CS 1083

Module 4 Assignment

By Ngoc Phuong Anh Nguyen - 3712361

July 08th 2021

Part A: Comparable Lottery Tickets:

Source codes:

ComparableTicket.java:

```
/**
 * This class represents a lottery ticket, with six numbers between 1 and 49.
 * @author Ngoc Phuong Anh Nguyen - 3712361
 */

public class ComparableTicket implements Comparable<ComparableTicket>
{
    /**
     * The largest possible number (49) on any ticket.
     */
    private static final int MAX_NUMBER = 49;
    /**
     * The smallest possible number (1) on any ticket.
     */
    private static final int MIN_NUMBER = 1;
    /**
     * The quantity of numbers (6) on each ticket.
     */
    public final int NUMBER_QTY = 6;
    /**
     * Six numbers between 1 and 49 with no duplicates, in unsorted order.
     */
    private int[] number = new int[NUMBER_QTY];
```

```
/**
 * A unique identifier for the ticket.
 */
private int ticketId;

/**
 * This methods constructs a lottery ticket given a ticket id.
 * @param ticketIdIn The ID of the ticket.
 */
public ComparableTicket(int ticketIdIn)
{
    ticketId = ticketIdIn;
    chooseRandomNumbers();
    IntSort.bubbleSort(getNumbers());
}

/**
 * Accessor method for ticket id
 * @return The ticket ID.
 */
public int getTicketId()
{
    return ticketId;
}

/**
 * Accessor method that returns a reference to the numbers array.
 * @return A reference to the numbers array.
 */
public int[] getNumbers()
{
    return number;
}

/**
 *
 * @return A String that displays ticket id and the 6 numbers in the format like this example "1005: 16  6 3 31 10 26"
 */
@Override
public String toString()
{
    String temp =":";

    for(int i = 0; i<number.length; i++)
    {

        temp += Util.padLeft(number[i]+"", 3);
    }
    return(getTicketId() + temp);
}

/**
 * Generates six pseudo-random integers between 1 and 49 without duplicates, populating the numbers array with these integer
 * values. Inclusion of this method helps to simplify the code for the constructor.
```

```
     */
    private void chooseRandomNumbers()
    {
        int i = 0;

        while(i < number.length)
        {
            int rand = (int)(Math.random() * (MAX_NUMBER - MIN_NUMBER + 1)) + MIN_NUMBER;
            number[i] = rand;
            if(!duplicateNumber(i))
            {
                i++;
            }
        }
    }

    /**
     * Uses the linear search algorithm to check if numbers[i] is a duplicate of numbers[j] for all values of j that are less than i.
     * In other words, checks to see if the most recently generated number is a duplicate of any of the previously generated numbers.
     * Inclusion of this method simplifies the code for selecting random numbers for the ticket.
     * @param i The index of the most recently generated number
     * @return true if a duplicate is found, false otherwise.
     */
    private boolean duplicateNumber(int i)
    {
        boolean foundPosition = false;
        int j = 0;
        while(!foundPosition && j < number.length)
        {
            if(number[i] == number[j] && i != j)
            {
                foundPosition = true;
            }
            j++;
        }
        return foundPosition;
    }

    /**
     * Repeatedly uses the linear search algorithm to check each of this ticket's numbers against the winning
     * numbers.
     * @param winningNumbers An array of six integers representing the winning numbers for a lottery draw. These will be six numbers
     * between 1 and 49 without duplicates, in unsorted order.
     * @return The quantity of numbers on this ticket that match a winning number; this result will always be between 0 and 6.
     */
    public int countWinningNumbers(int[] winningNumbers)
    {
        int count = 0;

        for(int i = 0;i<number.length;i++)
        {
            for(int j = 0; j<winningNumbers.length; j++)
            {
```

```
            if(winningNumbers[j]==number[i])
            {
                count++;
            }
        }
    }
    return count;
}

/**
 * Compares this instance of ticket with another instance based on their respective ticket id numbers. (Provides the compareTo()
 * method required by the Comparable interface)
 * @param other The other ticket object to which this ticket object is compared.
 * @return 0 if the two ticket objects have the same ticketId,
 *        -1 if this ticketId comes before the other ticketId,
 *         1 if this ticketId comes after the other ticketId.
 */
@Override
public int compareTo(ComparableTicket other)
{
    if (other == null)
    {
        return -1;
    }
    if(this.getTicketId() == other.getTicketId())
    {
        return 0;
        }
        else if(this.getTicketId() > other.getTicketId())
        {
            return 1;
        }
        else return -1;
    }
}
```

ComparableDraw.java:

```
/**
 * This class represents a Lottery Draw.
 * @author Ngoc Phuong Anh Nguyen - 3712361
 */

public class ComparableDraw
{
    /**
     * Indicates the $ prize amount for each possible number of winning numbers (0 to 6) that a ComparableTicket can match.
     */
    private static final double[] prizeAmount = {0.0, 0.0, 10.0, 100.0, 1000.0, 100000.0, 1000000.0};

    /**
     * The current number of ComparableTicket objects in the tickets array.
     */
    private int ticketQty = 0;
```

```
/**
 * An array of ComparableTicket objects.
 */
private ComparableTicket[] tickets;

/**
 * A single ComparableTicket representing the winning numbers for this draw.
 */
private ComparableTicket winningNumbers;

/**
 * Constructs a lottery draw given the maximum quantity of tickets for this draw
 * @param maxTickets The maximum quantity of tickets for this draw.
 */
public ComparableDraw(int maxTickets)
{
    tickets = new ComparableTicket[maxTickets];
    winningNumbers = new ComparableTicket(0);
}

/**
 * Accessor method for a single ComparableTicket.
 * @param index The index from which to retrieve a ComparableTicket.
 * @return A reference to the ComparableTicket in position 'index' (or null in the case of an invalid index).
 */
public ComparableTicket getTicket(int index)
{
        if(index<tickets.length && index>=0)
        {
            return tickets[index];
        }
        else
        {
            return null;
        }
    }

    /**
     * Accessor method for an array of tickets.
     * @return a reference to the entire array of tickets.
     */
    public ComparableTicket[] getTickets()
    {
        return tickets;
    }

    public int getTicketQuantity()
    {
        return ticketQty;
    }

    /**
     * Adds a ComparableTicket to this draw
     * @param t The ComparableTicket to be added
```

```
 * @return false if the draw is full and cannot accept any more tickets, true otherwise.
 */
public boolean addTicket(ComparableTicket t)
{
    if(ticketQty<tickets.length)
    {
        tickets[ticketQty] = t;
        ticketQty++;
        return true;
    }
    else
    {
        return false;
    }
}

/**
 * Returns the prize amount won for any ticket with a given quantity of numbers that match the winning numbers
 * @param n The quantity of matching numbers
 * @return The prize amount in dollars (0.0 in the case of an invalid value for n)
 */
public double getPrizeAmount(int n)
{
    double prize = 0.0;

    if(n <= 6 && n >= 0)
    {
            prize = prizeAmount[n];
        }
        return prize;
    }

    public int[] getWinningNumbers()
    {
        return winningNumbers.getNumbers();
    }
}
```

ComparableDrawTest.java:

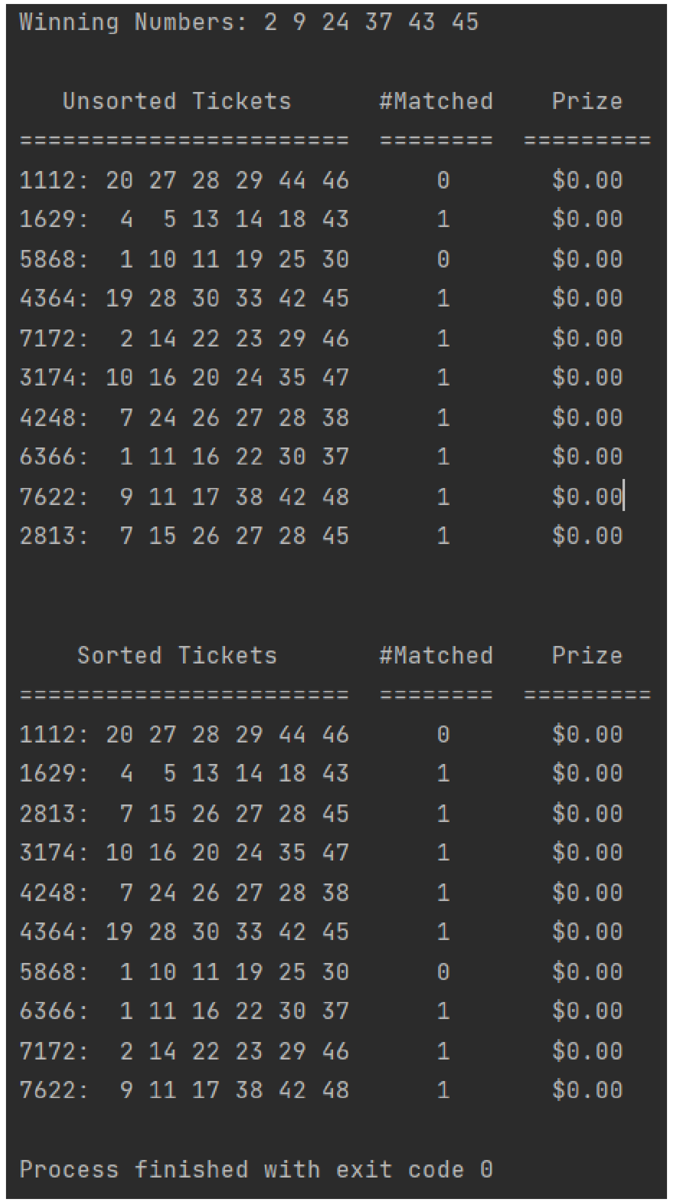
```
/**
 * This is a driver program.
 * @author Ngoc Phuong Anh Nguyen - 3712361
 */

import java.text.NumberFormat;

public class ComparableDrawTest
{
    public static void main(String[] args)
    {
        ComparableDraw comparableDraw = new ComparableDraw(20);
        NumberFormat numberFormat = NumberFormat.getCurrencyInstance();
        for(int i = 0; i < 10; i++)
        {
            ComparableTicket comparableTicket = new ComparableTicket((int) (Math.random() * (9999 - 1000 + 1) + 1000));
```

```java
    comparableDraw.addTicket(comparableTicket);
}

System.out.print("Winning Numbers: ");
for(int i = 0; i < comparableDraw.getWinningNumbers().length; i++)
{
    System.out.print(comparableDraw.getWinningNumbers()[i] + " ");
}

System.out.println("\n\n   Unsorted Tickets      "
        + "#Matched    "
        + "Prize\n=======================  ========  =========");
for (int i = 0;i<comparableDraw.getTicketQuantity();i++)
{
    System.out.println(
            comparableDraw.getTickets()[i].toString() + Util.padLeft(
                    comparableDraw.getTickets()[i].countWinningNumbers(
                            comparableDraw.getWinningNumbers()
                    ) + "", 7
            ) + Util.padLeft(
                    numberFormat.format(
                            comparableDraw.getPrizeAmount(
                                    comparableDraw.getTickets()[i].countWinningNumbers(
                                            comparableDraw.getWinningNumbers()
                                    )
                            )
                    ), 12
```

```
                )
        );
    }

    Sorter<ComparableTicket> ticketSorter = new Sorter<ComparableTicket>();
    ticketSorter.selectionSort(comparableDraw.getTickets());

    System.out.println("\n\n    Sorted Tickets       " + "#Matched    " + "Prize\n=======================  ========  =========");
    for (int i = 0;i< comparableDraw.getTicketQuantity();i++)
    {
        System.out.println(
                comparableDraw.getTickets()[i].toString() + Util.padLeft(
                        comparableDraw.getTickets()[i].countWinningNumbers(
                                comparableDraw.getWinningNumbers()
                        ) + "", 7
                ) + Util.padLeft(
                        numberFormat.format(
                                comparableDraw.getPrizeAmount(
                                        comparableDraw.getTickets()[i].countWinningNumbers(
                                                comparableDraw.getWinningNumbers()
                                        )
                                )
                        ), 12
                )
        );
    }
}
```

```
}
```

Output:



```
Winning Numbers: 2 9 24 37 43 45

   Unsorted Tickets        #Matched    Prize
=======================  ========  =========
1112: 20 27 28 29 44 46      0       $0.00
1629:  4  5 13 14 18 43      1       $0.00
5868:  1 10 11 19 25 30      0       $0.00
4364: 19 28 30 33 42 45      1       $0.00
7172:  2 14 22 23 29 46      1       $0.00
3174: 10 16 20 24 35 47      1       $0.00
4248:  7 24 26 27 28 38      1       $0.00
6366:  1 11 16 22 30 37      1       $0.00
7622:  9 11 17 38 42 48      1       $0.00
2813:  7 15 26 27 28 45      1       $0.00


    Sorted Tickets         #Matched    Prize
=======================  ========  =========
1112: 20 27 28 29 44 46      0       $0.00
1629:  4  5 13 14 18 43      1       $0.00
2813:  7 15 26 27 28 45      1       $0.00
3174: 10 16 20 24 35 47      1       $0.00
4248:  7 24 26 27 28 38      1       $0.00
4364: 19 28 30 33 42 45      1       $0.00
5868:  1 10 11 19 25 30      0       $0.00
6366:  1 11 16 22 30 37      1       $0.00
7172:  2 14 22 23 29 46      1       $0.00
7622:  9 11 17 38 42 48      1       $0.00

Process finished with exit code 0
```

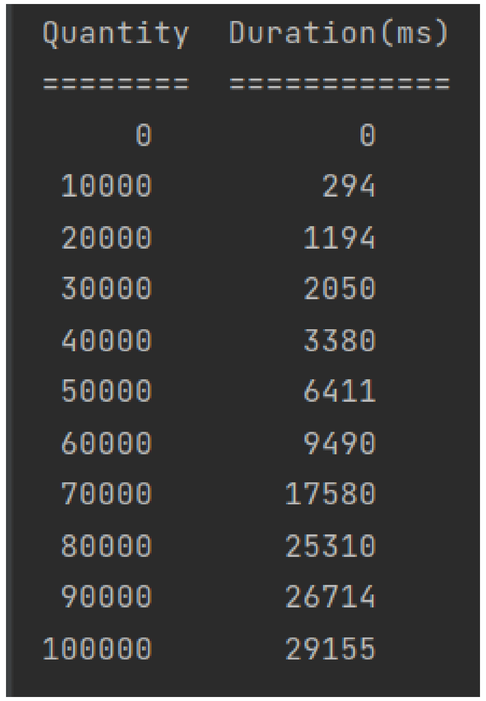
Part B: An Experiment:

Source code:

TimeTest.java:

```
/**
 * This is a driver program.
 * @author Ngoc Phuong Anh Nguyen - 3712361
 */

public class TimeTest
{
    public static void main(String[] args)
    {
        int i = 0;

        System.out.println("Quantity  Duration(ms)\n========  ============");

        while (i <= 100000)
        {
            ComparableDraw comparableDraw = new ComparableDraw(i);
            for (int j = 0; j <= i; j++)
            {
                ComparableTicket comparableTicket = new ComparableTicket((int)(Math.random() * (9999 - 1000 + 1) + 1000));
                comparableDraw.addTicket(comparableTicket);
            }

            Sorter<ComparableTicket> ticketSorter = new Sorter<ComparableTicket>();
            long before = System.currentTimeMillis();
            ticketSorter.selectionSort(comparableDraw.getTickets());
            long after = System.currentTimeMillis();

            System.out.printf("%6d%12d\n", comparableDraw.getTicketQuantity(), (after - before));
            i += 10000;
        }
    }
}
```

Output and Chart:



```
Quantity  Duration(ms)
========  ============
     0           0
 10000         294
 20000        1194
 30000        2050
 40000        3380
 50000        6411
 60000        9490
 70000       17580
 80000       25310
 90000       26714
100000       29155
```

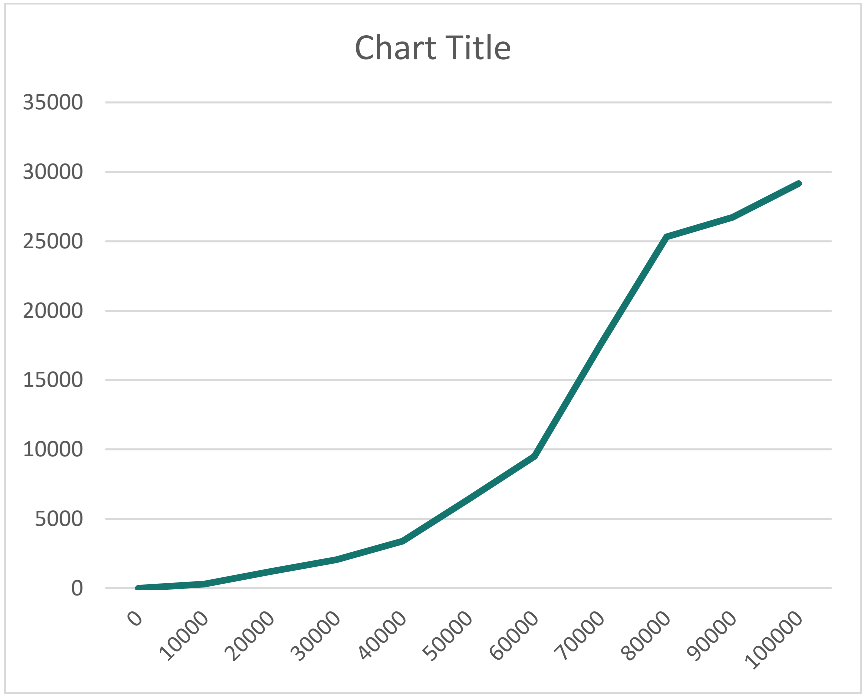


Chart Title