1 Question 1: Poker hand

1.1 Assumtions

1. Ace (A) can either be high or low in a straight I.E. (A-2-3-4-5 or 10-J-Q-K-A)
2. Straights excludes straight flushes.

1.2 Solution

Nr of possible ways to draw 5 cards from a deck:

$$52 \cdot 51 \cdot 50 \cdot 49 \cdot 48 = 311\ 875\ 200$$

Due to order of cards are irrelevant the number of hands is divided by $5! = 120$.

$$\frac{311\ 875\ 200}{120} = 2\ 598\ 960$$

Assumption 1 yields: Nr of straights =10

(A-2-3-4-5)
(2-3-4-5-6)
(. . . .)
(10-J-Q-K-A)

Each of the five cards in each straight can be any of the four suits. This gives us a total of $10 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 10\ 240$ possible straights. Assumption 2 yields that we need to subtract the straight flushes. Thees are 40 of thees in total (9 in each color + royal straight flush). Summing it up we end up with a total combination of relevant Straights:

$$10 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 - 4 \cdot 9 - 4 = 10\ 200$$

1.3 Answer

The probably of a Straight hand in a 5 card poker hand is therefore:

$$\frac{10\ 200}{2\ 598\ 960} = 0.392\%$$

If we include Straight flush and royal straight flush:

$$\frac{10\ 240}{2\ 598\ 960} = 0.394\%$$

2 Question 2: Lottery ticket

2.1 Solution

Expected value for a Ultra win lottery ticket:

$$E[U_0] = 9 \cdot \frac{2}{10} + 18 \cdot \frac{1}{10} + (E[U_1] + E[U_2]) \cdot \frac{1}{10} + 0 \cdot \frac{6}{10}$$

Since the UWL tickets are uncorrelated and comes from the same distribution.

$$E[U_0] = E[U_1] = E[U_2] := E[U] \Rightarrow$$

$$E[U] = 9 \cdot \frac{2}{10} + 18 \cdot \frac{1}{10} + (E[U] + E[U]) \cdot \frac{1}{10} + 0 \cdot \frac{6}{10} \Rightarrow$$

$$E[U] = \frac{36}{10} + \frac{2}{10} \cdot E[U] \Rightarrow \frac{8}{10} \cdot E[U] = \frac{36}{10} \Rightarrow E[U] = \frac{36}{8} = 4\,\frac{1}{2}$$

2.2 Answer

The expected value of a Ultra Win lottery ticket is $4\,\frac{1}{2}$ Euros. This answer can be verified by a simple recursive implementation and MC simulation. For 1 000 000 simulations we get $E_{sim}[U] = 4.503951$ which is close to our analytic solution of 4.5. The code for this is found in section 5.1.

3 Question 3: Uniform draw

3.1 Assumtions

1. The function randInteger(a) returns a uniform distributed random integer in the interval $[0, a)$ (inclusive 0 and exclusive a).

3.2 Answer

```
i=0
For(Element e : ARRAY)
{
i++;
if (randInteger(i) == 0){singleElement=e};
}
return singleElement;
```

3.3 Proof of correctness of algorithm

By induction:
n=1
Clearly the algorithm works since the first element will be chosen.
n ⇒ n+1
We need to show that each element is chosen with probability $\frac{1}{n+1}$ for the $n+1$'th iteration:
The last element can easily be seen to be chosen with probability $\frac{1}{n+1}$. We therefore have a $\frac{n}{n+1}$ chance to keep the old value. Each of the n first element then also have a $\frac{n}{n+1} \cdot \frac{1}{n} = \frac{1}{n+1}$ probability to be chosen.

3.4 Time complexity

O(n), n being the number of elements in ARRAY since the randInteger(a) can be implemented to be carried out in O(1) time.

4 Question 4: Simulation exercise

4.1 Answer

Using the code found in Section 5.2 the simulation converges to $P_{win} \approx 11.7\%$. This is reached after 1 000 000 simulations and increasing the number of simulations only slightly change the results.

5 Code

5.1 Code for Ultra Win Lottery

```
import java.util.Random;

public class UWL{

    public static void main(String args[]){

     Random randnum = new Random(System.nanoTime());
     int nrSim;
     double expValue,payout,totalPayout;
     payout=0;
     totalPayout=0;
     nrSim=1000000;

     for(int i=1;i<=nrSim;i++)
     {
         payout=uwlPayout(randnum);
         totalPayout=totalPayout + payout;
     }
     expValue=(double)totalPayout/nrSim;

        System.out.println("Exp value:" + expValue);
    }

    public static double uwlPayout(Random r){
        double payment;
        int event;
        payment=0;
        event=r.nextInt(10);

        switch(event){
            case 0: payment=9;
            break;
            case 1: payment=9;
            break;
            case 2: payment=18;
            break;
            case 3: payment=uwlPayout(r)+uwlPayout(r);
            break;
        }
        return payment;
     }
}
```

5.2 Code for Simulation exercise

5.2.1 Class: Card

```
/**
 * This class represents a Card. Standard implementation.
 * @author Jakob Moberg
 * @version Created Apr 8, 2013
 */
public class Card
{
    /**int parameter for representing the rank of the card  */

    private int rank;
    /**int parameter for representing the suit of the card */
    private int suit;

    private static String[] suits= { "Spades", "Hearts", "Diamonds", "Clubs" };
    private static String[] ranks= { "A","2", "3", "4", "5", "6", "7", "8", "9", "10", "
    /**
     Constructor
     @param suit
      * int representation of suit.
      * int -> str,
      *
      * 0 -> "Spades",
      * 1 -> "Hearts",
      * 2 -> "Diamonds",
      * 3 -> "Clubs"
     @param rank
      *int representation of rank.
      * int-> str,
      *
      * 0 -> "A",
      * 1 -> "2",
      * 2 -> "3",
      * 3 -> "4",
      * 4 -> "5",
      * 5 -> "6",
      * 6 -> "7",
      * 7 -> "8",
      * 8 -> "9",
      * 9 -> "10",
      * 10 -> "J",
      * 11 -> "Q",
      * 12  -> "K"*/
    Card(int suit, int rank)
    {
        this.rank=rank;
        this.suit=suit;
    }

    /** generates a String version of the Card. The output is of the form "J of Spades".
        public @Override String toString()
    {
          return ranks[rank] + " of " + suits[suit];
    }

    /** gets the int repesentation of rank.*/
        public int getRank(){
            return rank;
        }

   /** gets the int repesentation of suit.*/
        public int getSuit(){
            return suit;
        }
}
```

5.2.2 Class: CardCollection

```
import java.util.ArrayList;
/**
 * This class an abstract class for handling a collection of Card objects.
 * @author Jakob Moberg
 * @version Created Apr 8, 2013
 */

public abstract class CardCollection{
    /**ArrayList for keeping the card objects. */
    protected ArrayList <Card> cards;

     /**Constructor. Initilizes the cards ArrayList */
    CardCollection()
    {
        cards=new ArrayList<Card>();
    }

     /**Prints the card objects in cards. Mainly used during the testing.*/
    public void printString()
    {
        System.out.println(cards.toString());
    }

    /**Returns the number of Card objects in cards*/
    public int getTotalCards()
    {
        return cards.size();
    }

      /**Returns the ArrayList cards*/
    public ArrayList <Card> getCards()
    {

        return (ArrayList <Card>) cards.clone();
    }

       /**Adds CardCollection c to the end of the current instance of the class.
         * @param c                 CardCollection that is added to the end of this.card
         */

        public void addCardCollectionToBottom(CardCollection c)
    {
        cards.addAll(c.getCards());
    }

          /**Clears the cards ArrayList.*/
    public void clearCards()
    {
        cards.clear();
    }
}
```

5.2.3 Class: Deck

```
import java.util.ArrayList;
import java.util.Collections;
```

```
/**
 * Subclass to CardCollection. This class represents the main deck
 * @author Jakob Moberg
 * @version Created Apr 8, 2013
 */
public class Deck extends CardCollection{
   /** Constructor
    * Initializes and populates the deck.
    */
    Deck()
    {
        cards=new ArrayList<Card>();
        Card tempCard;

        for(int i=0; i<=12;i++)
        {
            for(int k=0; k<=3; k++)
            {
                cards.add(new Card(k,i));
            }
        }

    }

    /** Method for shuffling the cards in deck
     */
    public void shuffleDeck()
    {
        //using fisher-yates shuffle for generating of random permutation of the ArrayLis
       Collections.shuffle(cards);
    }

    /**Method for drawing a card.
     * The drawn card is returned and removed from the cards ArrayList.
     */

      public Card drawCard()
    {
        return cards.remove(0);
    }
}
```

5.2.4 Class: Game

```
import java.util.ArrayList;

/**
 * This class represents an instace of a Game of Wristwatch solitaire.
 * @author Jakob Moberg
 * @version Created Apr 8, 2013
 */

public class Game
{
    /** parameter for keeping the deck used.*/
    private Deck deck;

    /** parameter for keeping the 13 piles of cards.*/
    private ArrayList <Pile> piles;

    /** parameter for keeping track of the active piles. Once a match is found for a pil
      * is removed from openPiles. This parameter is used to get rid the number of itera
    private ArrayList <Integer> openPiles;

    /** Constructor. Initializes a deck, shuffles it and creates the 13 piles together w
      * initially contains all integers in the interval [0,12] */
    Game()
    {
        Deck deck=new Deck();
        deck.shuffleDeck();

        piles=new ArrayList<Pile>();
        openPiles = new ArrayList<Integer>();

        for(int i=0 ; i <= 12 ;i++)
        {
         piles.add(new Pile());
         openPiles.add(i);
        }

        this.deck=deck;
        this.piles=piles;
        this.openPiles=openPiles;

    }
    /**Prints the piles of the current game. Used during testing.*/
    public void printPiles()
    {
        for(Pile p:piles)
        {
            p.printString();
        }

    }
    /** Method for running the game*/

    public int runGame()
    {
        Card activeCard;
        int nrOfCards,result,i,pos,pileSize;
        i=0;
        result=0;
        nrOfCards= this.deck.getTotalCards();

 //     Loop for drawing a card and placing in the correct pile.

        while (nrOfCards>0)
        {
           activeCard=deck.drawCard();
//        Determines the next pile to get a card.
          pileSize=openPiles.size();
            i = i % pileSize;
//        Since only Integer objects can be stored in ArrayList this conversion is neede
            pos=openPiles.get(i).intValue();

//          Checks if the drawn card matches the position.
            if(pos==activeCard.getRank())
            {
                deck.addCardCollectionToBottom(piles.get(pos));
                piles.get(pos).clearCards();
                piles.get(pos).placeCard(activeCard);

//           Closes the pile.
                openPiles.remove(i);

//          Checks if All the piles are closed and the solitaire is won.
                if(openPiles.size()==0)
                {
                    result=1;
                    break;
                }

            }
//            if no match is found the card is placed in the current pile.
            else
            {
                piles.get(pos).placeCard(activeCard);
                i++;
            }
            nrOfCards=deck.getTotalCards();
        }
        return result;
    }
}
```

5.2.5 Class: mainProgram

```
    /**
     * This class is the main program for the "Wristwatch" solitaire implementation.
     * @author Jakob Moberg
     * @version Created Apr 8, 2013
     */
public class mainProgram{
    /**
     * The main method runs a number of games prints the percentage of won games.
     * Takes either a single integer as args[] where the int represents the number
     * of MC simulation I.E("run mainProgram 100" runs 100 MC-runs).
     * The methods defaults to a single MC run.
     */
    public static void main(String args[])

    {
        int nrSimRuns;
        int totalWon;
        double percentage;


        nrSimRuns=1;

//        Checks the args[] vector. If the input is bad format or has too many inputs
//        the program breaks and error message is printed.
```

```
        if(args.length ==1){
            try{nrSimRuns=Integer.parseInt(args[0]);}

            catch(NumberFormatException ex)
            {
                System.out.println(args[0] + " is not an integer.Please try again");
                return;
            }
        }
        else if(args.length>1){
            System.out.println("This programs can only take 0 or 1 arguments");
            return;
        }
        totalWon=0;
        int result;

//      Main loop
//      Each run creates and runs a single game of Wristwatch solitaire.
        for(int i=1;i<=nrSimRuns;i++)
        {

            Game g=new Game();
            result=g.runGame();
            totalWon=totalWon + result;

        }

//     Percentage calculation and printing of results
        percentage=(double) totalWon / nrSimRuns*100;
        System.out.println("Percentage of games won " + percentage + "%");
    }
}
```

5.2.6 Class: Pile

```
/**
 * Subclass to CardCollection. This class represents the 13 piles of face-up cards in the
 * @author Jakob Moberg
 * @version Created Apr 8, 2013
 */

public class Pile extends CardCollection{

    /**Place card c first in the ArrayList cards. This represents the action of placing t
    public void placeCard(Card c)
    {
        cards.add(0,c);
    }
}
```