CHAPTER I I
ARRAY & LINKED LIST ITS THEIR APPLICATIONS

2.1 Learning Objectives

1. Students can understand array and linked list
2. Students are able to implement array and linked list and know when to use them
3. Students can understand stack and queue
4. Students are able to implement stack and queue and know when to use them

2.2 Theory

2.2.1 Array

An *array* is one of the basic data structures available and is a collection of many variables with the same data type represented by an index for each element. We can imagine an array like a table, where each element is stored in each tablespace. There are 2 types of arrays, namely one-dimensional arrays called vector and two-dimensional array called matrix.

In Java, an array is a primitive type that is treated as an object. To create an array, you must use the new operator.

```
int[] intArray; // define an array
//create an array and set the array to intArray
intArray = new int[100];
```

The above declaration can be shortened with one statement:

```
int[] intArray = new int[100];
```

This declaration is used to create an array with null elements. If you want to insert elements at the beginning of the array declaration can be done in the following way:

```
//declare array contents
int[] intArray = {0,3,6,9,12,15,18,21,24,27};
```

Accessing an element in the array requires square brackets ([]), similar to other languages. Note that the index in the array starts with 0, meaning the first element is at index 0.

```
//get the contents of the fourth element in the array
temp = intArray[3];
//enter 66 into the eighth cell
intArray[7] = 66;
```

Once an array has been saved, we can use the array to process the data. For example in the case of search and replacement. This program will show how an array is stored, then display its value. Suppose this program asks for input in the form of array size and elements/values to be stored, then displays all values in one line as output. Then, the code continues with the data

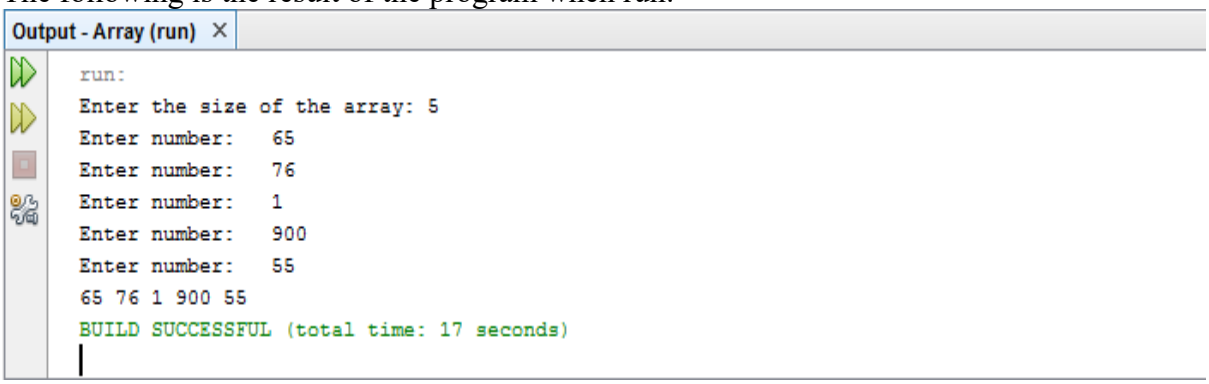
search to find whether an element is stored in the array or not. Furthermore, if data is found, it will replace the data with a certain value, so that a value in the array will be replaced. The implementation of this array can be seen in the following SearchReplacement.java code.

SearchReplacement.java

Source | History

```java
//SearchReplacement.java
//Demonstrates how to create an array and stores and display array elements

package searchreplacement;

import java.util.Scanner;
import java.io.IOException;

public class SearchReplacement {
    public static void main(String[] args) throws IOException{
        int size;                               //size of arrray that we want
        Scanner input = new Scanner(System.in);    //set up the input function;
        System.out.print("Enter the size of the array: ");
        size = input.nextInt();                    //insert the desired array
        System.out.println("");

        int[] arr;                              //reference
        arr = new int[size];                    //make array

        for(int j=0;j<size;j++){                //for each array space
            System.out.print("Enter number " + j +" :\t");
            arr[j] = input.nextInt();              //insert a number
        }
        for(int j=0;j<size;j++){                //for each element in array
            System.out.print(arr[j] + " ");     //print the elements
        }
        System.out.println(size);
        System.out.println("");

         //SEARCHING
        int numSearch;                          //search a number
        System.out.print("What number do you want to search? ");
        numSearch = input.nextInt();

        boolean found = false;                  //assume number is not present
        for(int j=0; j<size; j++){              //check every element
            if(numSearch == arr[j]){            //if the number is found
                found = true;                   //set found to true
                break;                          //exit loop
            }
        }

        if(found)                               //if number is found
            System.out.println(numSearch + " is available");
        else                                    //if number is not found
            System.out.println(numSearch + " not found");

        //REPLACEMENT
        int numToReplace;                       //number to be replaced
        int numReplacing;                       //new number
        System.out.print("What number do you want to replace? ");
        numToReplace = input.nextInt();
        System.out.print("What will be the new number? ");
        numReplacing = input.nextInt();
          for(int j=0; j<size; j++){            //check every element
            if(numToReplace == arr[j]){         //if the number is found
                arr[j] = numReplacing;          //replace the number
            }
        }

        for(int j=0;j<size;j++){                //display the resulting change
            System.out.print(arr[j] + " ");     //print the element
        }
        System.out.println("");
    }   //end main()
}       //end class Array
```

The following is the result of the program when run:



2.2.2 Linked List

Similar to array, linked list is another basic data structure for data storage and processing. However, the difference lies in the use of memory: while the array stores elements in large blocks of memory, the linked list stores each element in a separate memory space called a *node*. These spaces are then connected to each other using *pointer*, so they are called *linked list*.

A linked list requires two types of variables:

- Node

  The main body of the list, each node contains the stored element and a pointer to the next node in the list.

- Pointer

  The head of the list, the pointer contains the memory address of the next node .

In Java, the implementation of a linked list begins with the creation of node and pointer. Then, it is continued with the implementation of the linked list class which contains the initialization of the linked list and the operations that can be performed by the link list. These operations are methods that help us to process and manipulate linked lists. The methods on the linked list is as follows

- *insertFirst() and insertLast()*

  The *insertFirst()* method in the LinkListInit class is used to add a new node at the beginning of the list. While the *insertLast()* method in the LinkListInit class functions to add a new node at the end of the list.

- *deleteFirst()* and *deleteLast()*

  The *deleteFirst()* method in the LinkListInit class is used to delete the first node from the list. The *deleteLast()* method in the LinkListInit class is used to delete the last node from the list. This is done by moving the pointer from the first node

to the next for deleteFirst() and moving the pointer from the last node to the second last node for deleteLast.

- *displayList()*

  This method in LinkListInit is used to display the contents of the list, from the first node to the last node.

The following is an implementation of LinkList.java. In this program, two lists are declared, the first list contains integers inserted from the end, while the second list contains integers inserted at the beginning. The first node in the first list and node last in the second list are then deleted.

LinkList.java

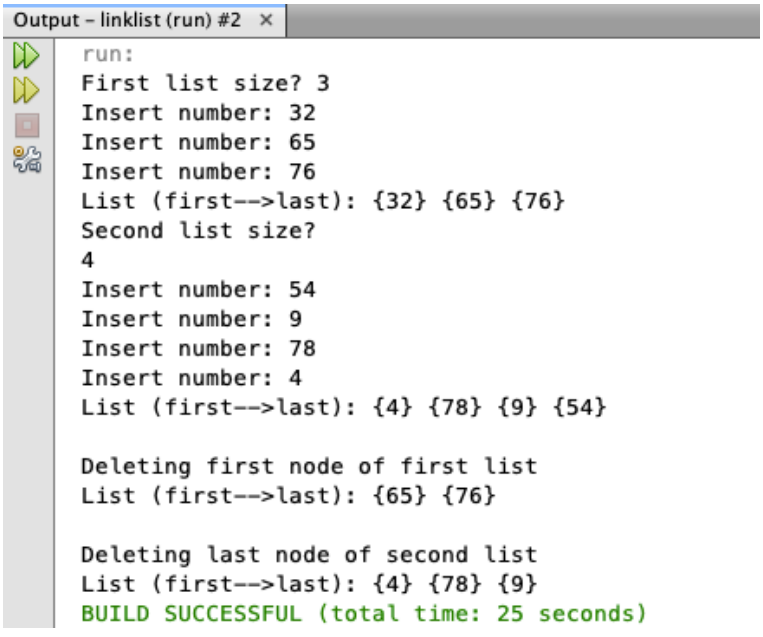
Source | History

```
1   //LinkList.java
2   //demonstrates linkedlist
3   package linklist;
4
5   import java.util.Scanner;
6   import java.io.IOException;
7
8   class Node{
9       public int iData;                   //data item
10      public Node next;                   //next link in list
11
12      public Node(int id){                //constructor
13          iData = id;                     //initialize data
14      }                                   //('next' is automatically set to null)
15
16      public void displayLink(){  //display ourself
17          System.out.print("{" + iData + "} ");
18      }
19  }   //end class Node
20
```

```
21  class LinkListInit{
22      private Node first;                 //ref to first node on list
23
24      public LinkListInit(){              //constructor
25          first = null;                   //no items on list yet
26      }
27
28      public boolean isEmpty(){           //true if list is empty
29          return (first==null);
30      }
31
32      public void insertFirst(int id){    //insert at start of list
33          Node newNode = new Node(id);    //make new node
34          newNode.next = first;           //newNode --> old first
35          first = newNode;                //first --> newNode
36      }
37
38      public void insertLast(int id){     //insert at end of list
39          if(first==null)                 //if the list is empty
40              insertFirst(id);            //create first node
41          else{                           //if not
42              Node temp = first;          //start from first node
43              while(temp.next!=null){     //until we are at the last node
44                  temp = temp.next;       //keep going
45              }
46              temp.next = new Node(id);   //add new node at the end
47          }
48      }
49
50      public Node deleteFirst(){          //delete first item (assume list not empty)
51          Node temp = first;              //save reference to node
52          first = first.next;             //delete it: first --> next node
53          return temp;                    //return deleted node
54      }
55
56      public Node deleteLast(){           //delete last node
57          Node temp = first;              //start from first node
58          while(temp.next.next != null){  //till we are at the second last node
59              temp = temp.next;           //keep going
60          }
61          Node toReturn = temp.next;      //store the last node
62          temp.next = null;               //delete reference to last node
63          return toReturn;                //return deleted node
64      }
65
66      public void displayList(){
67          System.out.print("List (first-->last): ");
68          Node current = first;           //start at the beginning of the list
69          while(current!=null){           //until end of list
70              current.displayLink();      //print data
71              current = current.next;     //move to next link
72          }
73          System.out.println("");
74      }
75  } //end class LinkListInit
76
```

```
77  class LinkList {
78      public static void main(String[] args) throws IOException{
79          LinkListInit theList1 = new LinkListInit();      //make new list
80          LinkListInit theList2 = new LinkListInit();      //make new list
81
82          Scanner in = new Scanner(System.in);             //set scanner
83          int nodeNum1;               //the number of integers for first list
84          int nodeNum2;               //the number of integers for second list
85          int tempNum;                //temporary holder for integer
86
87          System.out.print("First list size? ");
88          nodeNum1 = in.nextInt();    //insert first list size
89
90          for(int i=0; i<nodeNum1; i++){
91              System.out.print("Insert number: ");
92              tempNum = in.nextInt();                  //insert number
93              theList1.insertLast(tempNum);            //on the end of list
94          }
95          theList1.displayList();                      //display list
96
97          System.out.print("Second list size? ");
98          nodeNum2 = in.nextInt();    //insert second list size
99
100         for(int i=0; i<nodeNum2; i++){
101             System.out.print("Insert number: ");
102             tempNum = in.nextInt();                 //insert number
103             theList2.insertFirst(tempNum);          //on the start of list
104         }
105         theList2.displayList();                     //display list
106
107         System.out.println("\nDeleting first node of first list");
108         theList1.deleteFirst();     //deleting the first node of first list
109         theList1.displayList();     //first list after deletion
110
111         System.out.println("\nDeleting last node of second list");
112         theList2.deleteLast();      //deleting the last node of second list
113         theList2.displayList();     //second list after deletion
114     } //end main()
115 } //end class LinkList
```

The following is an example of the program result when run.



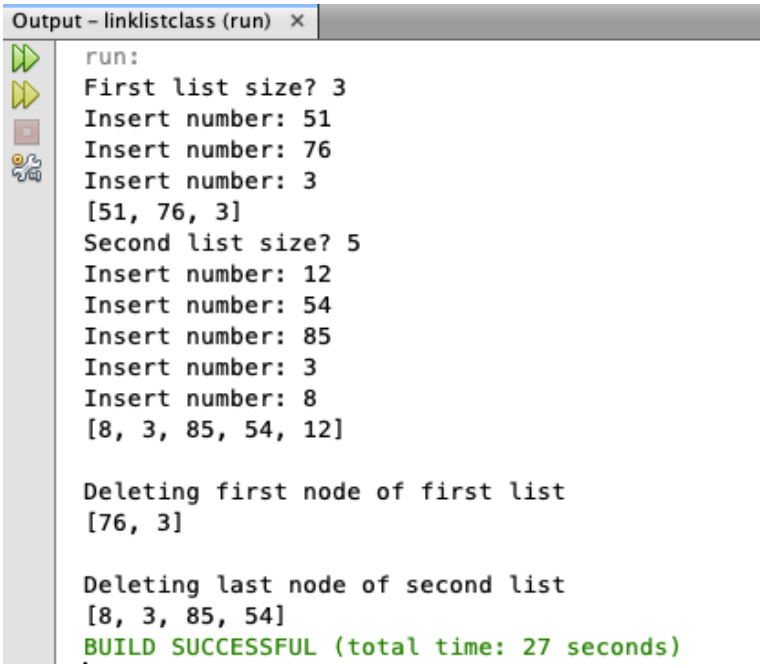
Another implementation of linked lists in Java is the use of the *built-in LinkedList class*. This class contains many operations, including those that have been performed previously, without having to create a new class. Here is an implementation of a linked list using the built-in.

LinkListClass.java

Source | History

```
1   //LinkListClass.java
2   //demonsrates the use of the built-in LinkedListClass
3   package linklistclass;
4   import java.io.IOException;
5   import java.util.Scanner;
6   import java.util.LinkedList;    //important for LinkedListClass
7
8   public class LinkListClass {
9       public static void main(String[] args) throws IOException{
10          LinkedList theList1 = new LinkedList(); //make new list
11          LinkedList theList2 = new LinkedList(); //make new list
12
13          Scanner in = new Scanner(System.in);    //set scanner
14          int nodeNum1;               //the number of integers for first list
15          int nodeNum2;               //the number of integers for second list
16          int tempNum;                //temporary holder for integer
17
18          System.out.print("First list size? ");
19          nodeNum1 = in.nextInt();        //insert first list size
20
21          for(int i=0; i<nodeNum1; i++){
22              System.out.print("Insert number: ");
23              tempNum = in.nextInt();     //insert number
24              theList1.addLast(tempNum);  //on the end of list
25          }
26          System.out.println(theList1);   //display list
27
28          System.out.print("Second list size? ");
29          nodeNum2 = in.nextInt();        //insert second list size
30
31          for(int i=0; i<nodeNum2; i++){
32              System.out.print("Insert number: ");
33              tempNum = in.nextInt();     //insert number
34              theList2.addFirst(tempNum); //on the start of list
35          }
36          System.out.println(theList2);   //display list
37
38          System.out.println("\nDeleting first node of first list");
39          theList1.removeFirst();         //deleting the first node of first list
40          System.out.println(theList1);   //first list after deletion
41
42          System.out.println("\nDeleting last node of second list");
43          theList2.removeLast();          //deleting the last node of second list
44          System.out.println(theList2);   //second list after deletion
45      }
46  }
```

The following is the result of the program when it is run.



2.2.3 Stack

A *stack* is a linear data structure that allows access to only one data item: the last item inserted. When deleting an item, the item accessed is the last item inserted. An example of a stack is like a pile of clothes: clothes are placed in the cupboard on the topmost pile and clothes are taken from the topmost pile. A stack uses the Last-In-First-Out (LIFO) storage mechanism because the last item inserted is the first item to be removed. The stack can be implemented using an array and linked list.

The stack has some basic operations (methods) as follows.

- Initialize stack

  Initialize the stack by creating an empty stack or a stack that has no elements in it
- Push

  Push is an operation to add elements to the top of the stack. The algorithm is as follows:

  a. Create a new data that will be inserted (pushed) into the stack
  b. Insert (push) the data into the stack
  c. Modify the top pointer to point to the data just inserted
- Pop

  Pop is an operation to remove elements from the stack. Assuming the stack contains several elements, the algorithm is as follows:

a. Take and remove the topmost data from the stack

b. Modify the top pointer to point to the next top data

- isEmpty

  isEmpty is an operation to check whether the stack contains elements or not. The operation returns true if the stack is empty and returns false if otherwise.

- printStack

  printStack is an operation to print the stack.

The following is an example of implementing a stack using arrays on Stack.java. The program stores the input (in numbers) on a stack, then displays the stored numbers.

Stack.java

Source | History

```
1  // Stack.java
2  // demonstrates stack
3  package stack;
4  import java.io.IOException;      //exception for I/O
5  import java.util.Scanner;        //for input
6  import java.util.Arrays;
7
8  class StackInit{  //contains stack methods
9      private final int maxSize;  //size of stack array
10     private int[] stackArray;   //initialize array
11     private int top;            //top of stack
12
13     public StackInit(int s){            //constructor
14         maxSize = s;                    //set array size
15         stackArray = new int[maxSize];  //create array
16         top = -1;                       //no items yet
17     }
18
19     public void push(int j){            //put item on top of stack
20         stackArray[++top] = j;          //increment top, insert item
21     }
22
23     public double pop(){                //take item from top of stack
24         return stackArray[top--];       //access item, decrement top
25     }
26
27     public boolean isEmpty(){           //true if stack is empty
28         return(top == -1);
29     }
30
31     public void printStack(){
32         System.out.println(Arrays.toString(stackArray));
33     }
34 } // end class StackInit
35
```

```
36  public class Stack {
37      public static void main(String[] args) throws IOException {
38          int stackSize;              //stack size
39          int stackNum;               //number to be inserted in stack
40          Scanner in = new Scanner(System.in);
41
42          System.out.print("How many integer? ");
43          stackSize = in.nextInt();   //insert stack size
44
45          StackInit theStack = new StackInit(stackSize);  //make new stack
46
47          for(int i=0; i<stackSize; i++){
48              System.out.print("Enter number: ");
49              stackNum = in.nextInt();        //insert number
50              theStack.push(stackNum);        //push element onto stack
51          }
52          theStack.printStack();              //print Stack
53
54          while(!theStack.isEmpty()){         //until it is empty, delete item from stack
55              double value = theStack.pop();
56              System.out.print(value);        //display the popped item
57              System.out.print(" ");
58          }
59
60          System.out.println("");
61      }   //end main()
62  }   //end class Stack
```

The following is the result of the program when it is run

```
Output - stack (run)
run:
How many integer? 4
Enter number: 1
Enter number: 2
Enter number: 6
Enter number: 7
[1, 2, 6, 7]
7.0 6.0 2.0 1.0
BUILD SUCCESSFUL (total time: 12 seconds)
```

Java has a built-in Stack class which is imported via java.util.Stack . Here is an example program similar to the previous one but using the Stack class on StackBasic.java

StackBasic.java

```java
// StackBasic.java
// demonstrates the built-in stack
package stackbasic;
import java.io.IOException;             //exception for I/O
import java.util.Scanner;               //for input
import java.util.Stack;                 //for Stack class

public class StackBasic {
    public static void main(String[] args) throws IOException {
        int stackSize;                  //stack size
        int stackNum;                   //number to be inserted in stack

        Scanner in = new Scanner(System.in);

        System.out.print("How many integer? ");
        stackSize = in.nextInt();       //insert stack size

        Stack theStack = new Stack();   //make new stack

        for(int i=0; i<stackSize; i++){
            System.out.print("Enter number: ");
            stackNum = in.nextInt();    //insert number
            theStack.push(stackNum);    //push element onto stack
        }

        System.out.print(theStack);
        System.out.println("");

        while(!theStack.isEmpty()){     //until it is empty, delete item from stack
            Integer value = (Integer) theStack.pop();
            System.out.print(value);    //display the popped item
            System.out.print(" ");
        }

        System.out.println("");
    }
}
```

The following is the result of the program when it is run.

Output - StackBasic (run)

```
run:
How many integer? 5
Enter number: 12
Enter number: 25
Enter number: 76
Enter number: 39
Enter number: 9
[12, 25, 76, 39, 9]
9 39 76 25 12
BUILD SUCCESSFUL (total time: 30 seconds)
```

2.2.4 Queue

A queue is a linear data structure similar to a stack, but the addition and deletion of data items are done at opposite ends. Items can be inserted from one end and deleted from the other. An example of a queue is like a supermarket queue: customers enter from one end of the queue and exit from the other to the cashier. A queue is said to be a First-In-First-Out (FIFO) storage mechanism because the first item to be inserted is the first to be removed. Queue can be implemented in two ways: array and linked list.

Queue has some basic operations (methods) as follows.

- Initialize queue

  Initialize the queue by creating an empty queue or a queue that has no elements in it.

- Enqueue

  Enqueue is the operation of adding a new element to the back end of the queue. The algorithm is as follows:

  a. Create a new data that will be inserted (pushed) into the queue
  b. Insert the data into the queue
  c. Change the last pointer to point to the newly inserted data

- Dequeue

  Dequeue is the operation of removing an element from the front end of the queue. Assuming the queue contains several elements, the algorithm is as follows:

  a. Take and remove the frontmost data from the queue
  b. Change the first pointer to point to the next forward data

- isEmpty

  isEmpty is an operation to check whether the queue contains elements or not. The operation returns true if the queue is empty and returns false if otherwise.

- isFull

  The opposite of isEmpty, the operation returns true if the queue is full and returns false if otherwise.

- printQueue

  printQueue is an operation to print a queue.

The following is an example of implementing a queue using an array in Queue.java. The program will implement a queue using the QueueInit class and operate on the appropriate command.

Queue.java

Source | History

```java
// Queue.java
// demonstrates queue
package queue;
import java.io.IOException;             //exception for I/O
import java.util.Scanner;               //for input
import java.util.Arrays;

class QueueInit{ //contains queue methods
    private int maxSize;
    private int[] queueArray;
    private int front;
    private int rear;
    private int nItems;

    public QueueInit(int s){            //constructor
        maxSize = s;
        queueArray = new int[maxSize];
        front = 0;
        rear = -1;
        nItems = 0;
    }

    public void enqueue(int j){         //put item at rear of queue
        if (rear == maxSize-1)          //deal with wraparound
            rear = -1;
        queueArray[++rear] = j;         //increment rear and insert
        nItems++;                       //one more item
    }

    public int dequeue(){               //take item from front of queue
        int temp = queueArray[front++]; //get value and increment front
        if(front == maxSize)            //deal with wraparound
            front = 0;
        nItems--;                       //one less item
        return temp;
    }

    public boolean isEmpty(){           //true if queue is empty
        return (nItems==0);
    }

    public boolean isFull(){            //true if queue is full
        return (nItems==maxSize);
    }

    public void printQueue(){
        System.out.println(Arrays.toString(queueArray));
    }
}
```

```
51  public class Queue {
52      public static void main(String[] args) throws IOException{
53          int queueSize;                  //for queue size
54          int numTemp;                    //for inserted number
55          int numChoice = 0;              //for command
56
57          Scanner in = new Scanner(System.in);            //for input
58          System.out.print("Enter queue size: ");
59          queueSize = in.nextInt();
60
61          QueueInit theQueue = new QueueInit(queueSize);  //set queue
62
63          while(numChoice != 3){
64              System.out.println("\n 1: Enqueue \t 2 : Dequeue \t 3 : End");
65              System.out.print("Enter command: ");
66              numChoice = in.nextInt();
67              if(numChoice == 1){
68                  if(theQueue.isFull())
69                      System.out.println("Queue is full");
70                  else{
71                      System.out.print("Enter number: ");
72                      numTemp = in.nextInt();
73                      theQueue.enqueue(numTemp);
74                  }
75              }
76              else if(numChoice == 2){
77                  if(theQueue.isEmpty())
78                      System.out.println("Queue is empty");
79                  else{
80                      numTemp = theQueue.dequeue();
81                      System.out.println("Dequeue number: " + numTemp);
82                  }
83              }
84              else if(numChoice != 3){
85                  System.out.println("Wrong command");
86              }
87          }   //end main()
88      }   //end class Queue
89  }
```

The following is the result of the program when it is run.

```
Output - Queue (run)
run:
Enter queue size: 5

 1: Enqueue      2 : Dequeue     3 : End
Enter command: 1
Enter number: 23

 1: Enqueue      2 : Dequeue     3 : End
Enter command: 1
Enter number: 45

 1: Enqueue      2 : Dequeue     3 : End
Enter command: 2
Dequeue number: 23

 1: Enqueue      2 : Dequeue     3 : End
Enter command: 2
Dequeue number: 45

 1: Enqueue      2 : Dequeue     3 : End
Enter command: 3
BUILD SUCCESSFUL (total time: 11 seconds)
```

Queues are usually used in situations where elements can be inserted one by one and processed one on by one thus requiring the use of a queue. One example is the banking queue, where people wait in a line to get banking services. The following is an example of this implementation on BankService.java. This program makes it possible to add customers to the queue, remove customers from the queue and check the queue situation. Note that this program uses the LinkedList class to create a queue.

BankService.java

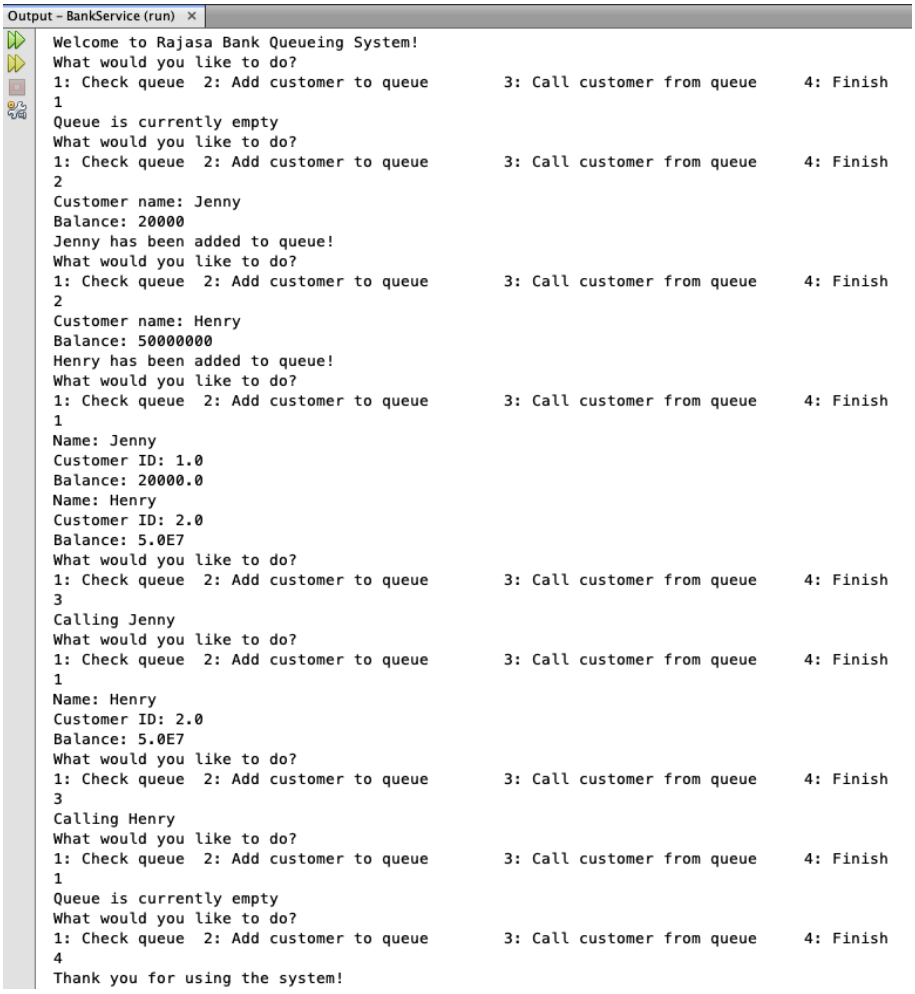
Source | History

```
1   //BankService.java
2   //demonstrates an application of queue using LinkedList
3   package bankservice;
4   import java.util.Scanner;       //for input
5   import java.io.IOException;     //for I/O
6   import java.util.LinkedList;    //for linked list
7   import java.util.Iterator;      //for list iteration
8
9   class Customer{                 //customer object constructor
10      private String name;        //customer name
11      private double idCust, balance; //customer id and balance
12
13      public Customer(String _name, double _idCust, double _balance){ //create customer
14          name = _name;
15          idCust = _idCust;
16          balance = _balance;
17      }
18
19      public String getName(){    //return customer name
20          return name;
21      }
22
23      public double getIdCust(){  //return customer id
24          return idCust;
25      }
26
27      public double getBalance(){ //return customer balance
28          return balance;
29      }
30  }   //end class Customer
31
```

```
class BankServiceStart{                 //queue constructor
    private LinkedList<Customer> list;  //declare linked list for queue

    public BankServiceStart(){
        list = new LinkedList<>();      //create linked list
    }

    public boolean isEmpty(){           //true if queue is empty
        return(list.isEmpty());
    }

    public void enqueue(Customer item){ //add customer to queue
        list.addLast(item);
    }

    public void dequeue(){              //remove customer from queue
        list.removeFirst();
    }

    public Customer callCust(){         //get front customer name
        return list.getFirst();
    }

    public void displayQueue(){
        Iterator<Customer> i = list.iterator();
        if(!i.hasNext())
            System.out.println("Queue is currently empty");
        else{
            while(i.hasNext()){
                Customer tempCust = i.next();
                System.out.println("Name: " + tempCust.getName() +  //print customer info
                        "\nCustomer ID: " + tempCust.getIdCust() +
                        "\nBalance: " + tempCust.getBalance());
            }
        }
    }
} //end class BankServiceStart

```

```java
public class BankService {
    public static void main(String[] args) throws IOException {
        String name_m;          //for getting customer name
        double idCust_m = 0;    //for getting customer id
        double balance_m;       //for getting customer balance
        int queueCom = 0;       //for queue command

        Scanner in = new Scanner(System.in);    //for input
        BankServiceStart bankQueue = new BankServiceStart();    //create queue

        //creating a bank system
        System.out.println("Welcome to Rajasa Bank Queueing System! ");

        while(queueCom != 4){       //while we are not done
            System.out.println("What would you like to do?");   //ask command
            System.out.println("1: Check queue\t" +
                    "2: Add customer to queue\t" +
                    "3: Call customer from queue\t" +
                    "4: Finish");
            queueCom = in.nextInt();
            switch(queueCom){
                case 1:
                    bankQueue.displayQueue();
                    break;
                case 2:
                    System.out.print("Customer name: ");
                    name_m = in.next();
                    idCust_m++;
                    System.out.print("Balance: ");
                    balance_m = in.nextDouble();
                    //create customer object
                    Customer newCust = new Customer(name_m, idCust_m, balance_m);
                    bankQueue.enqueue(newCust);
                    System.out.println(name_m + " has been added to queue!");
                    break;
                case 3:
                    if(bankQueue.isEmpty())
                        System.out.println("Queue is empty");
                    else{
                        Customer toServe = bankQueue.callCust();
                        System.out.println("Calling " + toServe.getName());
                        bankQueue.dequeue();
                    }
                    break;
                case 4:
                    System.out.println("Thank you for using the system!");
                    break;
                default:
                    System.out.println("Incorrect command. Try again.");
            }
        }
    }   //end main
}       //end class BankService
```

The following is an example of the output when the program is run.



2.3 Tasks

1. Using an array, create a program that can add up all the numbers inserted by the user. Note that the program will read the number of numbers to be inserted and the numbers to be inserted. Do the same as the question above, but use a linked list.
2. Create an array that stores 10 numbers, then triple each number.
3. Create a program to reverse the order of a string of characters inserted by the user (eg REVELATION -> NOITALEVER)
4. Create a program that checks whether a string is a palindrome (a palindrome is a string reads from the forward or backward, for example: TACOCAT)
5. Create a program that calculates the parking fee that must be paid for each car in the parking lot. Each car has information on the car model and parking time. Assume the rate is Rp. 2000 per hour.