**Assignment 2**

Write a time-driven simulation program for a customer servicing queuing system.

A *server* is the object that provides the service. The object receiving the service is the *customer*. The *transaction time* is the time it takes to serve a customer.

In a *time-driven simulation* the clock is implemented as a counter. The passage of, say 1 minute, can be implemented by incrementing the counter by 1.

In implementing queue operations, every element in the queue is assumed to be an object of the type `DataElement`. The class `DataElement` will be used as the superclass of every class specifying the data type of the queue elements. The following is the abstract `DataElement` class:

```
public abstract class DataElement
{
    public abstract boolean equals(DataElement otherElement);
      //Method to determine whether two objects contain the same data.
      //Postcondition: Returns true if this object contains the
      //               same data as the object otherElement;
      //               otherwise, it returns false.

    public abstract int compareTo(DataElement otherElement);
      //Method to compare two objects.
      //Postcondition: Returns a value < 0 if this object is
      //                    less than the object otherElement;
      //               Returns 0 if this object is the same as
      //                    the object otherElement.
      //               Returns a value > 0 if this object is
      //                  greater than the object otherElement.

    public abstract void makeCopy(DataElement otherElement);
      //Method to copy otherElement into this object.
      //Postcondition: The data of otherElement is copied into
      //               this object.

    public abstract DataElement getCopy();
      //Method to return a copy of this object.
      //Postcondition: A copy of this object is created and
      //               a reference of the copy is returned.
}
```

The following is the `Customer` class:

```
public class Customer extends DataElement
{
    private int customerNumber;
    private int arrivalTime;
    private int waitingTime;
    private int transactionTime;

    //default constructor
    public Customer()  { ...  }

    //constructor to initialize the data members
    public Customer(int custN, int aTime, int wTime, int tTime) { ... }

    //Method to set the data members according to the parameters
    public void setCustomerInfo(int custN, int aTime,
                                int wTime, int tTime)  { ... }

    //Method to return the waiting time of a customer.
    public int getWaitingTime() { ...  }

    //Method to set the waiting time of a customer.
    public void setWaitingTime(int time)  { ... }

    //Method to increment the waiting time.
    public void incrementWaitingTime()  { ... }

    //Method to return the arrival time of a customer.
    public int getArrivalTime()  { ... }

    //Method to return the transaction time of a customer.
    public int getTransactionTime()  { ... }

    //Method to return the customer number.
    public int getCustomerNumber()  { ... }

    public boolean equals(DataElement otherElement)  { ... }

    public int compareTo(DataElement otherElement)  { return 0; }

    public void makeCopy(DataElement otherElement)  { ... }

    public DataElement getCopy()  { ... }
}
```

The following is the `Server` class:

```
public class Server
{
    private Customer currentCustomer;
    private String status;            // value is “free” or “busy”
    private int transactionTime;

    //default constructor
    public Server()  { ... }

    //Method to determine whether a server is free.
    public boolean isFree()  { ... }

    //Method to set the status of a server to "busy".
    public void setBusy()  { ... }

    //Method to set the status of a server to "free".
    public void setFree()  { ... }

    //Method to set the transaction time according to the parameter t.
    public void setTransactionTime(int t)  { ... }

    //Method to set the transaction time according to customer’s time.
    public void setTransactionTime()  { ... }

    //Method to return the remaining transaction time.
    public int getRemainingTransactionTime()  { ... }

    //Method to decrease the transaction time by 1.
    public void decreaseTransactionTime()  { ... }

    //Method to set the current customer info according to cCustomer.
    public void setCurrentCustomer(Customer cCustomer)  { ... }

    //Method to return the customer number of the current customer.
    public int getCurrentCustomerNumber()  { ... }

    //Method to return the arrival time of the current customer.
    public int getCurrentCustomerArrivalTime()  { ... }

    //Method to return the current waiting time of the current customer.
    public int getCurrentCustomerWaitingTime()  { ... }

    //Method to return the transaction time of the current customer.
    public int getCurrentCustomerTransactionTime()  { ... }
}
```

The following is the `ServerList` class:

```
public class ServerList
{
    private int numOfServers;
    private Server[] servers;

    //default constructor to initialize a list of servers
    public ServerList()  { ... }

    //constructor to initialize a list of servers specified by num.
    public ServerList(int num)  { ... }

    //Method to search the list of servers for a free server,
    //return the ID of a free server if found, else return -1.
    public int getFreeServerID()  { ... }

    //Method to return the number of busy servers.
    public int getNumberOfBusyServers()  { ... }

    //Method to set a server to "busy".
    //Postcondition: To serve the customer specified by
    //    cCustomer the server specified by serverID is set
    //    to "busy", and the transaction time is set according
    //    to the parameter tTime.
    public void setServerBusy(int serverID,  Customer cCustomer,
                              int tTime)  { ... }

    //Method to set a server to busy.
    //Postcondition: To serve the customer specified by
    //    cCustomer, the server specified by serverID is set
    //    to "busy", and the transaction time is set according
    //    to the customer's transaction time.
    public void setServerBusy(int serverID, Customer cCustomer)  { ... }

    //Method to update the transaction time of each busy server.
    //Postcondition: The transaction time of each busy server
    //    is decremented by one time unit. If the transaction
    //    time of a busy server is reduced to zero, the
    //    server is set to "free" and a message indicating which
    //    customer was served, together with the customer's
    //    departing time, is printed on the screen.
    public void  updateServers()
    {
      ...
      Output the following message to the screen:
      "Server number XXX, Customer number YYY departed clock unit ZZZ"
    }
}
```

In the assignment, a queue will be implemented by a circular array. The implementation should check for error conditions, for example, attempt to access an element when the queue is empty, or attempt to add an element when the queue is full, etc. An appropriate error message should be printed to the screen when error conditions are detected. The following is the `CirArrayQueue` class:

```
public class CirArrayQueue
{
    private int maxQueueSize;
    private int count;           // number of elements in the queue
    private int queueFront;
    private int queueRear;
    private DataElement[] list;  //Array of references to the
                                 //objects that store queue elements

    //default constructor, creates a queue of default size 100
    public CirArrayQueue()  { ... }

    //constructor with a parameter
    public CirArrayQueue(int queueSize)  { ... }

    //copy constructor
    public CirArrayQueue(CirArrayQueue otherQueue)  { ... }

    //Method to initialize the queue to an empty state.
    public void initializeQueue()  { ...  }

    //Method to determine whether the queue is empty.
    public boolean isEmpty()  { ... }

    //Method to determine whether the queue is full.
    public boolean isFull()  { ... }

    //Method to return the first element of the queue.
    public DataElement peekFront()  { ... }

    //Method to return the last element of the queue.
    public DataElement peekRear()  { ... }

    //Method to add queueElement to the rear of the queue.
    public void enqueue(DataElement queueElement)  { ... }

    //Method to remove the first element of the queue.
    public void dequeue()  { ... }

    //Method to make a copy of otherQueue.
    public void copyQueue(CirArrayQueue otherQueue)  { ... }
}
```

The following is the `WaitingCustomerQueue` class:

```
public class WaitingCustomerQueue extends CirArrayQueue
{
    //default constructor
    public WaitingCustomerQueue()  { ... }

    //constructor with parameter queue size
    public WaitingCustomerQueue(int size)  { ...}

    //copy constructor
    public WaitingCustomerQueue(WaitingCustomerQueue otherQ)  { ... }

    //Method to increment the waiting time of each
    //customer in the queue by one time unit.
    //Postcondition: The waiting time of each customer in
    //               the queue is incremented by one time unit.
    public void updateWaitingQueue()  { ... }
}
```

The main program of the simulation would look like the following:

```
public class CustomerServicingSimulation
{
    private static int simulationTime;
    private static int numberOfServers;
    private static int transactionTime;
    private static int timeBetweenCustomerArrival;

    public static void main(String[] args) throws IOException
    {
         setSimulationParameters();
         runSimulation();
    }

    public static void setSimulationParameters()
    {
          Read in and store the simulation time.
          Read in and store the number of servers.
          Read in and store the transaction time.
          Read in and store the time between customer arrivals.
    }

    public static boolean isCustomerArrived(double arvTimeDiff)
    {
        double value;
        value = (double) (Math.random());
        return (value > Math.exp(- 1.0/arvTimeDiff));
    }
```

```
public static void runSimulation()
{
    Declare and initialize the variables such as the simulation parameters, customer number, clock,
    total and average waiting times, number of customers arrived, number of customers served,
    number of customers left in the waiting queue, number of customers left with the servers, the
    waiting queue, a list of serves

    for(clock = 1; clock <= simulationTime; clock++)
    {
        Update the server list to decrement the transaction time of each busy server by one time unit.

        If the customer's queue is nonempty, increment the waiting time of each customer by one
        time unit.

        if(isCustomerArrived(timeBetweenCustomerArrival))
        {
            A customer just arrives, increment the number of customers by 1 and add the new
            customer to the queue.

            Output the following message to the screen:
            "Customer number XXX arrived at time unit YYY"
        }

        If a server is free and the customers' queue is nonempty then remove a customer from the
        front of the queue and send the customer to the free server.
    }

    Print the following summary results of the simulation to the screen:
    "Simulation ran for XXX time units"
    "Number of servers: XXX"
    "Average transaction time: XXX"
    "Average arrival time difference between customers: XXX"
    "Total wait time of all customers: XXX"
    "Number of customers who completed a transaction: XXX"
    "Number of customers left in the servers: XXX"
    "Number of customers left in the queue: XXX"
    "Average wait time: XXX"
    "************** END SIMULATION *************"
}
}
```

Let t = total simulation time, s = number of servers,
r = average transaction time, a = average time between customers arrivals.
Run the test program with different parameters.

(a) t=100, s=1, r=5, a=4.
(b) t=100, s=2, r=5, a=4.
(c) t=1000, s=1, r=5, a=4.
(d) t=1000, s=3, r=5, a=4.

For test cases (c) and (d), do not print the two intermediate output messages:

i. `"Customer number XXX arrived at time unit YYY"`
(from method `updateServers` of `ServerList` class)

ii. `"Server number XXX, Customer number YYY departed clock unit ZZZ"`
(from method `runSimulation` of the main program `CustomerServicingSimulation`)

A sample of one simulation run might look like the following:

```
Enter the simulation time: 100
Enter the number of servers: 2
Enter the transaction time: 5
Enter the time between customer arrivals: 4

Customer number 1 arrived at time unit 3
Customer number 2 arrived at time unit 5
Customer number 3 arrived at time unit 6
Server No: 1 Customer number 1 departed at clock unit 8
Server No: 2 Customer number 2 departed at clock unit 10
Customer number 4 arrived at time unit 12
Server No: 1 Customer number 3 departed at clock unit 13
Customer number 5 arrived at time unit 13
Customer number 6 arrived at time unit 16
Server No: 2 Customer number 4 departed at clock unit 17
Server No: 1 Customer number 5 departed at clock unit 18
Server No: 2 Customer number 6 departed at clock unit 22
Customer number 7 arrived at time unit 23
Customer number 8 arrived at time unit 24
Customer number 9 arrived at time unit 25
Server No: 1 Customer number 7 departed at clock unit 28
Server No: 2 Customer number 8 departed at clock unit 29
Customer number 10 arrived at time unit 30
Customer number 11 arrived at time unit 32
Server No: 1 Customer number 9 departed at clock unit 33
Server No: 2 Customer number 10 departed at clock unit 35
Server No: 1 Customer number 11 departed at clock unit 38
Customer number 12 arrived at time unit 39
Server No: 1 Customer number 12 departed at clock unit 44
Customer number 13 arrived at time unit 48
Customer number 14 arrived at time unit 50
Customer number 15 arrived at time unit 51
Server No: 1 Customer number 13 departed at clock unit 53
Customer number 16 arrived at time unit 53
Server No: 2 Customer number 14 departed at clock unit 55
Server No: 1 Customer number 15 departed at clock unit 58
Customer number 17 arrived at time unit 59
Server No: 2 Customer number 16 departed at clock unit 60
Server No: 1 Customer number 17 departed at clock unit 64
Customer number 18 arrived at time unit 68
Customer number 19 arrived at time unit 69
Server No: 1 Customer number 18 departed at clock unit 73
Server No: 2 Customer number 19 departed at clock unit 74
Customer number 20 arrived at time unit 85
Customer number 21 arrived at time unit 89
Server No: 1 Customer number 20 departed at clock unit 90
Server No: 2 Customer number 21 departed at clock unit 94
Customer number 22 arrived at time unit 95
Customer number 23 arrived at time unit 97
Server No: 1 Customer number 22 departed at clock unit 100

Simulation ran for 100 time units
Number of servers: 2
Average transaction time: 5
Average arrival time difference between customers: 4
Total wait time: 11
Number of customers who completed a transaction: 22
Number of customers left in the servers: 1
Number of customers left in the queue: 0
Average wait time: 0.48
************** END SIMULATION *************
```