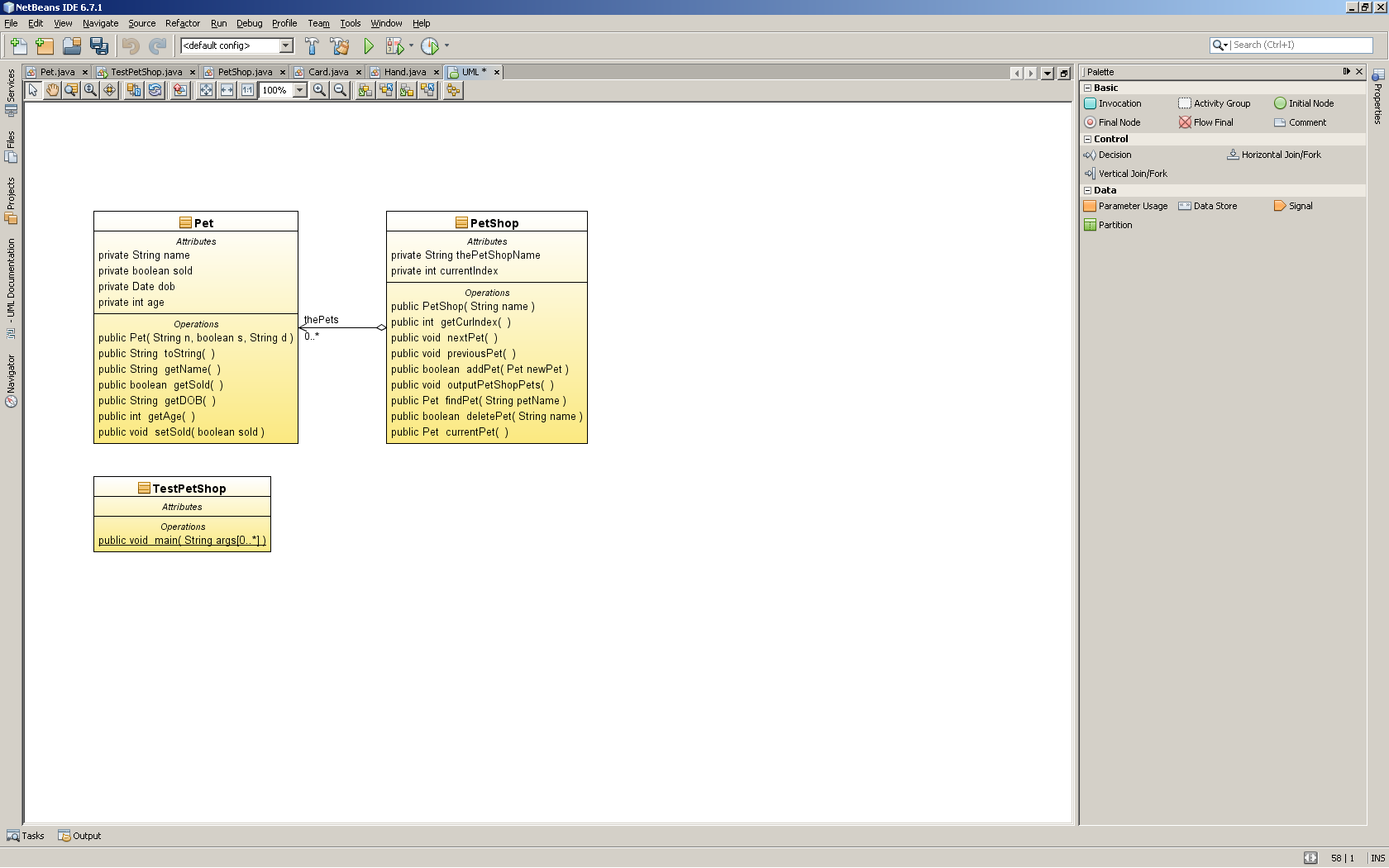
thePetShop.previousPet();

System.out.println(thePetShop.currentPet());

}

}

## A UML class diagram showing attributes, methods, and the relationship between the PetShop and Pet classes



# Tutorial 2

## The code listing for the new version of the Pet class, with code listing for its test class and test output

### Code

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

import java.text.SimpleDateFormat;

/\*\*

\*

\* @author Alex J Davison

\*/

public class Pet {

private String name;

private boolean sold;

private Date dob;

private int age;

public Pet(String n, boolean s, String d) {

name = n;

sold = s;

dob = new Date(d);

}

public String toString() {

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

String s = "Name: " + name + " Sold: " + sold + " Date Of Birth: " + sdf.format(dob) + " Age: " + getAge();

return s;

}

public String getName() {

return name;

}

public boolean getSold() {

return sold;

}

public String getDOB() {

return dob.toString();

}

public int getAge() {

Date now = new Date();

long diff = now.getTime() - dob.getTime();

long age = (diff / (1000L \* 60 \* 60 \* 24 \* 365));

return (int) age;

}

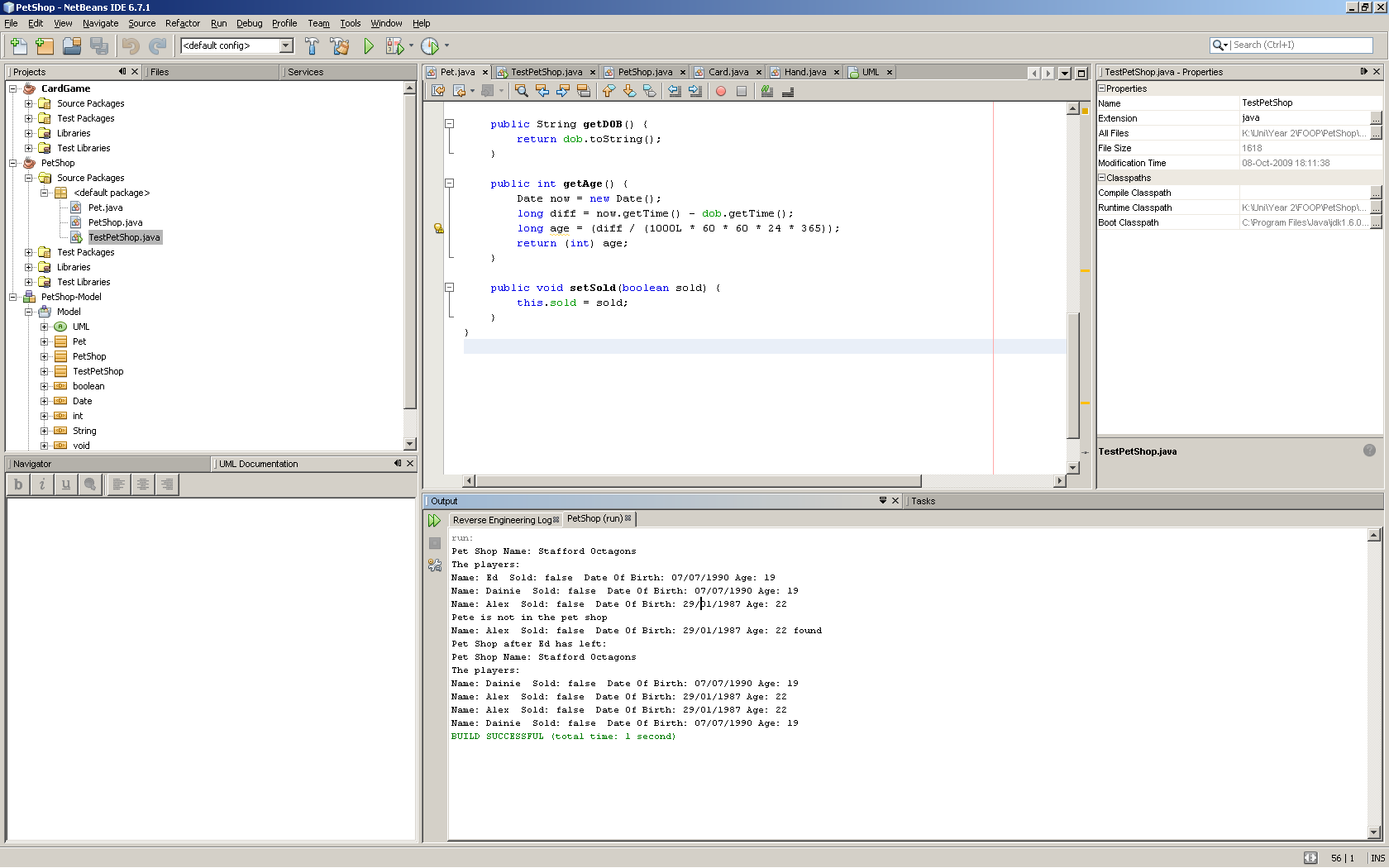
public void setSold(boolean sold) {

this.sold = sold;

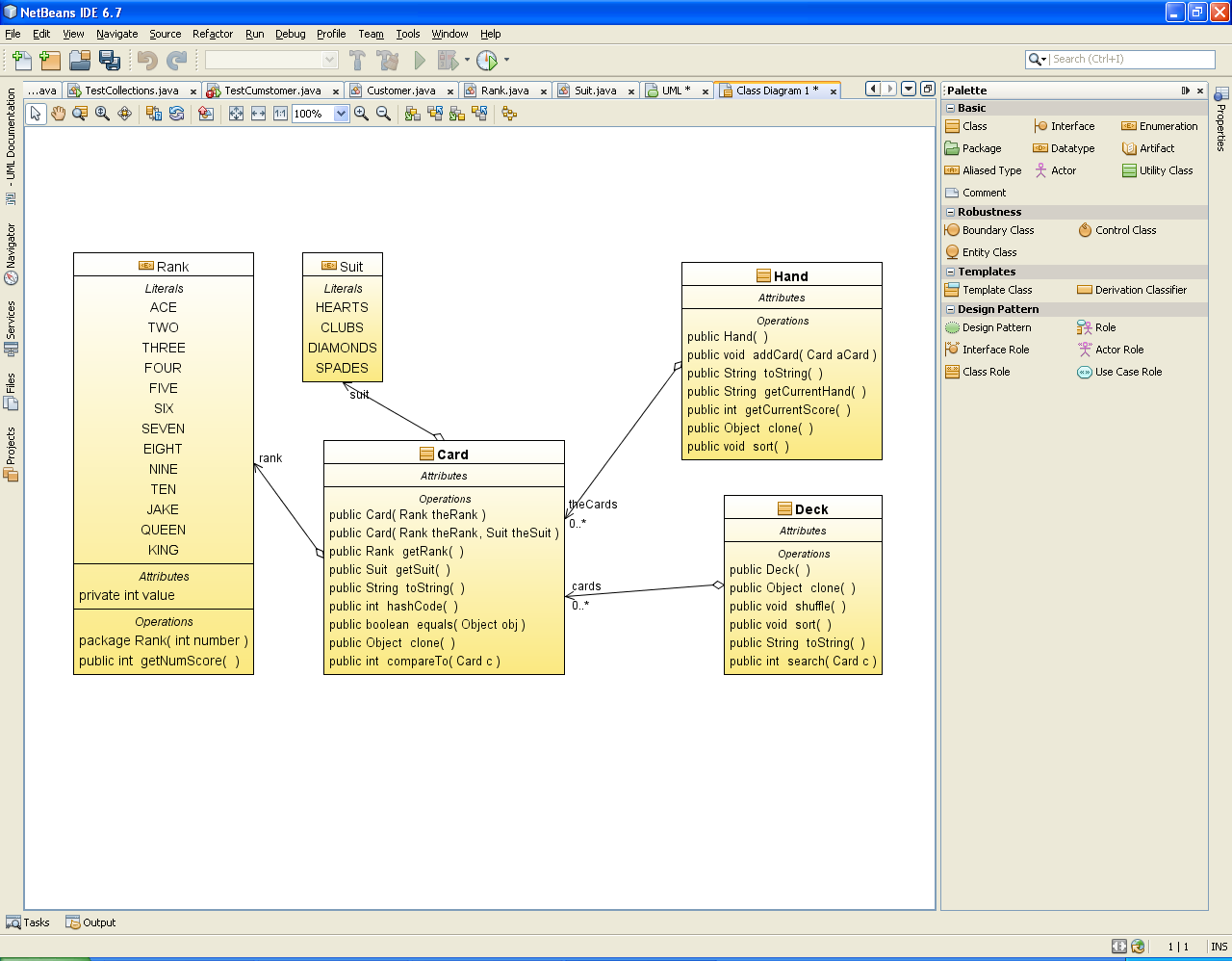
}

}

### Testing



## The design (class diagram, including associations) of the Card, Hand and Deck classes, and associated enums



## The code listing for each of these classes and enums

### Rank

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author DV003874

\*/

public enum Rank {

ACE(1),

TWO(2),

THREE(3),

FOUR(4),

FIVE(5),

SIX(6),

SEVEN(7),

EIGHT(8),

NINE(9),

TEN(10),

JAKE(10),

QUEEN(10),

KING(10);

private final int value;

Rank(int number) {

value = number;

}

public int getNumScore() {

return value;

}

}

### Suit

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author DV003874

\*/

public enum Suit {

HEARTS, CLUBS, DIAMONDS, SPADES

}

### Card

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author Alex J Davison

\*/

public class Card implements Cloneable, Comparable<Card>{

private Rank rank;

private Suit suit;

public Card(Rank theRank) {

this(theRank, Suit.HEARTS);

}

public Card(Rank theRank, Suit theSuit) {

rank = theRank;

suit = theSuit;

}

public Rank getRank() {

return rank;

}

public Suit getSuit() {

return suit;

}

public String toString() {

return (rank + " of " + suit + "\n");

}

}

### Hand

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author Alex J Davison

\*/

public class Hand {

// a class to represent a hand of cards

private Card theCards[];

private int numCards;

private static final int max = 5;

public Hand() {

theCards = new Card[max];

numCards = 0;

}

public void addCard(Card aCard) {

if (numCards < max) {

theCards[numCards++] = aCard;

}

}

public String toString() {

String s = "";

for (int i = 0; i < numCards; ++i) {

s += "\n" + theCards[i];

}

return s;

}

public String getCurrentHand() {

String s = "";

for (Card theCard : theCards) {

s = s + theCard;

}

return s;

}

public int getCurrentScore() {

int s = 0;

for (Card theCard : theCards) {

s = s + theCard.getRank().getNumScore();

}

return s;

}

}

### Deck

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author dv003874

\*/

public class Deck {

private static final List<Card> deck = new ArrayList<Card>();

static {

for (Card.Suit suit : Card.Suit.values()) {

for (Card.Rank rank : Card.Rank.values()) {

deck.add(new Card(rank, suit));

}

}

}

public static ArrayList<Card> newDeck() {

return new ArrayList<Card>(deck); // Return copy of prototype deck

}

}

## A sample main method testing the Card class, with output

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author dv003874

\*/

public class TestCard {

public static void main(String[] args) {

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

for (Card.Suit suit : Card.Suit.values()) {

for (Card.Rank rank : Card.Rank.values()) {

deck.add(new Card(rank, suit));

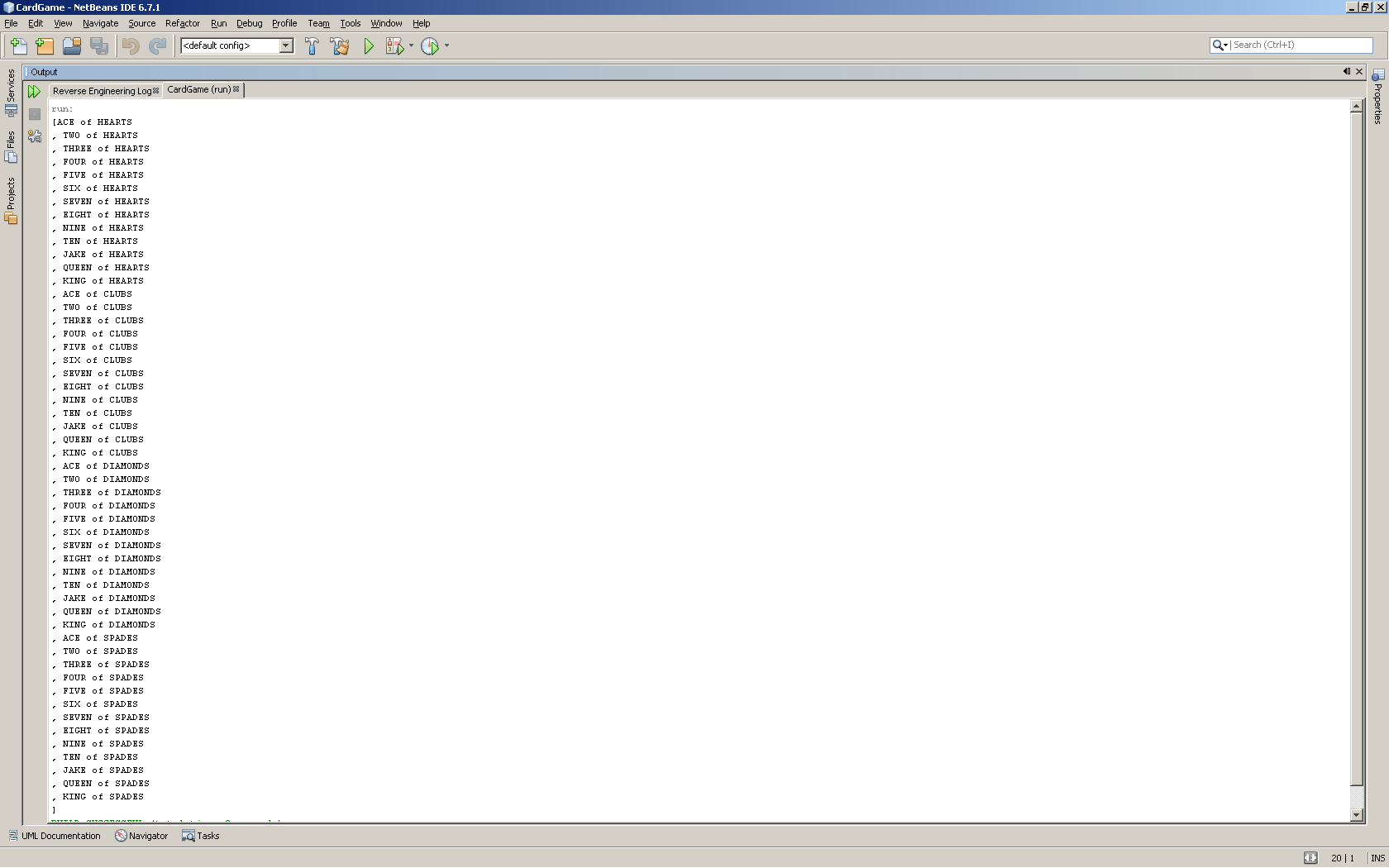
}

}

System.out.println(deck);

}

}



## A sample main method testing the Hand class, with output

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author dv003874

\*/

public class TestHand {

public static void main(String[] args) {

ArrayList<Card> deck = Deck.newDeck();

Hand player1 = new Hand();

for (int i = 0; i < 5; i++) {

player1.addCard(deck.get(i));

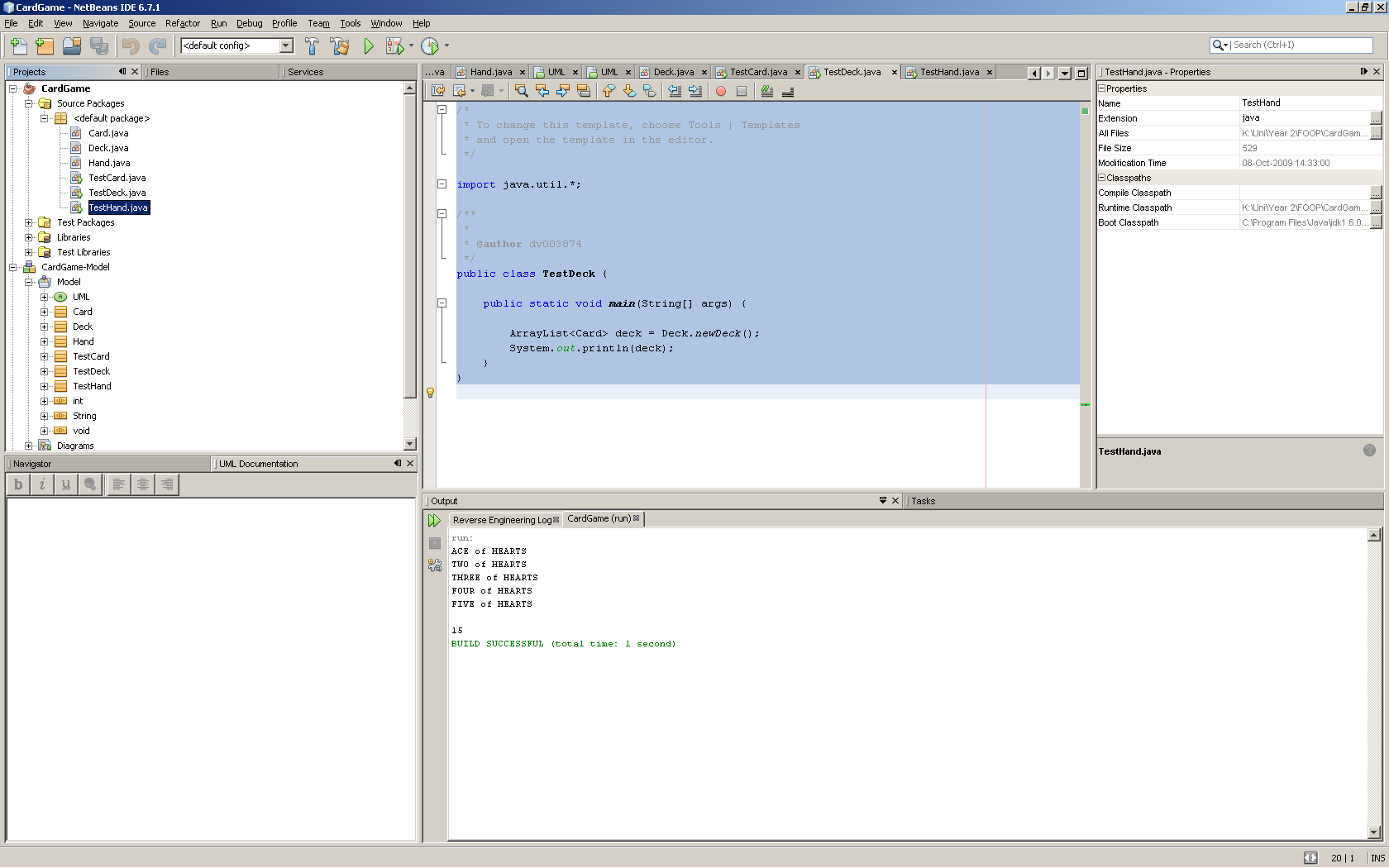
}

System.out.println(player1.getCurrentHand());

System.out.println(player1.getCurrentScore());

}

}



## A sample main method testing the Deck class, with output

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author dv003874

\*/

public class TestDeck {

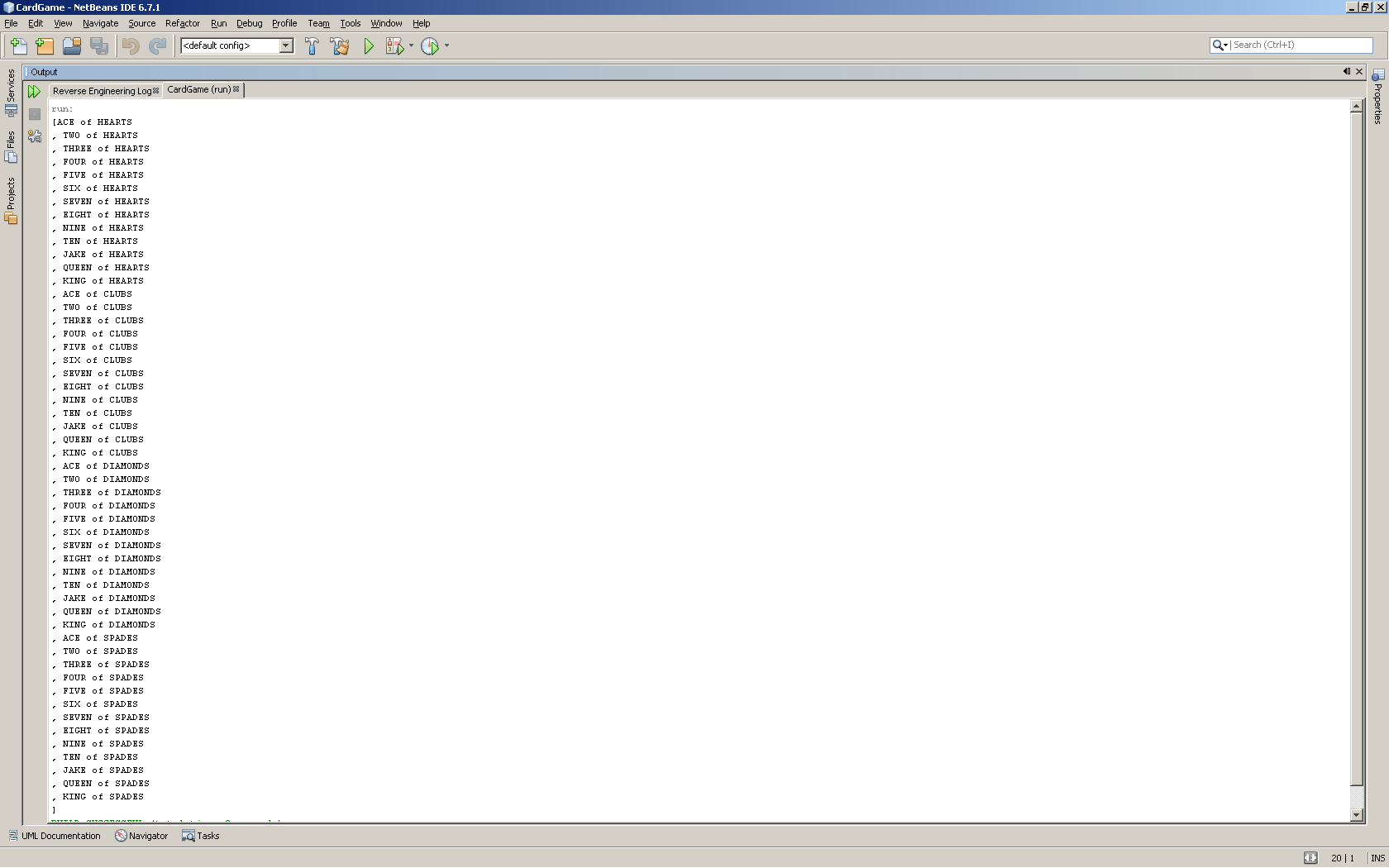
public static void main(String[] args) {

ArrayList<Card> deck = Deck.newDeck();

System.out.println(deck);

}

}



# Tutorial 3

## What is the relationship of the address object anotherAddress to the Customer object c1?

AnotherAddress can access the private member variable of c1 but does not hold a reference to the object anotherAddrees.

## Customer class, TestCustomer class, and their output, in your portfolio

### Customer class

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author Alex J Davison

\*/

public class Customer {

private String name;

private Address homeAddress, workAddress;

public class Address {

private int number;

private String street;

public Address(int no, String street) {

number = no;

this.street = street;

}

public String toString() {

return number + "\n" + street;

}

}

public Customer(String name, int houseNumber, String homeStreet) {

this.name = name;

homeAddress = new Address(houseNumber, homeStreet);

}

public Customer(String name, int houseNumber, String homeStreet, int workNumber, String workStreet) {

this.name = name;

homeAddress = new Address(houseNumber, homeStreet);

workAddress = new Address(workNumber,workStreet);

}

public void setWorkAddress(int workNumber, String workStreet){

workAddress = new Address(workNumber,workStreet);

}

public String toString() {

String s = "";

String ha = "";

String wa = "";

if(homeAddress == null){

ha = "None";

}

else{

ha = homeAddress.toString();

}

if(workAddress == null){

wa = "None";

}

else{

wa = workAddress.toString();

}

s = "Name: " + name + "\nHome Address:" + ha + "\nWork Address:" + wa;

return s;

}

}

### TestCustomer

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author Alex J Davison

\*/

public class TestCumstomer {

public static void main(String[] args) {

Customer c1 = new Customer("Cathy", 10, "Unicorn lane");

Customer.Address anotherAddress = c1.new Address(8, "Octagon Road");

c1.setWorkAddress(40,"Beaconside Rise");

// note the use of new

System.out.println(anotherAddress.toString());

System.out.println(c1.toString());

}

}

### Testing

run:

8

Octagon Road

Name: Cathy

Home Address:10

Unicorn lane

Work Address:40

Beaconside Rise

## Which of the statements in TestCustomer are no longer valid, and why?

Customer.Address anotherAddress = c1.new Address(8, "Octagon Road");

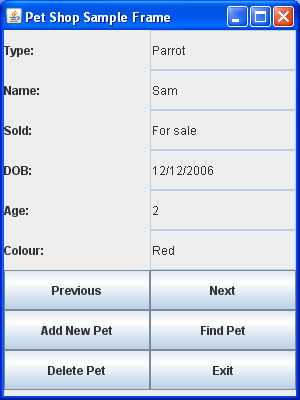
The address class is now private so you can not access the address class from outside the outer class.

## Which method of Address is no longer valid and why?

Customer.Address anotherAddress = c1.new Address(8, "Octagon Road");

The address is static so it no longer belongs to a customer object.

## Design and screenshot of the running GUI program



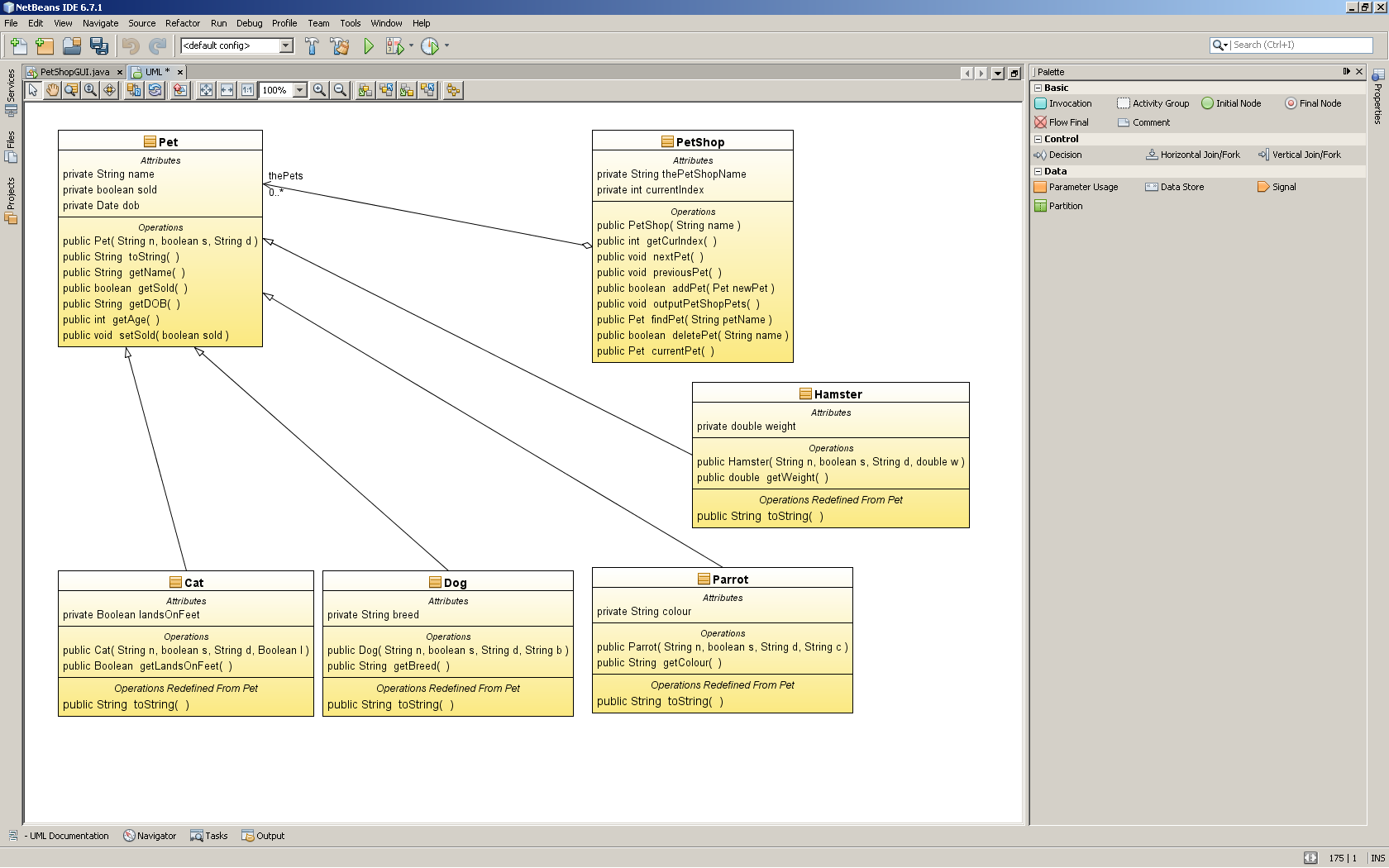
JLabels

JTextFields

JButtons

# Tutorial 4

## The design (class diagram, including inheritance relationships) of your new classes to represent types of pets



## The code listing for each of these classes

### Dog

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author dv003874

\*/

public class Dog extends Pet {

private String breed ;

public Dog (String n, boolean s, String d, String b)

{

super(n, s, d);

breed = b;

}

public String getBreed()

{

return breed;

}

public String toString() {

String s = super.toString() + " Breed:" + breed;

return s;

}

}

### Cat

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author dv003874

\*/

public class Cat extends Pet {

private Boolean landsOnFeet ;

public Cat (String n, boolean s, String d, Boolean l)

{

super(n, s, d);

landsOnFeet = l;

}

public Boolean getLandsOnFeet()

{

return landsOnFeet;

}

public String toString() {

String s = super.toString() + " Lands On Feet: " + landsOnFeet;

return s;

}

}

### Parrot

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author dv003874

\*/

public class Parrot extends Pet {

private String colour ;

public Parrot (String n, boolean s, String d, String c)

{

super(n, s, d);

colour = c;

}

public String getColour()

{

return colour;

}

public String toString() {

String s = super.toString() + " Colour:" + colour;

return s;

}

}

### Hamster

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author dv003874

\*/

public class Hamster extends Pet {

private double weight ;

public Hamster (String n, boolean s, String d, double w)

{

super(n, s, d);

weight = w;

}

public double getWeight()

{

return weight;

}

public String toString() {

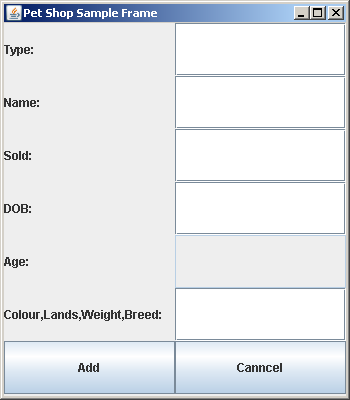
String s = super.toString() + " Weight:" + weight + "g";

return s;

}

}

## Your design for the add Pet dialog



JTextFields

Jlabels

Note: All with in a frame

JButton

## The code listing for the add Pet dialog

public void addPetView (){

for (int i = 2; i < numbtn; i++) {

contentPane.remove(buttons[i]);

}

contentPane.setLayout(new GridLayout(numlbl + (2 / 2), 2));

for (int i = 0; i < numlbl; i++) {

textFeilds[i].setText(null);

textFeilds[i].setEditable(true);

}

textFeilds[4].setEditable(false);

for (int i = 0; i < numbtn; i++) {

buttons[i].setVisible(false);

}

detaillbl[5].setText("Colour,Lands,Weight,Breed:");

this.setSize(350, 400);

buttons[0].setText("Add");

buttons[1].setText("Canncel");

buttons[0].setVisible(true);

buttons[1].setVisible(true);

}

public void addPet(){

String cat = "cat";

String hamster = "hamster";

String dog = "dog";

String parrot = "parrot";

if(cat.compareToIgnoreCase(textFeilds[0].getText()) ==0)

{

Cat c3 = new Cat(textFeilds[1].getText(),false,textFeilds[3].getText(),true);

thePetShop.addPet(c3);

JOptionPane.showMessageDialog(null, "A cat has been added.");

}

else if(hamster.compareToIgnoreCase(textFeilds[0].getText()) ==0)

{

Hamster h3 = new Hamster(textFeilds[1].getText(),false,textFeilds[3].getText(),0.0);

thePetShop.addPet(h3);

}else if(dog.compareToIgnoreCase(textFeilds[0].getText()) ==0)

{

Dog d3 = new Dog(textFeilds[1].getText(),false,textFeilds[3].getText(),textFeilds[5].getText());

thePetShop.addPet(d3);

}else if(parrot.compareToIgnoreCase(textFeilds[0].getText()) ==0)

{

Parrot p3 = new Parrot(textFeilds[1].getText(),false,textFeilds[3].getText(),textFeilds[5].getText());

thePetShop.addPet(p3);

}

restView();

}

## The additional code that you added to your application to allow Pets to be deleted

buttons[4].addActionListener(new ActionListener() {public void actionPerformed (ActionEvent e){

thePetShop.deletePet(thePetShop.currentPet().getName());

thePetShop.nextPet();

updateDetails();

}});

public boolean deletePet(String name) {

Pet p = findPet(name);

if (p != null) {

thePets.remove(p);

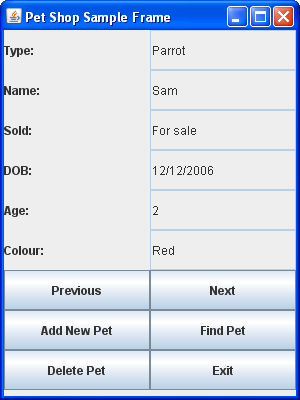
return true;

}

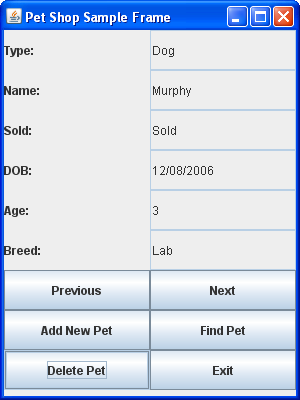
return false;

}

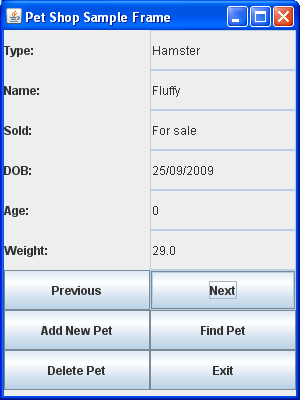
## Evidence of testing of your application



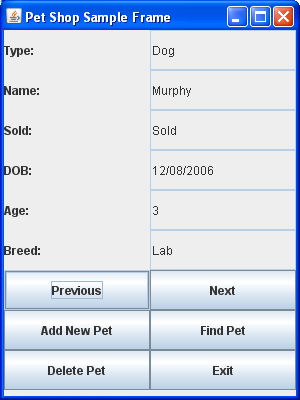
### Delete Sam Parrot



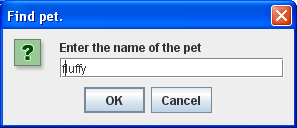
### Go to next pet

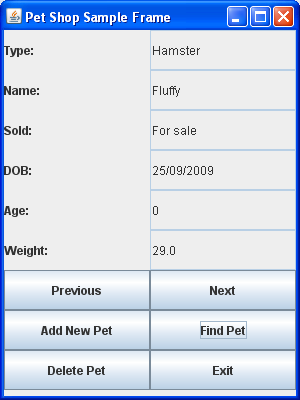


### Go to pervious pet



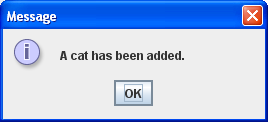
### Find Fluffy

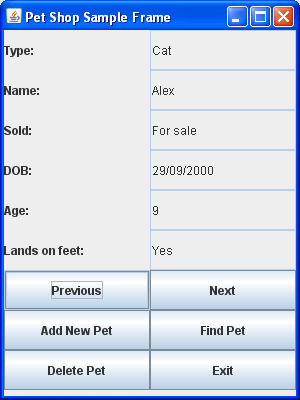




### Add Pet Alex The Cat







# Tutorial 5

## The new code that you added to the Card, Hand and Deck classes to answer parts 1, 2 and 4

### Card

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author Alex J Davison

\*/

public class Card implements Cloneable, Comparable<Card>{

private Rank rank;

private Suit suit;

public Card(Rank theRank) {

this(theRank, Suit.HEARTS);

}

public Card(Rank theRank, Suit theSuit) {

rank = theRank;

suit = theSuit;

}

public Rank getRank() {

return rank;

}

public Suit getSuit() {

return suit;

}

public String toString() {

return (rank + " of " + suit + "\n");

}

public int hashCode(){

int hash = rank.hashCode() + suit.hashCode();

return hash;

}

public boolean equals(Object obj)

{

if (rank.compareTo(((Card)obj).rank) == 1 && suit.compareTo(((Card)obj).suit) == 1)

{

return true;

}else{

return false;

}

}

public Object clone() throws CloneNotSupportedException

{

Card result = (Card)super.clone();

return result;

}

public int compareTo(Card c)

{

if (rank.getNumScore() < c.getRank().getNumScore()){

return -1;

}

else if (rank.getNumScore() > c.getRank().getNumScore())

{

return 1;

}

return 0;

}

}

### Deck

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

import java.util.Collections.\*;

/\*\*

\*

\* @author dv003874

\*/

public class Deck implements Cloneable{

private List<Card> cards = new ArrayList<Card>();

public Deck (){

for (Suit suit : Suit.values()) {

for (Rank rank : Rank.values()) {

cards.add(new Card(rank, suit));

}

}

}

public Object clone()

{

Deck deck2 = new Deck ();

try{

for (int i=0 ; i < cards.size(); i++)

{

deck2.cards.add((Card)cards.get(i).clone());

}

} catch (CloneNotSupportedException e) {System.out.println("Clone error");}

return deck2;

}

public void shuffle()

{

Collections.shuffle(cards);

}

public void sort()

{

Collections.sort(cards, new Compare());

}

public String toString()

{

String s = "";

for(Card c : cards){

s+= c.toString();

}

return s;

}

public int search(Card c){

sort();

return Collections.binarySearch(cards, c);

}

}

### Hand

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author Alex J Davison

\*/

public class Hand implements Cloneable{

// a class to represent a hand of cards

private ArrayList<Card> theCards;

public Hand() {

theCards = new ArrayList<Card>(5);

}

public void addCard(Card aCard) {

if (theCards.size() < 5) {

theCards.add(aCard);

}

}

public String toString() {

String s = "";

for (int i = 0; i < theCards.size(); ++i) {

s += "\n" + theCards.get(i);

}

return s;

}

public String getCurrentHand() {

String s = "";

for (Card theCard : theCards) {

s = s + theCard;

}

return s;

}

public int getCurrentScore() {

int s = 0;

for (Card theCard : theCards) {

s = s + theCard.getRank().getNumScore();

}

return s;

}

public Object clone() throws CloneNotSupportedException

{

Hand result = (Hand)super.clone();

return result;

}

public void sort()

{

Collections.sort(theCards);

}

}

## A code listing of the class you wrote to implement Comparator for type Card

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author dv003874

\*/

public class Compare implements Comparator<Card> {

public int compare (Card c1, Card c2){

int comp = c1.getSuit().compareTo(c2.getSuit());

if(comp != 0)

{

return comp;

}

else

{

return c1.getRank().compareTo(c2.getRank());

}

}

}

## Evidence of testing this new functionality

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author dv003874

\*/

public class TestCloneHashEquals {

public static void main(String[] args) {

Card c1 = new Card(Rank.ACE,Suit.CLUBS);

Card c2 = new Card(Rank.ACE,Suit.CLUBS);

Card c3 = null;

System.out.println(c1.hashCode());

System.out.println(c2.hashCode());

System.out.println(c1.equals(c2));

try{

c3 = (Card)c1.clone();

} catch (CloneNotSupportedException e) {System.out.println("Clone error");}

System.out.println(c1.hashCode());

System.out.println(c3.hashCode());

System.out.println(c1.equals(c3));

System.out.println(c1.toString());

System.out.println(c2.toString());

System.out.println(c3.toString());

Deck deck = new Deck();

Deck deck2;

Hand player1 = new Hand();

Hand player2 = null;

Card card1 = new Card(Rank.KING,Suit.DIAMONDS);

Card card2 = new Card(Rank.SEVEN,Suit.DIAMONDS);

Card card3 = new Card(Rank.FIVE, Suit.HEARTS);

Card card4 = new Card(Rank.TWO, Suit.SPADES);

Card card5 = new Card(Rank.JAKE,Suit.CLUBS);

player1.addCard(card1);

player1.addCard(card2);

player1.addCard(card3);

player1.addCard(card4);

player1.addCard(card5);

try{

player2 = (Hand)player1.clone();

} catch (CloneNotSupportedException e) {System.out.println("Clone error");}

System.out.println(player1.getCurrentHand());

System.out.println(player2.getCurrentHand());

System.out.println(player1.hashCode());

System.out.println(player2.hashCode());

deck2 = (Deck)deck.clone();

System.out.println(deck2);

}

}

run:

15778823

15778823

false

15778823

15778823

false

ACE of CLUBS

ACE of CLUBS

ACE of CLUBS

KING of DIAMONDS

SEVEN of DIAMONDS

FIVE of HEARTS

TWO of SPADES

JAKE of CLUBS

KING of DIAMONDS

SEVEN of DIAMONDS

FIVE of HEARTS

TWO of SPADES

JAKE of CLUBS

11077203

14576877

ACE of HEARTS

TWO of HEARTS

THREE of HEARTS

FOUR of HEARTS

FIVE of HEARTS

SIX of HEARTS

SEVEN of HEARTS

EIGHT of HEARTS

NINE of HEARTS

TEN of HEARTS

JAKE of HEARTS

QUEEN of HEARTS

KING of HEARTS

ACE of CLUBS

TWO of CLUBS

THREE of CLUBS

FOUR of CLUBS

FIVE of CLUBS

SIX of CLUBS

SEVEN of CLUBS

EIGHT of CLUBS

NINE of CLUBS

TEN of CLUBS

JAKE of CLUBS

QUEEN of CLUBS

KING of CLUBS

ACE of DIAMONDS

TWO of DIAMONDS

THREE of DIAMONDS

FOUR of DIAMONDS

FIVE of DIAMONDS

SIX of DIAMONDS

SEVEN of DIAMONDS

EIGHT of DIAMONDS

NINE of DIAMONDS

TEN of DIAMONDS

JAKE of DIAMONDS

QUEEN of DIAMONDS

KING of DIAMONDS

ACE of SPADES

TWO of SPADES

THREE of SPADES

FOUR of SPADES

FIVE of SPADES

SIX of SPADES

SEVEN of SPADES

EIGHT of SPADES

NINE of SPADES

TEN of SPADES

JAKE of SPADES

QUEEN of SPADES

KING of SPADES

ACE of HEARTS

TWO of HEARTS

THREE of HEARTS

FOUR of HEARTS

FIVE of HEARTS

SIX of HEARTS

SEVEN of HEARTS

EIGHT of HEARTS

NINE of HEARTS

TEN of HEARTS

JAKE of HEARTS

QUEEN of HEARTS

KING of HEARTS

ACE of CLUBS

TWO of CLUBS

THREE of CLUBS

FOUR of CLUBS

FIVE of CLUBS

SIX of CLUBS

SEVEN of CLUBS

EIGHT of CLUBS

NINE of CLUBS

TEN of CLUBS

JAKE of CLUBS

QUEEN of CLUBS

KING of CLUBS

ACE of DIAMONDS

TWO of DIAMONDS

THREE of DIAMONDS

FOUR of DIAMONDS

FIVE of DIAMONDS

SIX of DIAMONDS

SEVEN of DIAMONDS

EIGHT of DIAMONDS

NINE of DIAMONDS

TEN of DIAMONDS

JAKE of DIAMONDS

QUEEN of DIAMONDS

KING of DIAMONDS

ACE of SPADES

TWO of SPADES

THREE of SPADES

FOUR of SPADES

FIVE of SPADES

SIX of SPADES

SEVEN of SPADES

EIGHT of SPADES

NINE of SPADES

TEN of SPADES

JAKE of SPADES

QUEEN of SPADES

KING of SPADES

BUILD SUCCESSFUL (total time: 0 seconds)

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.util.\*;

/\*\*

\*

\* @author Alex Davison

\*/

public class TestCollections {

public static void main(String[] args) {

Deck deck = new Deck();

System.out.println(deck);

deck.shuffle();

System.out.println(deck);

deck.sort();

System.out.println(deck);

Card card1 = new Card(Rank.KING,Suit.DIAMONDS);

Card card2 = new Card(Rank.SEVEN,Suit.DIAMONDS);

Card card3 = new Card(Rank.FIVE, Suit.HEARTS);

Card card4 = new Card(Rank.TWO, Suit.SPADES);

Card card5 = new Card(Rank.JAKE,Suit.CLUBS);

Hand player1 = new Hand();

player1.addCard(card1);

player1.addCard(card2);

player1.addCard(card3);

player1.addCard(card4);

player1.addCard(card5);

player1.sort();

System.out.println(player1);

System.out.println(deck.search(card5));

}

}

run:

ACE of HEARTS

TWO of HEARTS

THREE of HEARTS

FOUR of HEARTS

FIVE of HEARTS

SIX of HEARTS

SEVEN of HEARTS

EIGHT of HEARTS

NINE of HEARTS

TEN of HEARTS

JAKE of HEARTS

QUEEN of HEARTS

KING of HEARTS

ACE of CLUBS

TWO of CLUBS

THREE of CLUBS

FOUR of CLUBS

FIVE of CLUBS

SIX of CLUBS

SEVEN of CLUBS

EIGHT of CLUBS

NINE of CLUBS

TEN of CLUBS

JAKE of CLUBS

QUEEN of CLUBS

KING of CLUBS

ACE of DIAMONDS

TWO of DIAMONDS

THREE of DIAMONDS

FOUR of DIAMONDS

FIVE of DIAMONDS

SIX of DIAMONDS

SEVEN of DIAMONDS

EIGHT of DIAMONDS

NINE of DIAMONDS

TEN of DIAMONDS

JAKE of DIAMONDS

QUEEN of DIAMONDS

KING of DIAMONDS

ACE of SPADES

TWO of SPADES

THREE of SPADES

FOUR of SPADES

FIVE of SPADES

SIX of SPADES

SEVEN of SPADES

EIGHT of SPADES

NINE of SPADES

TEN of SPADES

JAKE of SPADES

QUEEN of SPADES

KING of SPADES

ACE of CLUBS

TEN of DIAMONDS

FOUR of CLUBS

THREE of HEARTS

KING of DIAMONDS

SIX of CLUBS

EIGHT of SPADES

FOUR of SPADES

SIX of SPADES

TWO of CLUBS

SEVEN of CLUBS

SIX of DIAMONDS

TEN of HEARTS

ACE of SPADES

FIVE of SPADES

ACE of DIAMONDS

FIVE of DIAMONDS

FIVE of HEARTS

EIGHT of HEARTS

TWO of DIAMONDS

THREE of DIAMONDS

NINE of CLUBS

THREE of SPADES

KING of SPADES

TWO of SPADES

QUEEN of DIAMONDS

SEVEN of DIAMONDS

JAKE of DIAMONDS

EIGHT of CLUBS

TWO of HEARTS

TEN of SPADES

ACE of HEARTS

JAKE of CLUBS

QUEEN of CLUBS

NINE of DIAMONDS

KING of CLUBS

KING of HEARTS

TEN of CLUBS

QUEEN of HEARTS

NINE of SPADES

NINE of HEARTS

FOUR of HEARTS

FIVE of CLUBS

SEVEN of SPADES

FOUR of DIAMONDS

JAKE of SPADES

JAKE of HEARTS

QUEEN of SPADES

SEVEN of HEARTS

EIGHT of DIAMONDS

SIX of HEARTS

THREE of CLUBS

ACE of HEARTS

TWO of HEARTS

THREE of HEARTS

FOUR of HEARTS

FIVE of HEARTS

SIX of HEARTS

SEVEN of HEARTS

EIGHT of HEARTS

NINE of HEARTS

TEN of HEARTS

JAKE of HEARTS

QUEEN of HEARTS

KING of HEARTS

ACE of CLUBS

TWO of CLUBS

THREE of CLUBS

FOUR of CLUBS

FIVE of CLUBS

SIX of CLUBS

SEVEN of CLUBS

EIGHT of CLUBS

NINE of CLUBS

TEN of CLUBS

JAKE of CLUBS

QUEEN of CLUBS

KING of CLUBS

ACE of DIAMONDS

TWO of DIAMONDS

THREE of DIAMONDS

FOUR of DIAMONDS

FIVE of DIAMONDS

SIX of DIAMONDS

SEVEN of DIAMONDS

EIGHT of DIAMONDS

NINE of DIAMONDS

TEN of DIAMONDS

JAKE of DIAMONDS

QUEEN of DIAMONDS

KING of DIAMONDS

ACE of SPADES

TWO of SPADES

THREE of SPADES

FOUR of SPADES

FIVE of SPADES

SIX of SPADES

SEVEN of SPADES

EIGHT of SPADES

NINE of SPADES

TEN of SPADES

JAKE of SPADES

QUEEN of SPADES

KING of SPADES

TWO of SPADES

FIVE of HEARTS

SEVEN of DIAMONDS

KING of DIAMONDS

JAKE of CLUBS

25

BUILD SUCCESSFUL (total time: 0 seconds)

## Answers to the questions of Part 3

### What is the difference between a deep and a shallow copy?

A deep copy involves making a copy of any objects that are referred to in the starting object

In a shallow copy the data in each of the memory locations associated with an object are copied to form the new object

### Which type of copy did you use in your clone methods and why?

Deep as it creates a separate instance of the object because of the member objects be separate.

### If you override the equals method which other method should you override?

Hashcode.

### What relationship should the two methods have?

Two objects that are equal should have the same hash code however two object with the same hashcode are not necessarily equal.