02- QUEUE

Objective: At the end of this topic, students should be able to:

1. Understand the concept of Queue
2. Design and write programs for solving problems by using Queue data structure.

Section A: Service Simulation
Section B: Queue Data Structure
Section C:Worked Example - ABC Wash Machine
Section D: DEF Wash Machine (Extension of ABC Wash Machine)

Section A
Service Simulation

In this topic, we will learn about queue through some problems by using service simulation. Service simulation is used to imitate the real operation system and gather statistic of certain information. Statistical report that can be generated by simulation are *arrival rates and patterns*, *waiting and service times* and *percentage of time* the automated equipment is utilized. Queue in service simulation is the replication of the real queue. Thus, it will retain the queue concept: *first item in queue will be served prior to the next item.*

Beside queue concept, other information that you need to identify regarding service simulation is its complexity. These complexities depend on (1) the number of servers (example: counter) and (2) the number of services provided.

Complexity	Simulation Characteristic	Example
1	One server with one service. One queue with many customers.	Queue at Car wash machine.
2	One server with many services. One queue with many customers.	Automated teller machine (ATM)
3	Many servers with one services. One queue with many customers.	*Can you think of one example?*
4	Many servers with many services. One queue with many customers.	*Can you think of one example?*
5	Many servers with many services. Many queue with many customers.	*Can you think of one example?*

Since the real situation often involves many customers and services, it would be easier if the simulation process can be assisted by computer. Such advantages of using computer to do simulation are:

1st : The information would be gathered without involving real customers.
2nd : A simulation by computer can be faster than the actual implementation because of the speed of the computer.
3rd : The simulation could be easily replicated.

In real service system situation, most of the data for example customer arrival time and type of service required are different among the customers. Hence, random data are needed in order to carry out simulation. To get the random value, we can either:

(1) use standard input (keyboard or file feeding) or file stream.
(2) randomly generated value using the rand() function.

Please refer to figure 1.

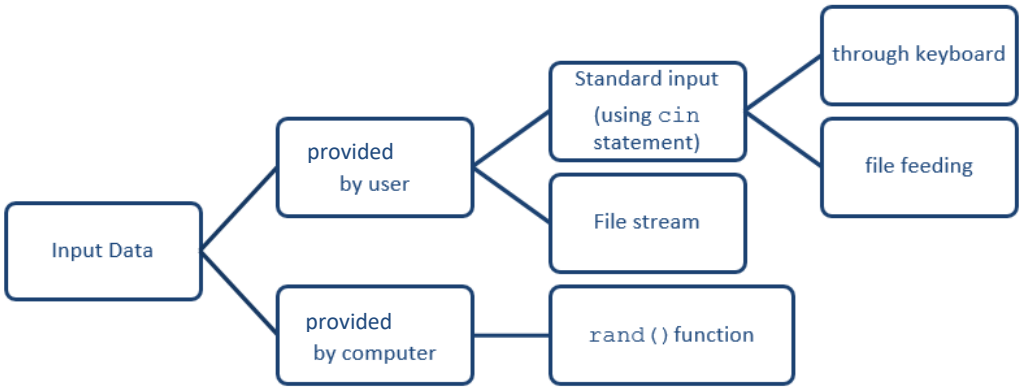


Figure 1: Approach to get random data.

For the simulation problem discussed in this topic, we will use rand() function to generate the random value (refer to section: **About rand() function)**.

Usually simulation program will output a statistical information that involves *time-based data* for instance *arrival_rates*, *longest_waiting_time* and *average waiting_time*. To hold and process the time-based data, the simulation program should use a time data type that comprises of 3 attributes: hours, minutes and seconds. Since it it is not support by Java standard library, we need to define a new class for handling time (refer to section: **About class *clockType***).

My Note:

About *rand()* function

rand is a class defined in `java.util.Random;`. It is used to generate random number.

For using this class to generate random numbers, we have to first create an instance of this class and then invoke methods such as nextInt(), nextDouble(), nextLong() etc using that instance.

Rand class can be used to generate random numbers of types integers, float, double, long, boolean.

We can pass arguments to the methods for placing an upper bound on the range of the numbers to be generated.

For example, nextInt(6) will generate numbers in the range 0 to 5 both inclusive.

Below is an example on how to use *random class*.

```
import java.util.*;

public class simulation {
	public static void main(String[] args) {

	    Random rand = new Random();

	    int n = rand.nextInt(10); // Obtain a number between [0 - 9].
	    System.out.print(" " + n);
	}
}
```

My Note:

Self-study:

1. Write Java statements to obtain a random value from range 1 to 2 using rand function.
2. Write Java statements to obtain and print 10 random value from range 0 to 9 using rand function.
3. Write Java statements to obtain and print 5 random values of below data using random function:
    a. **Department code** from range 1 to 4
    b. **Subject code** from range 0 to 10

About class *Clock*

The class *clock* is a **user defined class**. The class *Clock* hold variables and functions to support and process instructions of time data type.

The declaration of class *Clock* comprises of 3 data members of type integer: hr, min and sec, and 11 member functions with public access: *setTime, getTime, printTime, incrementSeconds, incrementMinutes, incrementHours, equalTime, addTimeMinute, lessThan, earlier, durationSec*. This declaration can also be retrieved from link http://www.cs.sfu.ca/CourseCentral/101/jregan/SourceCode/Ch8/Clock%20method%20%20toString/Clock.java.

```
public class Clock
{
  private int hr;  //store hours
  private int min; //store minutes
  private int sec; //store seconds

//Constructor with parameters, to set the time
//The time is set according to the parameters
//Postcondition: hr = hours; min = minutes; sec = seconds
  public Clock(int hours, int minutes, int seconds)
  {
              setTime(hours, minutes, seconds);
  }

//Default constructor
//Postcondition: hr = 0; min = 0; sec = 0
  public Clock()
  {
      setTime(0, 0, 0);
  }

//Method to set the time
//The time is set according to the parameters
//Postconditions: hr = hours; min = minutes; sec = seconds
  public void setTime(int hours, int minutes, int seconds)
  {
              if(0 <= hours && hours < 24)
                      hr = hours;
              else
                      hr = 0;

      if(0 <= minutes && minutes < 60)
                      min = minutes;
              else
                      min = 0;

              if(0 <= seconds && seconds < 60)
                      sec = seconds;
              else
                 sec = 0;
  }

 public void getTime (int hours, int minutes, int seconds) {
               hours = hr;
               minutes = min;
               seconds = sec;
       }


//Method to return the hours
//Postconditions: the value of hr is returned
```

```
  public int getHours()
  {
               return hr;
  }

//Method to return the minutes
//Postconditions: the value of min is returned
  public int getMinutes()
  {
               return min;
  }

//Method to return the seconds
//Postconditions: the value of sec is returned
  public int getSeconds()
  {
               return sec;
  }

//Method to print the time
//Postconditions: Time is printed in the form hh:mm:ss
  public void printTime()
  {
               if(hr < 10)
                  System.out.print("0");
            System.out.print(hr + ":");

               if(min < 10)
                  System.out.print("0");
               System.out.print(min + ":");

               if(sec < 10)
                  System.out.print("0");
               System.out.print(sec);
  }

//Method to increment the time by one second
//Postconditions: The time is incremented by one second
//If the before-increment time is 23:59:59, the time
//is reset to 00:00:00
  public void incrementSeconds()
  {
        sec++;
               if(sec > 59)
               {
                  sec = 0;
                  incrementMinutes(); //increment minutes
               }
  }

//Function to increment the time by one minute
//Postconditions: The time is incremented by one minute
//If the before-increment time is 23:59:53, the time
//is reset to 00:00:53
  public void incrementMinutes()
  {
               min++;
               if(min > 59)
               {
                  min = 0;
                  incrementHours(); //increment hours
               }
  }
```

```
  public void addTimeMinute(int minutes) { //new
              for (int i=0; i<minutes; i++)
                     incrementMinutes();
       }

//Method to increment the time by one hour
//Postconditions: The time is incremented by one hour
//If the before-increment time is 23:45:53, the time
//is reset to 00:45:53
  public void incrementHours()
  {
             hr++;
             if(hr > 23)
                hr = 0;
  }

//Function to compare the two times
//Postconditions: Returns true if this time is equal to
//                otherTime; otherwise returns false
  public boolean equals(Clock otherClock)
         {
      return(hr == otherClock.hr
                    && min == otherClock.min
             && sec == otherClock.sec);
  }

//Method to copy time
//Postcondition: The data members of otherTime are copied
//               into the correspoding data members of
//               this time.
//               hr = otherTime.hr; min = otherTime.min;
//               sec = otherTime.sec;
  public void makeCopy(Clock otherClock)
         {
      hr = otherClock.hr;
      min = otherClock.min;
      sec = otherClock.sec;
  }

//Method to return a copy of time
//Postcondition: A copy of the object is created
//               and a reference of the copy is returned
  public Clock getCopy()
  {
             Clock temp = new Clock();

             temp.hr = hr;
             temp.min = min;
             temp.sec = sec;

             return temp;
  }

  public String toString()
  {
      String str = "";

      if(hr < 10)
                str = "0";
      str = str + hr + ":";

      if(min < 10)
```

```
                str = str + "0" ;
        str = str + min + ":";

        if(sec < 10)
                str = str + "0";
        str = str + sec;

        return str;
    }

    public int durationSec (Clock stopA) { //new
                int durSec, durMin, durHour;
                boolean borrowHour=false, borrowMin=false;
                Clock stop;

                stop=stopA;
                if (stop.sec >= sec)  durSec=stop.sec-sec;
                else {
                        durSec=((stop.sec+60) - sec);
                        borrowMin=true;
                }

                if (borrowMin)
                        stop.min-=1;
                if (stop.min >= min)  durMin=(stop.min-min);
                else {
                        durMin=((stop.min+60) - min);
                        borrowHour=true;
                }

                if (borrowHour)
                        stop.hr-=1;
                durHour=(stop.hr - hr);

                return (durHour*60)+durMin+(int)(durSec/60.0);
        }

     public boolean equalTime(Clock otherClock) {
                return (hr == otherClock.hr && min == otherClock.min && sec ==
otherClock.sec);
        }

public boolean lessThan (Clock otherClock) { //new
                if (hr < otherClock.hr) return true;
                else if (hr == otherClock.hr) {
                        if (min < otherClock.min) return true;
                        else if (min == otherClock.min) {
                                if (sec == otherClock.sec) return true;
                        }
                }
                return false;
        }
}
```

The class C*lock* have been prepared for you to use in your program. We will not discuss the detail of class Clock in this note, as you already learned about it in OOP topics before this. Please try and play around with the class Clock.

Example of code to declare an object of class Clock:

```
Clock clock1 = new Clock();
Clock clock2 = new Clock(2,10,40);
Clock washEnd = new Clock();
```

Once we have declared the respective object of class Clock, we can use it. Below are the examples:

```
clock1.incrementHours();
clock2.addTimeMinute(10);
clock1.printTime();
washEnd.addTimeMinute(5);    // to access the addTimeMinute function for
                             // object washEnd
int k=i.durationSec(washEnd); // to access the durationSec from object i to washEnd
```

My Note:

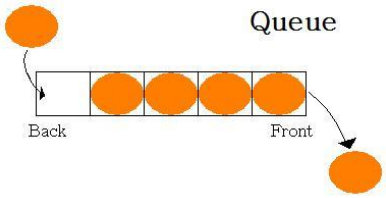
Self-Activity:

1. Based on the declaration of class *Clock*, answer below questions:
    a. List the data members for class *Clock*.
    b. How many function members of class *Clock*?
    c. Write Java codes that do the following:
        i. declare two variables *myClock* and *yourClock* of type Clock.
        ii. set *myClock* to 8:30 and *yourClock* to 10:40.
        iii. increase *myClock* to 20 minutes.
        iv. display the time of *myClock* and *yourClock*.
        v. display the time difference between *myClock* and *yourClock*.
        vi. check whether *myClock* equals, earlier or later than *yourClock*.

About QUEUE
Using Queue Interface from java.util library

The Queue interface is available in ***java.util package*** and extends the Collection interface. Being an interface the queue needs a concrete class for the declaration, in this topic we will use class LinkedList in Java.

Concept: First in first out.



How to declare a queue?

```
Queue<Integer> intQueue = new LinkedList<>(); //intQueue is a queue of integer.
Queue<String> strQueue = new LinkedList<>();  //strQueue is a queue of string.
```

What are the common methods used in this problem?

Push: to put data onto queue.

```
intQueue.push(4);
strQueue.push("Hello");
```

Pop: to remove data from queue.

```
intQueue.pop();
strQueue.pop();
```

front: to retrieve the first data in queue.

```
int a; string b;
a= intQueue.front();
b= strQueue.front();
```

Empty: to check whether the queue is empty.

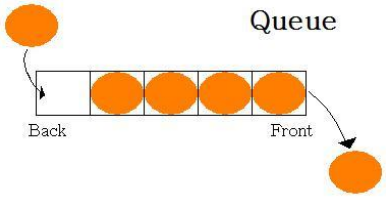
```
if (intQueue.empty())
     cout << "no more data in intQueue";
```

About QUEUE
Using Queue Class / Queue API

The class Q**ueue** implement the same concept queue as queue in java library. The class *Queue* is a **user defined class**. It holds variables and functions to support the concept of Queue.

The declaration of Queue class comprises of 3 data members, two of them of type Node: first and last and the int data type with a number of member functions such as `enqueuer, dequeuer, peek, size, isEmpty`. The full version of Queue class can also be obtained from UKMFolio.

Concept: First in first out.



How to declare a queue using Queue API?

How to declare a queue?

```
Queue <Integer> q1 = new Queue<>();      //q1 is a queue of type integer.
Queue <Clock> waitQueue= new Queue <>(); //waitQueue is a queue of Clock.
```

What are the common methods used in this problem?

Enqueue: to put data onto queue.

```
q1.enqueue(10);
waitQueue.enqueue(i.getCopy());
```

dequeue: to remove data from queue.

```
del = q1.dequeue();
del = strQueue.dequeue();
```

peek: to retrieve the first data in queue.

```
int a; Clock b;
a= q1.peek();
b= waitQueue.peek();
```

isEmpty: to check whether the queue is empty.

```
if (q1.isEmpty())
     System.out.print("no more data in intQueue");
```

Class Queue (Queue.java)

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class Queue<Item> implements Iterable<Item> {
    private int n;          // number of elements on queue
    private Node first;     // beginning of queue
    private Node last;      // end of queue
```

```
    // helper linked list class
    private class Node {
        private Item item;
        private Node next;
    }

    /**
     * Initializes an empty queue.
     */
    public Queue() {
        first = null;
        last  = null;
        n = 0;
    }


    public boolean isEmpty() {
        return first == null;
    }


    public int size() {
        return n;
    }


    public int length() {
        return n;
    }


    public Item peek() {
        if (isEmpty())
               throw new NoSuchElementException("Queue underflow");
        return first.item;
    }


    public void enqueue(Item item) {
        Node oldlast = last;
        last = new Node();
        last.item = item;
        last.next = null;
        if (isEmpty()) first = last;
        else           oldlast.next = last;
        n++;
    }

    public Item dequeue() {
        if (isEmpty()) throw new NoSuchElementException("Queue underflow");
        Item item = first.item;
        first = first.next;
        n--;
        if (isEmpty()) last = null;   // to avoid loitering
        return item;
    }


    public String toString() {
        StringBuilder s = new StringBuilder();
        for (Item item : this) {
            s.append(item);
            s.append(' ');
```

```
        }
        return s.toString();
    }

    public Iterator<Item> iterator()  {
        return new ListIterator();
    }

    // an iterator, doesn't implement remove() since it's optional
    private class ListIterator implements Iterator<Item> {
        private Node current = first;

        public boolean hasNext()  { return current != null;                     }
        public void remove()      { throw new UnsupportedOperationException();  }

        public Item next() {
            if (!hasNext()) throw new NoSuchElementException();
            Item item = current.item;
            current = current.next;
            return item;
        }
    }
}
```

Tutorial Activity:

1. Explained the concept of Queue.
2. List out any 3 methods class Queue from Queue API and queue from java library. Define its purpose.
3. List the requirements to use Queue from Queue API and queue from java library.

Hands-on Activity

1. Mission: Create a program that receives sequence of integers that ends with 0, whenever the integer is odd push them onto an *oddQueue,* otherwise push onto *evenQueue*. Then pop and display the elements of *oddQueue*, *evenQueue* and its size respectively. Display your output in the following format:

   <name of queue> <size of queue>: <elements of the queue>

   Sample IO:

Input	Output
2 34 1 8 5 22 0 10 7 16 -2 0	oddQueue 2: 1 5 evenQueue 3: 34 8 22 oddQueue 1: 7 evenQueue 3: 10 16 -2

2. ***Palindrome*** is a sequence of characters, a word, phrase, number or sequence of words which reads the same backward as forward. Example of palindrome words are katak, civic and anna. Write a program that will read the word and identify whether it is a palindrome or not. In this task, it is compulsory to use Stack and Queue data structure.

Sample I/O:

Input: racecar
Ouput: racecar not a palindrome

Input: p(;(p
Ouput: p(;(p is a Palindrome

Worked-Example 1: ABC Wash Machine

ABC WASH MACHINE	
Input	Randomly generated
Output	Standard Output
Java Elements	Selection, Looping
Data Structure	Queue
Additional Function	class Clock, rand()

Problem Description

ABC Wash Machine provides a self-wash car for its customer. The customer must queue up in order to get the service. The customer will immediately be served if the queue is empty. If the machine is free, then it will serve the front customer in the queue. The machine takes 5 minutes to complete one service. The machine operation starts at 8.00 am and ends at 8.30 am. However, it will continue serve customer who arrive before or by 8.30 am.

The owner of the ABC Wash Machine would like to know some statistical information so that he can improve the machine in the future. The information are:

- Number of customer that arrives by 8.30 am.
- Longest customer waiting time.
- Average customer waiting time.

Assume that all customers are an ethical person and determined to get the service, also the machine is ideal. Input for customer arrival times will randomly generated by rand() function. Assume that the next customer is likely to arrive within 9 minutes after the current customer.

Your task is to write a program that will print out the above statistical information.

Input

Randomly generated.

Sample Output

Number of customer that arrives by 8.30 am: <Number_ customer _arrives_by 8.30>
Longest customer waiting time: < Longest_waiting_time>
Average customer waiting time: < Average_waiting_time>

Solution

The algorithm for ABC Wash Machine simulation:

Set the operation time.
*Generate all car arrivals, and push them onto **arrival queue**.*
Set the initial state for the wash machine (status, startWash, endWash)
Set current event.

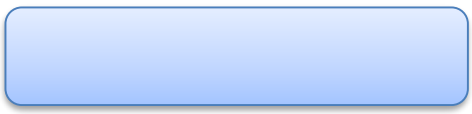
While there is still an event

If car arrival event occurs:
Push car onto waiting queue.
If machine finish washing
Update machine state.
If machine available and there is a car waiting:
Front car in waiting queue will be washed.
Remove this car from waiting queue.
Update the machine state and statistical information.
If machine available and there is no car waiting:
Update endWash to beyond operation time.
Jump to the next earliest event that is either car arrival or machine finish washing.

Produce report.

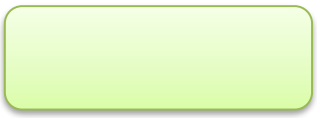
Basic structure

```
public static void main(String[] args) {
            Clock carArrival;
            Queue <Clock> waitQueue= new Queue <>();
            Queue <Clock> arrivalQueue = new Queue<>();

            //enum status {free, busy} machineWash;
            status machineWash;

            //observation data
            Clock totalWaitTime= new Clock();
            Clock maxWaitTime = new Clock();
            Clock totalServiceTime = new Clock();
            Clock startTime= new Clock();
            Clock endTime= new Clock();
            Clock washEnd = new Clock();
            Clock washStart = new Clock();

            Clock i;
            int nextArrival=0;
            Random rand = new Random();
```



```
for(i=startTime;(i.lessThan(endTime)||(!waitQueue.isEmpty())||(!arrivalQueue.isEmpty())); )
   {
```

```
   }
```



}

Full structure of the worked-example program: ABC Wash Machine Simulation

```
1   import java.util.*;
2
3   public class ABCWashMachine {
4      public enum status { free, busy}
5      static int maxDur=0;
6      static int numCar=0;
7      static int totalWait=0;
8      static int totalWork=0;
9
10     public static void main(String[] args) {
11          Clock carArrival;
12          Queue <Clock> waitQueue= new Queue <>();
13          Queue <Clock> arrivalQueue = new Queue<>();
14
15          //enum status {free, busy} machineWash;
16          status machineWash;
17
18          //observation data
19          Clock totalWaitTime= new Clock();
20          Clock maxWaitTime = new Clock();
21          Clock totalServiceTime = new Clock();
22          Clock startTime= new Clock();
23          Clock endTime= new Clock();
24          Clock washEnd = new Clock();
25          Clock washStart = new Clock();
26
27          Clock i;
28          int nextArrival=0;
29          Random rand = new Random();
30
31          startTime.setTime(8,0,0);
32          endTime.setTime(8,30,0); // can change to 12 pm
33
34          for (i=startTime.getCopy();i.lessThan(endTime); ){
35                 nextArrival = rand.nextInt(10); nextArrival in the range 0 to 9
36                 i.addTimeMinute(nextArrival);
37                 if(i.lessThan(endTime)) {
38                    arrivalQueue.enqueue(i.getCopy());
39                    System.out.println("car arrival: " + i.toString() + " < " +
40                                              endTime.toString());
41                  }
42          }
43
44         //start the simulation
45          machineWash=status.free;
46          if (!arrivalQueue.isEmpty()) {
47                 startTime=arrivalQueue.peek();
48                 washEnd=startTime.getCopy();
49                 washEnd.addTimeMinute(5);
50          } else
51                 startTime=endTime.getCopy();
52
```

```
53          Clock del;
54          for
55          (i=startTime;(i.lessThan(endTime)||(!waitQueue.isEmpty())||(!arrivalQueue.isEmpty()));)
56          {
57                  if (!arrivalQueue.isEmpty())
58                          if (i.equalTime(arrivalQueue.peek())) {
59                                  waitQueue.enqueue(i.getCopy());
60                                  del=arrivalQueue.dequeue();
61                          }
62
63                          if ((machineWash==status.busy) && (i.equalTime(washEnd))) {
64                                  washEnd.setTime(14,0,0);
65                                  machineWash=status.free;
66                          }
67
68                          if ((machineWash==status.free) && !(waitQueue.isEmpty())) {
69                                  washStart=i.getCopy();
70                                  washEnd=i.getCopy(); washEnd.addTimeMinute(5);
71                                  doAnalysis(i,waitQueue.peek(),washEnd); // call doAnalysis method
72                                  del=waitQueue.dequeue();
73                                  machineWash=status.busy;
74                          }
75
76                          if ((machineWash==status.free) && (waitQueue.isEmpty()))
77                                  washEnd.setTime(14,0,0);
78
79                          //jump to next event.
80                          if (!arrivalQueue.isEmpty())
81                                  if (washEnd.lessThan(arrivalQueue.peek())) {
82                                          i=washEnd.getCopy();
83                                  }
84                                  else {
85                                          i=arrivalQueue.peek().getCopy();
86                                  }
87                          else  {
88                                  i=washEnd.getCopy();
89                          }
90          }
91
92          //report
93          System.out.print("\nREPORT\n");
94          System.out.print("Number of customer arrive by 8.30 am: " + numCar + "\n");
95          System.out.print("Longest waiting time: " + maxDur + " minutes\n");
96          System.out.print(String.format("Average waiting time: %.2f minutes",
97                          totalWait/(float)numCar));
98    }
99
100
101   public static void doAnalysis(Clock waitStop, Clock start, Clock washStop) {
102
103       int carWait,machineWork;
104
105       carWait=start.durationSec(waitStop.getCopy());
106       machineWork=waitStop.durationSec(washStop.getCopy());
107       numCar++;
108       totalWait+= carWait;
109       totalWork+=machineWork;
110       if (maxDur<carWait)
111            maxDur=carWait;
112    }
113 }
114
```

Self-activity

1. What is the statistical information needed in this problem?
2. Let say the wash machine operation hours is from 8.00am to 9.00am and the machine takes 10 minutes to complete one service. And, the car arrival times are as follows:
    i. 8.05am, 8.09am, 8.12am, 8.25am, and 8.55am.
3. Calculate the statistical information in this case.
4. Determine the simulation characteristic of the ABC Wash Machine.

Tutorial Activity

1. How the program get the input of customer's arrival?
2. List the variables of type C*lock*.
3. What is the data type used to determine the average waiting time and number of customer. Why?
4. List out the standard function and user defined function used in the given program.
5. Identify main and supportive identifiers in the program
6. Which line in the program shows the following:
    - Generate all car arrivals, and push them onto arrival queue.
    - Set the initial state for the wash machine
    - Jump to the next earliest event that is either car arrival or machine finish washing.
    - Update the statistical information.
7. If we were to change the machine working hours from 8am -9am, which line of code need to be change?

Hands-on Activity

Implement the worked-example of *ABC Wash Machine* program. Try to understand the output printed by the program.

Problem: Extension of ABC Wash Machine

DEF WASH MACHINE	
Input	Randomly generated
Output	Standard Output
Java Elements	Selection, Looping
Data Structure	Queue
Additional Function	class Clock, rand(),

Problem Description

The DEF Wash Machine problem will use the same problem description as the worked-example of ABC Wash Machine with regard the DEF Wash Machine provides two types of services:

Type 1: normal wash (takes 6 minutes for every service)
Type 2: wash and polish (takes 10 minutes for every service).

The owner of the DEF Wash Machine would like to know the statistical information as below:

- Total customers: <total_customer>
- Number of customers for service of type 1: <Number_ customer _type_1>
- Number of customers for service of type 2: <Number_ customer _type_2>
- Longest customer waiting time: < Longest_waiting_time>
- Average customer waiting time: < Average_waiting_time>

Your task is to modify the worked-example of ABC Wash Machine and print out the above statistical information.

Input

Input are randomly generated.

Sample Output

Total customers:
Number of customers for service of type 1: <Number_ customer _type_1>
Number of customers for service of type 2: <Number_ customer _type_2>
Longest customer waiting time: < Longest_waiting_time>
Average customer waiting time: < Average_waiting_time>

Tips

1. Queue in DEF Wash Machine problem holds two information: arrival time and service type. So, we would like to suggest you to use class Customer as follows:

```
public class Customer {
       Clock time = new Clock ();
       int serviceType;

public Customer (Clock tm, int sType)
  {
            time.setTime(tm.getHours(),tm.getMinutes(),tm.getSeconds());
            serviceType = sType;
  }

public Customer()
  {
      time.setTime(0,0,0);
         serviceType =0;
  }

public int getSeviceType()
  {
         int temp;
         temp=this.serviceType;

         return temp;
  }

public int setSeviceType()
  {
         int temp;
         temp=this.serviceType;

         return temp;
  }

 public Customer CopyCustomer()
  {
            Customer temp = new Customer();

            temp.time = time.getCopy();
            temp.serviceType= getSeviceType();
            temp.toString();

             return temp;
  }

 public String toString()
  {
       String str;

           str = this.time.toString();
           str = str +  " (" + serviceType + ") ";

           return str;
  }
}
```

2. Be careful with your variable WashEnd. (Why?)
3. Manipulation of a queue which hold single information is simple. However, queue which hold complex information such as class Customer is not easy to handle. Hence, we give you code example on how to manipulate this type of queue.

```
Clock i = new Clock();
Customer car = new Customer();
int nextArrival, carService;
Queue <Customer> myQueue = new Queue<>();

nextArrival = rand.nextInt(10);   // nextArrival in the range 0 to 9
i.addTimeMinute(nextArrival);
car.time = i.getCopy();
car.serviceType = 2;

myQueue.enqueue(car.CopyCustomer());
if(arrivalQueue.peek().serviceType==2)
       System.out.print("service type is 2- Wash Only" );
```

DEF Wash Machine simulation algorithm:

Set the operation time.
*Generate all car arrivals and its service type, and push them onto **arrival queue**.*
Set the initial state for the wash machine (status, startWash, endWash)
Set current event.

While there is still an event

If car arrival event occurs:
Push car onto waiting queue.
If machine finish washing
Update machine state.
If machine available and there is a car waiting:
Front car in waiting queue will be washed.
Remove this car from waiting queue.
Update the machine state and statistical information.
If machine available and there is no car waiting:
Update endWash to beyond operation time.
Jump to the next earliest event that is either car arrival or machine finish washing.

Produce report.

Tutorial Activity

1. Determine the simulation characteristics of the DEF Wash Machine simulation.
2. What is the difference between ABC Wash Machine algorithm and DEF Wash machine algorithm?
3. What is the different between queue in ABC Wash Machine and queue in DEF Wash machine?
4. What are the modifications needed from the ABC Wash Machine code/program?

Other Applications of Queue

Beside the service simulation problem, queue is also used in many other applications. Some of the examples are as below:

1. In a computer systems, queue is used as a “holding area” for messages between two processes, two programs, or even two systems.
2. Serving requests of a single shared resource (printer, disk, CPU)
3. Center phone systems will use a queue to hold people in line until a service is available.