VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnana Sangama", Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K. R. Road, V. V. Puram, Bengaluru-560 004

Department of Artificial Intelligence and Machine Learning
Java Mini Project Report - 18CS42 & 18CS45

ON

"TIME IS MONEY"

Submitted By

1BI20AI003	ADITYA GANESH SHANBHAG
1BI20AI029	MANOGNYA LOKESH REDDY
1BI20AI045	SHARATH

for the academic year 2022

Faculty In-Charge

Prof. Shruthiba A
Department of Artificial Intelligence and Machine Learning
BIT, Bangalore

ACKNOWLEDGEMENT

The knowledge & satisfaction that accompany the successful completion of any task would be incomplete without mention of people who made it possible, whose guidance and encouragement crowned my effort with success. I would like to thank all and acknowledge the help I have received to carry out this Mini Project.

I would like to convey my sincere thanks to **Dr. M. U. Aswath**, Principal, BIT and **Dr. Jyothi D G**, HOD, Department of AI&ML, BIT for being kind enough to provide the necessary support to carry out the mini project.

I am humbled to mention the enthusiastic influence provided by the faculty in-charge **Prof. Shruthiba A,** on the project for the ideas, time to time suggestions for being a constant guide and co-operation showed during the venture and making this project a great success.

I would also take this opportunity to thank my friends and family for their constant support and help. I'm very much pleasured to express my sincere gratitude to the friendly co-operation showed by all the staff members of AIML Department, BIT.

ADITYA
MANOGNYA
SHARATH

TABLE OF CONTENTS

1. INTRODUCTION

“The time is the most valuable thing.” -Vinay Chhabra & Manish Dewan. Time is much more valuable for engineers since it is a natural constraint that can not be stopped and what engineers do is to fight with constraints.

A food processing factory processes the raw materials sent by customers and turns theminto the desired final product.There are two machines in the factory. These are the A and B machines.For a raw material to be converted into the desired final product, it must be processed in A and B machines at certain predetermined times depending on the properties of the product. This process takes place sequentially.Machine A takes the raw material and turns it into an intermediate product,while machine B transforms this intermediate productinto the final product.The raw materialis perishable before entering machine A,so it must be sent to the factory by the customer exactly at the time it should enter the machine A.The intermediate product produced by machine A is not perishable and can be sent to the machine B at any time to be processed.

The factory has received an order. In this order, the names of the requested products,the time required for the raw material to be spent in the A and B machines to convert the products into the final product,and the profits to be obtained from the products are written respectively.Factory engineer is a smart person with good time management. S/he listed all these jobs as the shortest time from the beginning of the first job to the end of the last job.

According to this list,s/he determined the start time of each job. S/he also determined the time when all jobs will be done.S/he sent these start times to the customer so that they could send the required raw materials exactly at that time, and the end time was the deadline for the contract to be made.The customer found these deadlines appropriate, and the contract was made.If the factory cannot grow a product within the specified deadline,it cannot receive the money it should receive for this product.

After the contract was signed,inspectors from the Ministry of Environment and Urbanization came to the factory.Investigators inspected machines A and B and sealed them as they were not environmentally friendly. The factory, which will no longer use these two machines,has bought a new machine that performs the work of these two machines on a single belt.This machine takes the raw material and turns it directly into the final product. The time taken for this process is equal to one of the times that the product must spend in machine A or B in the order list.If the product is a solid type, the total time taken for the new machine to produce the product is equal to the time the product must spend in machine A in the order list; if the product is a liquid type, the total time it takes for the new machine to produce the product is equal to the time the product must spend in machine B in the order list.

The engineer examined the new machine and predicted that under these conditions, some of the products in the contract may not be grown within the specified deadline.Therefore, if it cannot grow all of them,it has decided to produce some of the products in a way that will maximize the profit of the factory. In this project, you are asked to calculate the maximum profit that the company can achieve in this case by looking at the order list given to you by the factory engineer.

2.1 PROBLEM STATEMENT

YOU ARE ASKED TO CALCULATE THE MAXIMUM PROFIT THAT THE COMPANY CAN ACHIEVE IN THE GIVEN SCENARIO BY LOOKING AT THE ORDER LIST GIVEN TO YOU BY THE FACTORY ENGINEER.

Scenario:

- A new machine is bought that performs the work of the two machines A and B on a single belt.This machine takes the raw material and turns it directly into the final product.
- It is predicted that under these conditions,some of the products in the contract may not be grown within the specified deadline.It is decided to produce some of the products in a way that will maximize the profit of the factory.
- The time taken for this process is equal to one of the times that the product must spend in machine A or B in the order list.
    - If the product is a solid type, the total time taken for the new machine to produce the product is equal to the time the product must spend in machine A in the order list.
    - If the product is a liquid type, the total time it takes for the new machine to produce the product is equal to the time the product must spend in machine B in the order list.

2.2 DETAILS

- For each case, output consists of one integer, int1 . It is is the maximum profit that the factory can gain after machine A and machine B are banned and new machine starts to process.
- Customers sends raw material on the day in the contract. Dates are determined according to times that products started to be processed on machine A. Raw material can not be sent before or after that time because of its perishable properties.
- Start and end times of the products that may be processed on the new machine are determined based on the type of products. Start time is always the time when raw material is reached to the factory.(It is actually starting time of the process on machine A for each product). End time is start time + X where X is machine A’s process time of the product if the product is solid otherwise machine B’s process time of the product.

3.1 INPUTS

- The first line represents the type of products respectively.
- The second line represents the processing times of the products on machine A respectively.
- The third line represents the processing times of the products on machine B respectively.
- The fourth line represents the profit gained by factory by processing products respectively.
- The fifth line represents the receiving times of each products according to the signed contract.

SAMPLE INPUT	EXPLANATIONS
s l s l l	Types of product1, product2, product3, product4, and product5 respectively. ’s’ means ”solid”, ’l’ means ”liquid”.
6 2 10 4 11	Minimum times are 6, 2, 10, 4, and 11 for product1, product2, product3, product4, and product5 on machine A respectively.
3 7 8 9 5	Minimum times are 3, 7, 8, 9, and 5 for product1, product2, product3, product4, and product5 on machine B respectively.
10 5 7 6 8	Factory gains 10, 5, 7, 6, and 8 profits if it produce product1,product2, product3, product4, and product5 respectively.
27 0 6 2 16	arrival times are 27, 0 , 6, 2, 16 for product1, product2, product3, product4 and product 5 respectively.

Table 1: Sample Input with Explanations

Products	**NEW MACHINE**		**Profits**
	Time in	Time out	
1	27	33	10
2	0	7	5
3	6	16	7
4	2	11	6
5	16	21	8

Table 2: Possible processing intervals of products on new machine.

3.2 OUTPUT

An integer representing the potential maximum profit of the factory.

OUTPUT	EXPLANATION
25	Maximum profit of factory that the new machine can generate.

4. ALGORITHM

Pseudocode on DYNAMIC PROGRAMMING to find the maximum profit

Algorithm : Find Maximum Profit

Parameters offers: list of objects containing start time, end time, and profit of each order. It is sorted based on end time **totalOffer**: total number of offers.

```
1: maxProfit ← [0] *totalOffer
2: for iteration = 1,2,...,n do
3:       j ← iteration -1
4:       while j >= 0 and offers[j - 1].endTime > offers[i - 1].startTime do
5:       j ← j-1
6:       end while
7: maxProfit[i] ← max(maxProfit[i-1], maxProfit[j] + offers[i-1].profit)
8: end for
9: return maxProfit[n]
```

"Those who cannot remember the past are condemned to repeat it."

- George Santayana, The Life of Reason.

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

Dynamic programming amounts to **breaking down an optimization problem** into simpler sub-problems, and **storing the solution to each sub-problem** so that each sub-problem is only solved once.

STEPS TO SOLVE A DP PROBLEM

Step 1: *Identify the sub-problem in words.*

Step 2: *Write out the sub-problem as a recurring mathematical decision.*

```
OPT(i) = max(v_i + OPT(next[i]), OPT(i+1))
```

Step 3: *Solve the original problem using Steps 1 and 2.*

Step 4: *Determine the dimensions of the memoization array and the direction in which it should be filled.*

```
OPT(1) = max(v_1 + OPT(next[1]), OPT(2))
```

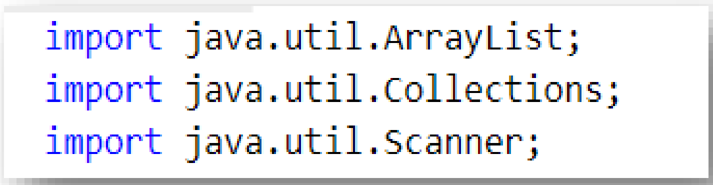
Step 5: *Implement in code.*

SAMPLE CODE IN JAVA

```
// initialize to -1
int dp[MAXN];

// this function returns the number of arrangements to form 'n'.
int solve(int n)
{

        // base case
        if (n < 0)
                return 0;
        if (n == 0)
                return 1;

        // checking if already calculated
        if (dp[n]!=-1)
                return dp[n];

        // storing the result and returning
        return dp[n] = solve(n-1) + solve(n-3) + solve(n-5);
}
```

5. CODE IMPLEMENTATION

➢ IMPORTED CLASSES



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

1. java.util.ArrayList

ArrayList is a part of collection framework and is present in java.util package. It provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.
Since ArrayList is a dynamic array and we do not have to specify the size while creating it, the size of the array automatically increases when we dynamically add and remove items. Though the actual library implementation may be more complex, the following is a very basic idea explaining the working of the array when the array becomes full and if we try to add an item:

- Creates a bigger-sized memory on heap memory (for example memory of double size).
- Copies the current memory elements to the new memory.
- New item is added now as there is bigger memory available now.
- Delete the old memory.

2. java.util.Collections

Collections class is basically used with the static methods that operate on the collections or return the collection. All the methods of this class throw the NullPointerException if the collection or object passed to the methods is null.The collection class basically contains 3 fields as listed below which can be used to return immutable entities. The java. util package contains the Collections class.

- EMPTY_LIST to get an immutable empty List
- EMPTY_SET to get an immutable empty Set
- EMPTY_MAP to get an immutable empty Map

3. java.util.Scanner

The java.util.Scanner class is a simple text scanner which can parse primitive types and strings using regular expressions.Scanner is a class in java. util package used for obtaining the input of the primitive types like int, double, etc. and strings. It is the easiest way to read input in a Java program, though not very efficient if you want an input method for scenarios where time is a constraint like in competitive programming.

➢ IN-BUILT METHODS

1. get()

The get() method of ArrayList in Java is used to get the element of a specified index within the list.
SYNTAX: arraylistObj.get(index);

2. max()

The Java.lang.math.max() function is an inbuilt function in Java which returns maximum of two numbers. The arguments are taken in int, double, float and long.If a negative and a positive number is passed as argument then the positive result is generated. And if both parameters passed are negative then the number with the lower magnitude is generated as result.
SYNTAX: Math.max(op1,op2);

3. replaceAll()

The Java String class replaceAll() method returns a string replacing all the sequence of characters matching regex and replacement string. Regex : regular expression, Replacement : replacement sequence of characters.
SYNTAX: stringVar.replaceAll(regex,replacement);

4. toCharArray()

The java string toCharArray() method converts the given string into a sequence of characters. The returned array length is equal to the length of the string.
SYNTAX: char var[]= stringVar.toCharArray();

5. add()

The add() method is present in java.util.ArrayList class .This method appends the specified element to the end of the arraylist.
SYNTAX: arrayListVar.add(variable);

6. sort()

sort() method is present in java.util.Collections class. It is used to sort the elements present in the specified list of Collection in ascending order. It works similar to java.util.Arrays.sort() method but it is better , as it can sort the elements of Array as well as linked list, queue and many more present in it.
SYNTAX: Collections.sort(object);

➢ INTERFACE

• COMPARABLE<>

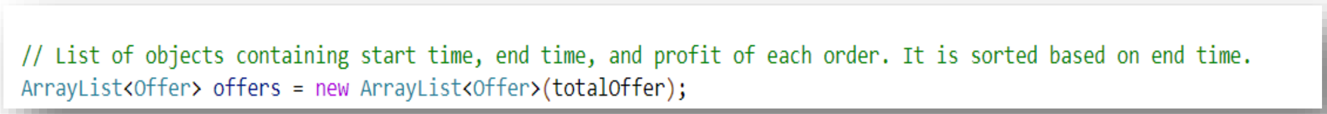
Java Comparable interface is used to order the objects of the user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be rollno, name, age or anything else.

public int compareTo(Object obj): It is used to compare the current object with the specified object. It returns

- ❖ positive integer, if the current object is greater than the specified object.
- ❖ negative integer, if the current object is less than the specified object.
- ❖ zero, if the current object is equal to the specified object.

➢ DATA STRUCTURE

• Arraylist



```
// List of objects containing start time, end time, and profit of each order. It is sorted based on end time.
ArrayList<Offer> offers = new ArrayList<Offer>(totalOffer);
```

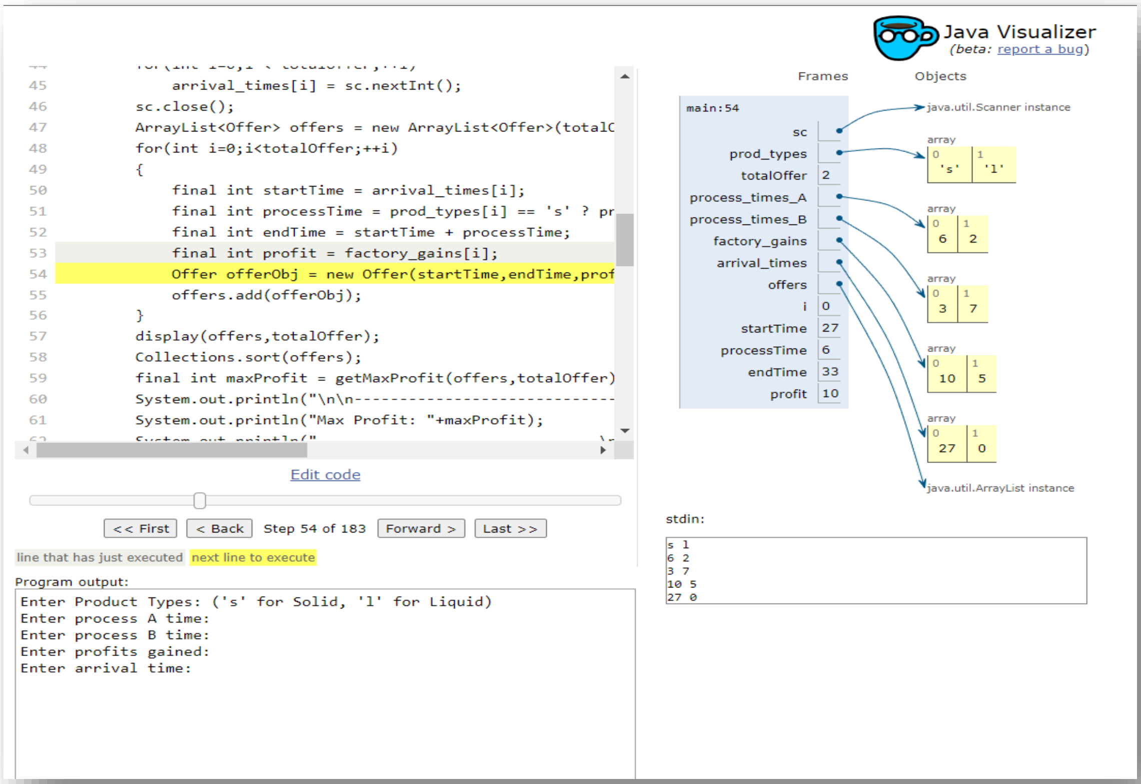
ArrayList is a resizable array implementation in java. ArrayList grows dynamically and ensures that there is always a space to add elements. The backing data structure of ArrayList is an array of Object classes. ArrayList class in Java has 3 constructors. It has its own version of readObject and writeObject methods. Object Array in ArrayList is transient. It implements RandomAccess, Cloneable, and java.io.Serializable (which are Marker Interface in Java).

ArrayList(): This constructor is used to create an empty ArrayList with an initial capacity of 10 and this is a default constructor. We can create an empty Array list by reference name arr_name object of ArrayList class as shown below.
ArrayList arr_name = new ArrayList();

Angle Bracket in Java is used to define Generics. It means that the angle bracket takes a generic type, say T, in the definition and any class as a parameter during the calling. The idea is to allow type (Integer, String, … etc and user-defined types) to be a parameter to methods, classes, and interfaces. For example, classes like HashSet, ArrayList, HashMap, etc use generics very well. We can use them for any type.

```
<T> // of type T
<Integer> // of type Integer
<String> // of type String
<MyClass> // of type MyClass
```

6. FLOW VISUALIZATION



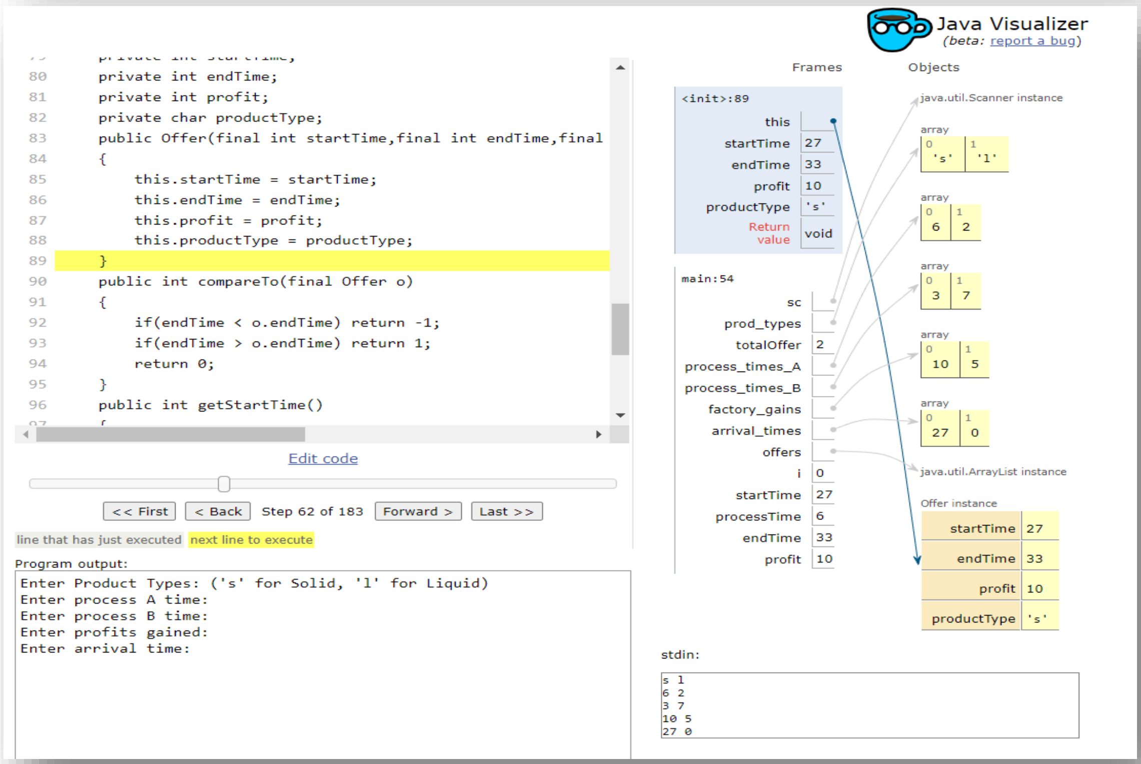
```
            arrival_times[i] = sc.nextInt();
        sc.close();
        ArrayList<Offer> offers = new ArrayList<Offer>(totalC
        for(int i=0;i<totalOffer;++i)
        {
            final int startTime = arrival_times[i];
            final int processTime = prod_types[i] == 's' ? pr
            final int endTime = startTime + processTime;
            final int profit = factory_gains[i];
            Offer offerObj = new Offer(startTime,endTime,prof
            offers.add(offerObj);
        }
        display(offers,totalOffer);
        Collections.sort(offers);
        final int maxProfit = getMaxProfit(offers,totalOffer)
        System.out.println("\n\n-----------------------------
        System.out.println("Max Profit: "+maxProfit);
```

Program output:

```
Enter Product Types: ('s' for Solid, 'l' for Liquid)
Enter process A time:
Enter process B time:
Enter profits gained:
Enter arrival time:
```

stdin:

```
s l
6 2
3 7
10 5
27 0
```



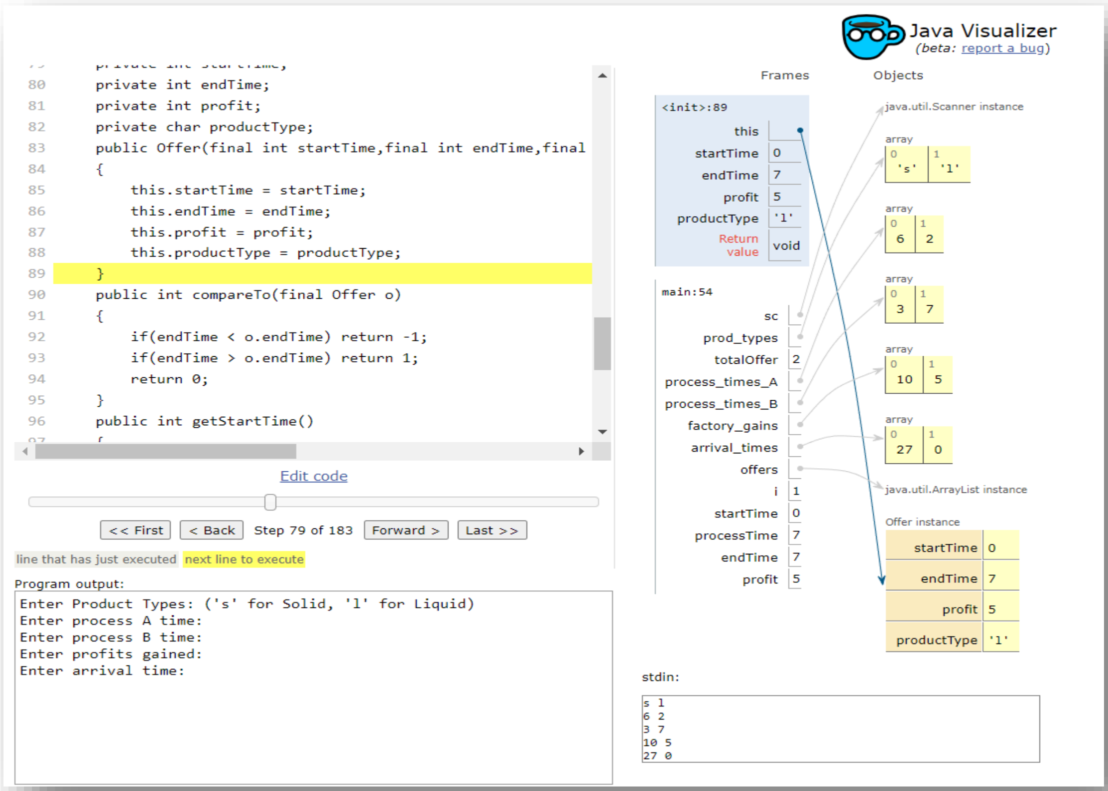
```
    private int endTime;
    private int profit;
    private char productType;
    public Offer(final int startTime,final int endTime,final
    {
        this.startTime = startTime;
        this.endTime = endTime;
        this.profit = profit;
        this.productType = productType;
    }
    public int compareTo(final Offer o)
    {
        if(endTime < o.endTime) return -1;
        if(endTime > o.endTime) return 1;
        return 0;
    }
    public int getStartTime()
```

Program output:

```
Enter Product Types: ('s' for Solid, 'l' for Liquid)
Enter process A time:
Enter process B time:
Enter profits gained:
Enter arrival time:
```

stdin:

```
s l
6 2
3 7
10 5
27 0
```



```
    private int endTime;
    private int profit;
    private char productType;
    public Offer(final int startTime,final int endTime,final
    {
        this.startTime = startTime;
        this.endTime = endTime;
        this.profit = profit;
        this.productType = productType;
    }
    public int compareTo(final Offer o)
    {
        if(endTime < o.endTime) return -1;
        if(endTime > o.endTime) return 1;
        return 0;
    }
    public int getStartTime()
```

Frames

```
<init>:89
          this
     startTime  0
       endTime  7
        profit  5
   productType  '1'
  Return value  void

main:54
              sc
      prod_types
      totalOffer  2
 process_times_A
 process_times_B
   factory_gains
   arrival_times
          offers
               i  1
       startTime  0
     processTime  7
         endTime  7
          profit  5
```

Objects

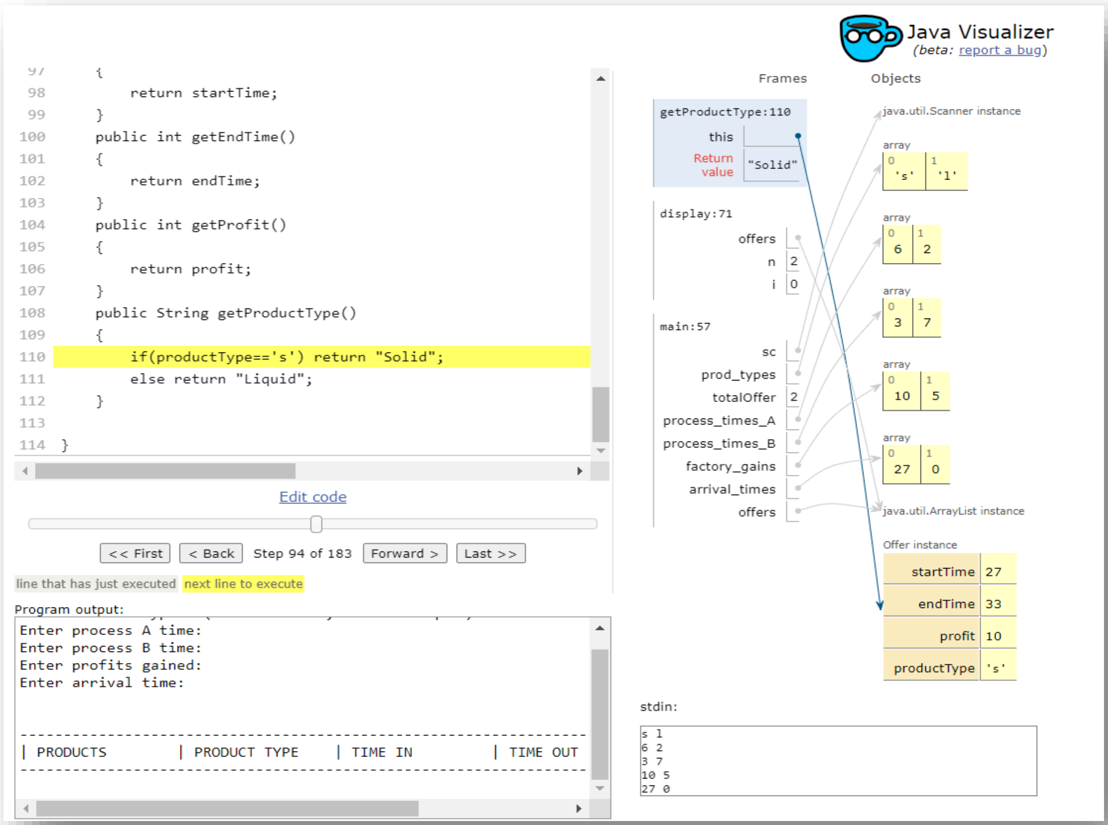
```
java.util.Scanner instance
array  0: 's'  1: '1'
array  0: 6    1: 2
array  0: 3    1: 7
array  0: 10   1: 5
array  0: 27   1: 0
java.util.ArrayList instance
Offer instance
  startTime    0
  endTime      7
  profit       5
  productType  '1'
```

Program output:

```
Enter Product Types: ('s' for Solid, '1' for Liquid)
Enter process A time:
Enter process B time:
Enter profits gained:
Enter arrival time:
```

stdin:

```
s 1
6 2
3 7
10 5
27 0
```



```
        return startTime;
    }
    public int getEndTime()
    {
        return endTime;
    }
    public int getProfit()
    {
        return profit;
    }
    public String getProductType()
    {
        if(productType=='s') return "Solid";
        else return "Liquid";
    }

}
```

Frames

```
getProductType:110
          this
  Return value  "Solid"

display:71
        offers
             n  2
             i  0

main:57
              sc
      prod_types
      totalOffer  2
 process_times_A
 process_times_B
   factory_gains
   arrival_times
          offers
```

Objects

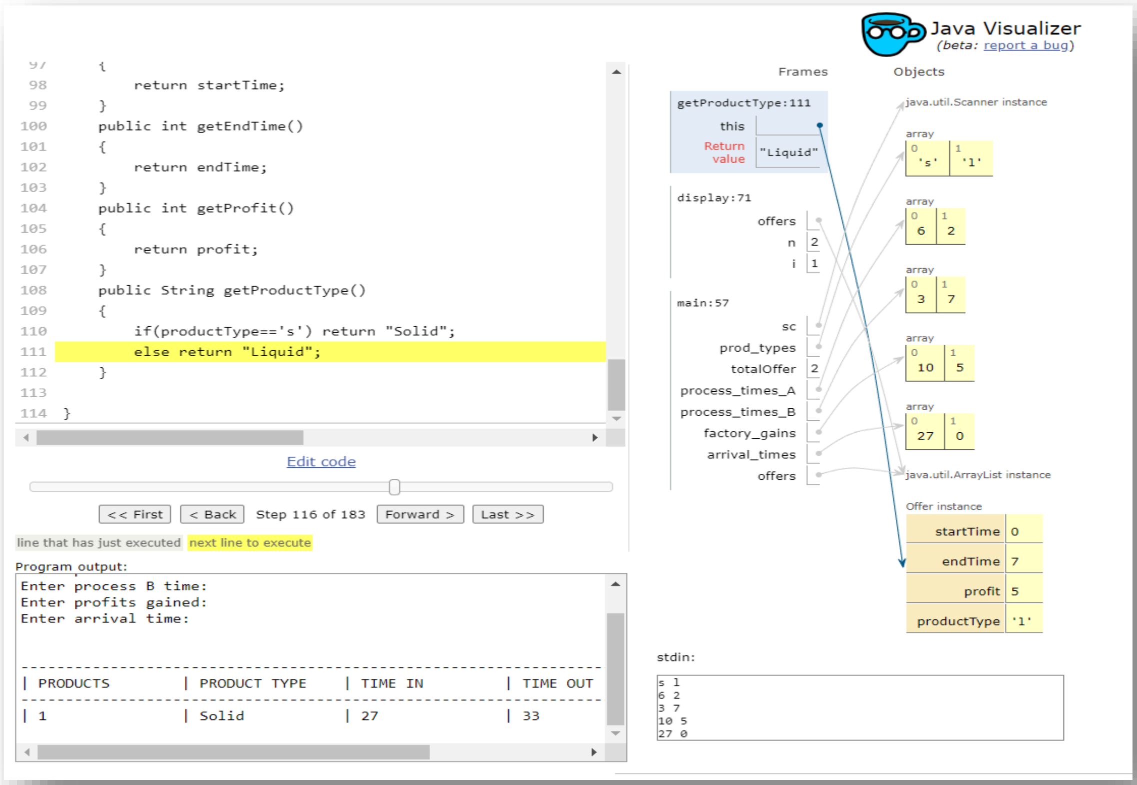
```
java.util.Scanner instance
array  0: 's'  1: '1'
array  0: 6    1: 2
array  0: 3    1: 7
array  0: 10   1: 5
array  0: 27   1: 0
java.util.ArrayList instance
Offer instance
  startTime    27
  endTime      33
  profit       10
  productType  's'
```

Program output:

```
Enter process A time:
Enter process B time:
Enter profits gained:
Enter arrival time:

------------------------------------------------------------------------
| PRODUCTS          | PRODUCT TYPE      | TIME IN           | TIME OUT
------------------------------------------------------------------------
```

stdin:

```
s 1
6 2
3 7
10 5
27 0
```



```
    {
        return startTime;
    }
    public int getEndTime()
    {
        return endTime;
    }
    public int getProfit()
    {
        return profit;
    }
    public String getProductType()
    {
        if(productType=='s') return "Solid";
        else return "Liquid";
    }

}
```

Program output:

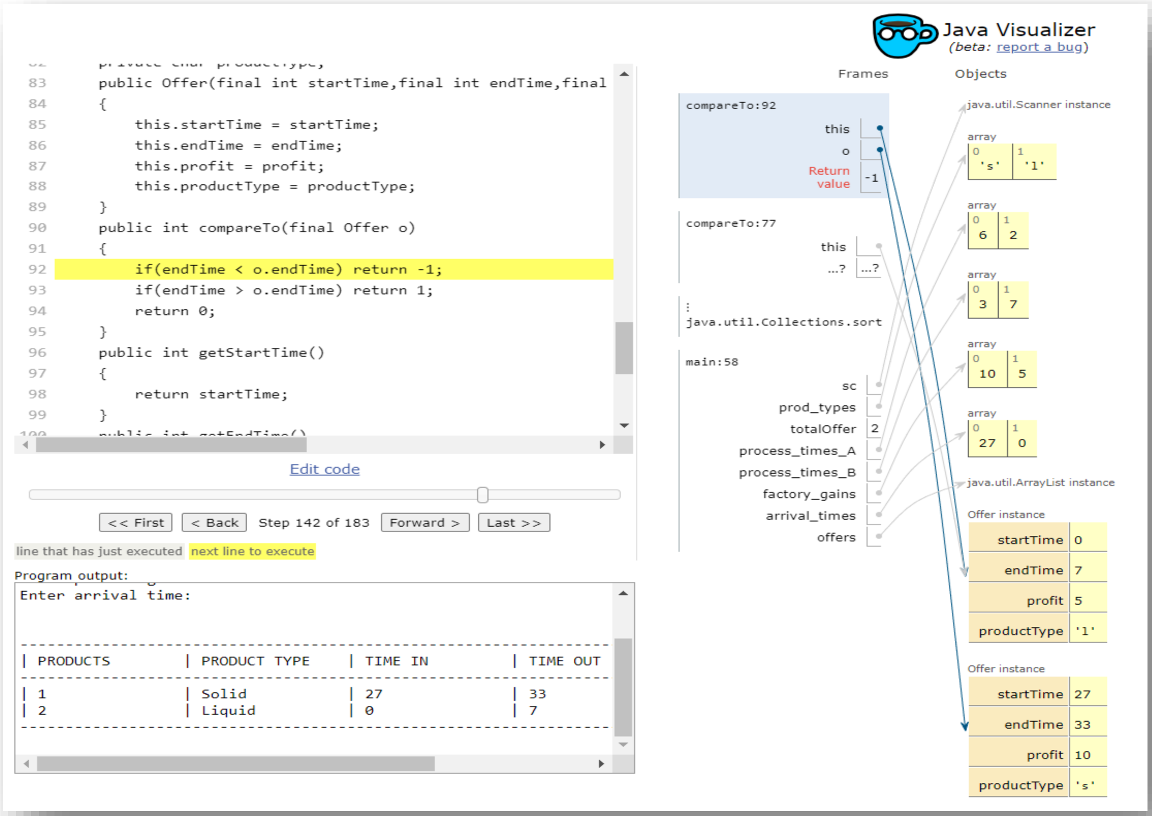
```
Enter process B time:
Enter profits gained:
Enter arrival time:

-------------------------------------------------------------------------
| PRODUCTS            | PRODUCT TYPE        | TIME IN             | TIME OUT
-------------------------------------------------------------------------
| 1                   | Solid               | 27                  | 33
```

```
getProductType:111
          this
  Return value  "Liquid"

display:71
        offers
             n  2
             i  1

main:57
            sc
    prod_types
    totalOffer  2
process_times_A
process_times_B
  factory_gains
 arrival_times
        offers
```

```
java.util.Scanner instance
array  0 's'  1 '1'
array  0 6    1 2
array  0 3    1 7
array  0 10   1 5
array  0 27   1 0
java.util.ArrayList instance
Offer instance
  startTime    0
  endTime      7
  profit       5
  productType  '1'
```

stdin:

```
s 1
6 2
3 7
10 5
27 0
```



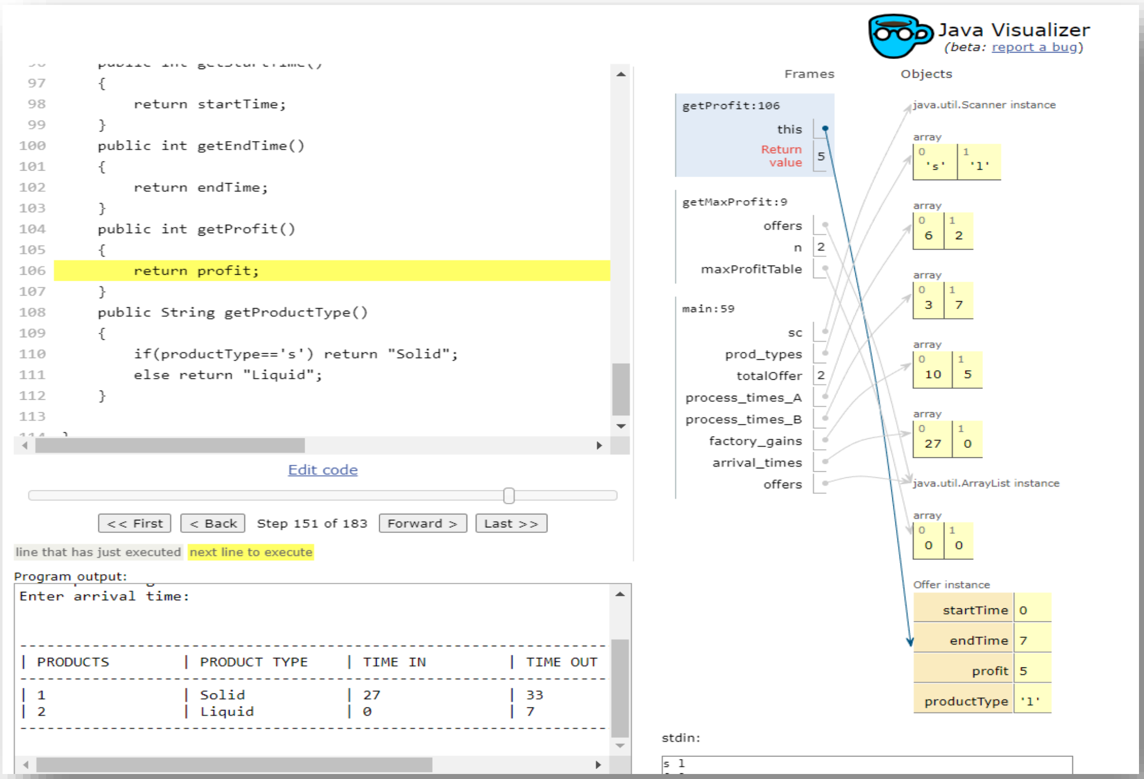
```
    private char productType;
    public Offer(final int startTime,final int endTime,final
    {
        this.startTime = startTime;
        this.endTime = endTime;
        this.profit = profit;
        this.productType = productType;
    }
    public int compareTo(final Offer o)
    {
        if(endTime < o.endTime) return -1;
        if(endTime > o.endTime) return 1;
        return 0;
    }
    public int getStartTime()
    {
        return startTime;
    }
    public int getEndTime()
```

Program output:

```
Enter arrival time:

-------------------------------------------------------------------------
| PRODUCTS            | PRODUCT TYPE        | TIME IN             | TIME OUT
-------------------------------------------------------------------------
| 1                   | Solid               | 27                  | 33
| 2                   | Liquid              | 0                   | 7
-------------------------------------------------------------------------
```

```
compareTo:92
          this
             o
  Return value  -1

compareTo:77
          this
           ...?  ...?
:
java.util.Collections.sort

main:58
            sc
    prod_types
    totalOffer  2
process_times_A
process_times_B
  factory_gains
 arrival_times
        offers
```

```
java.util.Scanner instance
array  0 's'  1 '1'
array  0 6    1 2
array  0 3    1 7
array  0 10   1 5
array  0 27   1 0
java.util.ArrayList instance
Offer instance
  startTime    0
  endTime      7
  profit       5
  productType  '1'
Offer instance
  startTime    27
  endTime      33
  profit       10
  productType  's'
```



```
    {
        return startTime;
    }
    public int getEndTime()
    {
        return endTime;
    }
    public int getProfit()
    {
        return profit;
    }
    public String getProductType()
    {
        if(productType=='s') return "Solid";
        else return "Liquid";
    }

```

Program output:

```
Enter arrival time:

--------------------------------------------------------------------
| PRODUCTS           | PRODUCT TYPE       | TIME IN            | TIME OUT
--------------------------------------------------------------------
| 1                  | Solid              | 27                 | 33
| 2                  | Liquid             | 0                  | 7
--------------------------------------------------------------------
```

getProfit:106
this
Return value 5

getMaxProfit:9
offers
n 2
maxProfitTable

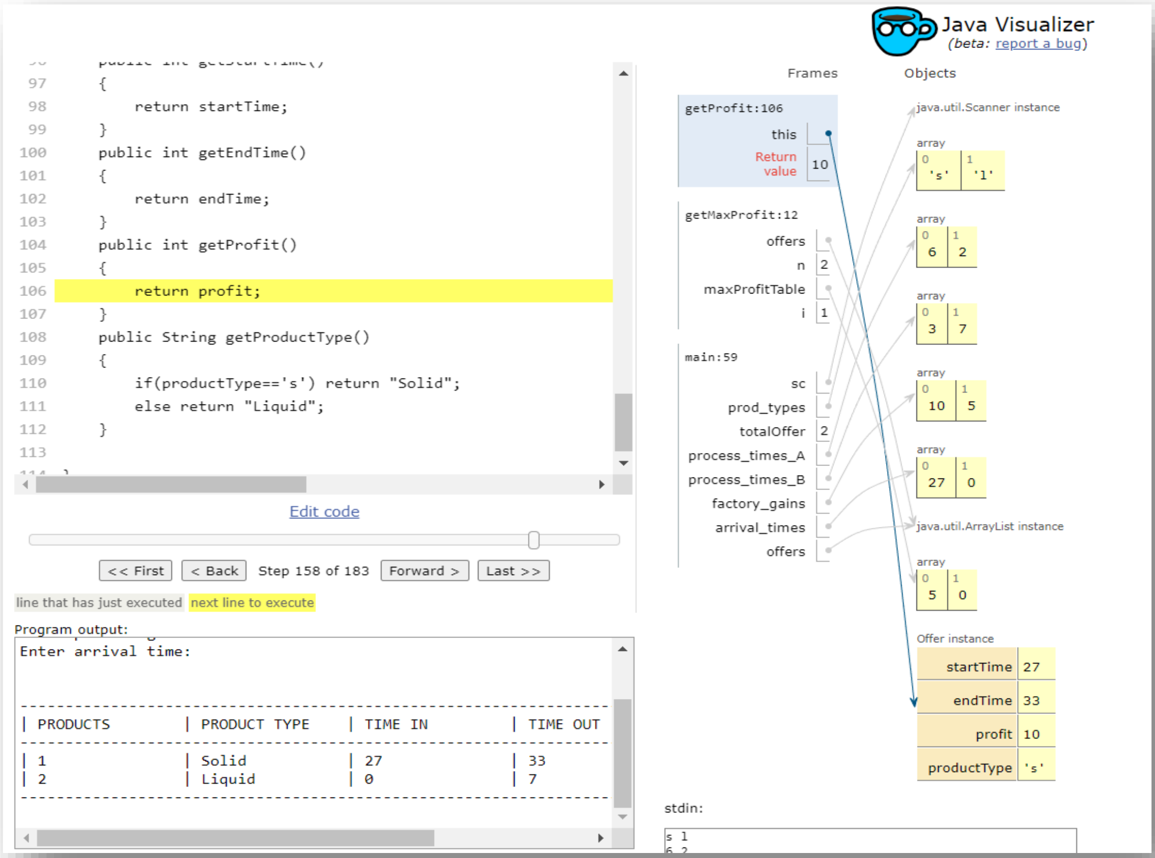
main:59
sc
prod_types
totalOffer 2
process_times_A
process_times_B
factory_gains
arrival_times
offers

java.util.Scanner instance
array 0 's' 1 '1'
array 0 6 1 2
array 0 3 1 7
array 0 10 1 5
array 0 27 1 0
java.util.ArrayList instance
array 0 0 1 0

Offer instance

startTime	0
endTime	7
profit	5
productType	'1'

```
s 1
```



```
    {
        return startTime;
    }
    public int getEndTime()
    {
        return endTime;
    }
    public int getProfit()
    {
        return profit;
    }
    public String getProductType()
    {
        if(productType=='s') return "Solid";
        else return "Liquid";
    }

```

Program output:

```
Enter arrival time:

--------------------------------------------------------------------
| PRODUCTS           | PRODUCT TYPE       | TIME IN            | TIME OUT
--------------------------------------------------------------------
| 1                  | Solid              | 27                 | 33
| 2                  | Liquid             | 0                  | 7
--------------------------------------------------------------------
```

getProfit:106
this
Return value 10

getMaxProfit:12
offers
n 2
maxProfitTable
i 1

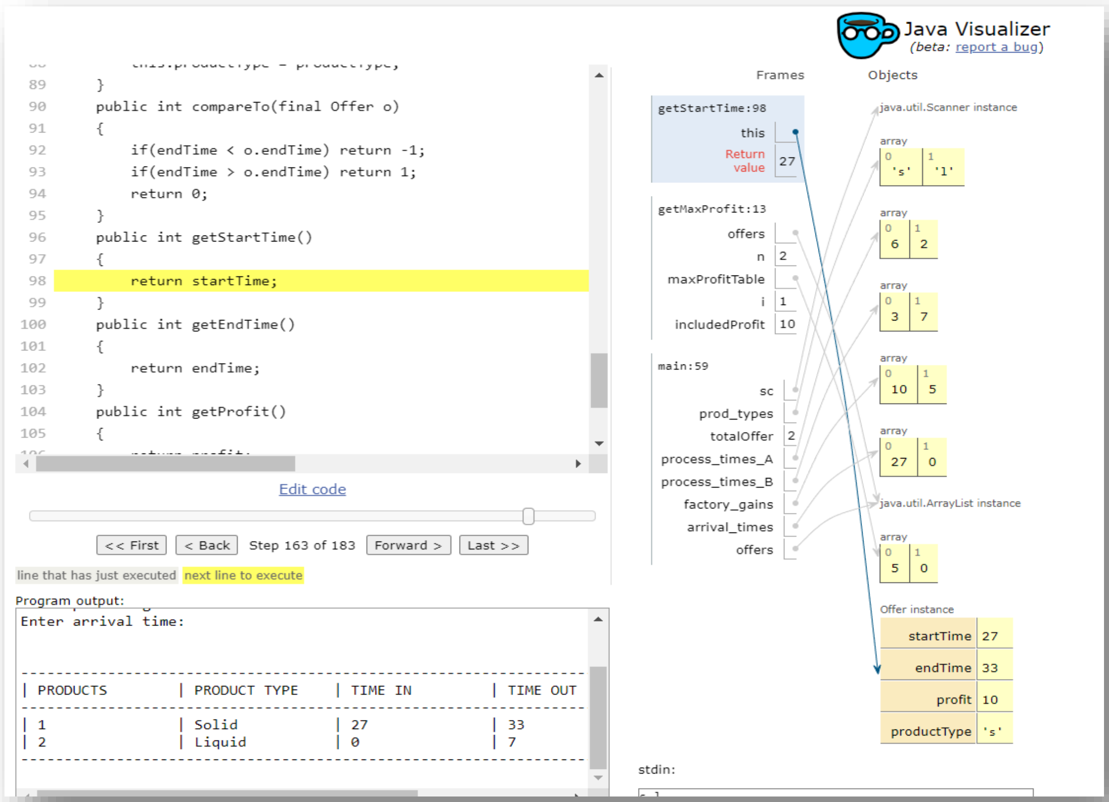
main:59
sc
prod_types
totalOffer 2
process_times_A
process_times_B
factory_gains
arrival_times
offers

java.util.Scanner instance
array 0 's' 1 '1'
array 0 6 1 2
array 0 3 1 7
array 0 10 1 5
array 0 27 1 0
java.util.ArrayList instance
array 0 5 1 0

Offer instance

startTime	27
endTime	33
profit	10
productType	's'

```
s 1
```



```
    }
    public int compareTo(final Offer o)
    {
        if(endTime < o.endTime) return -1;
        if(endTime > o.endTime) return 1;
        return 0;
    }
    public int getStartTime()
    {
        return startTime;
    }
    public int getEndTime()
    {
        return endTime;
    }
    public int getProfit()
    {
```

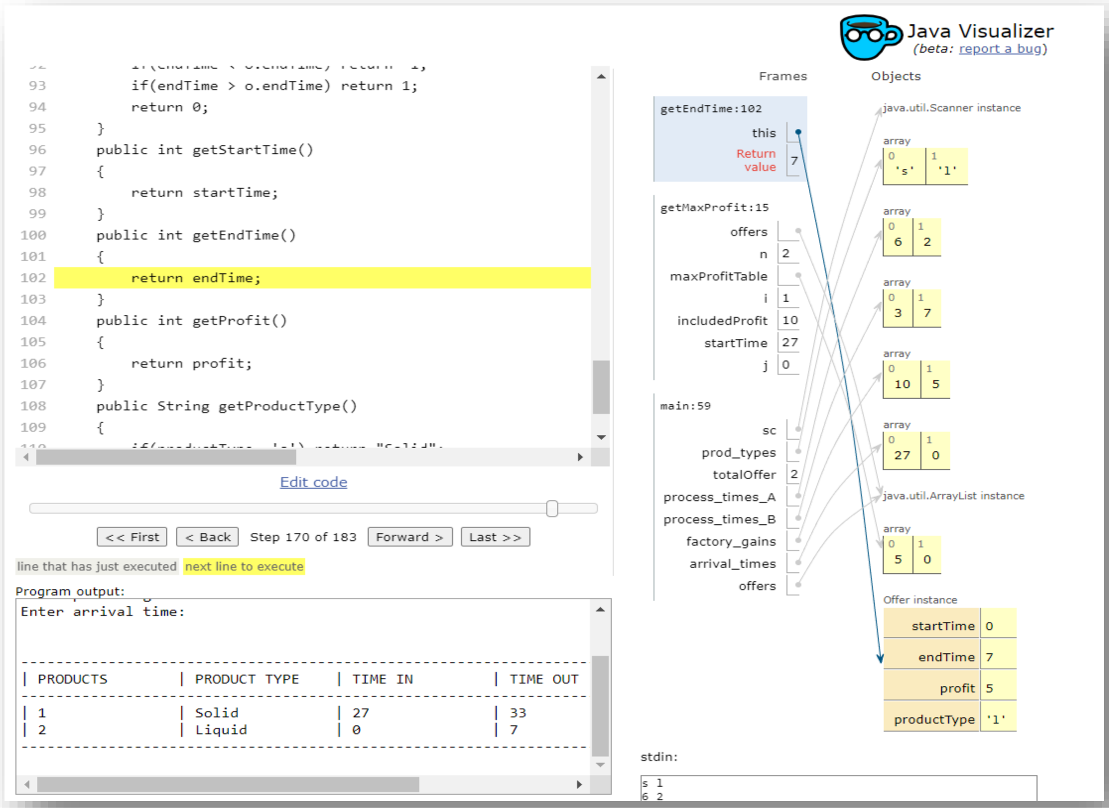
Program output:

```
Enter arrival time:

-------------------------------------------------------------
| PRODUCTS          | PRODUCT TYPE      | TIME IN          | TIME OUT
-------------------------------------------------------------
| 1                 | Solid             | 27               | 33
| 2                 | Liquid            | 0                | 7
-------------------------------------------------------------
```

```
getStartTime:98
         this
 Return value  27

getMaxProfit:13
        offers
             n  2
maxProfitTable
             i  1
 includedProfit 10

main:59
             sc
     prod_types
     totalOffer  2
process_times_A
process_times_B
  factory_gains
  arrival_times
         offers
```

```
java.util.Scanner instance
array  0: 's'   1: '1'
array  0: 6     1: 2
array  0: 3     1: 7
array  0: 10    1: 5
array  0: 27    1: 0
java.util.ArrayList instance
array  0: 5     1: 0
Offer instance
  startTime  27
    endTime  33
     profit  10
productType  's'
```

stdin:



```
        if(endTime > o.endTime) return 1;
        return 0;
    }
    public int getStartTime()
    {
        return startTime;
    }
    public int getEndTime()
    {
        return endTime;
    }
    public int getProfit()
    {
        return profit;
    }
    public String getProductType()
    {
```

Program output:

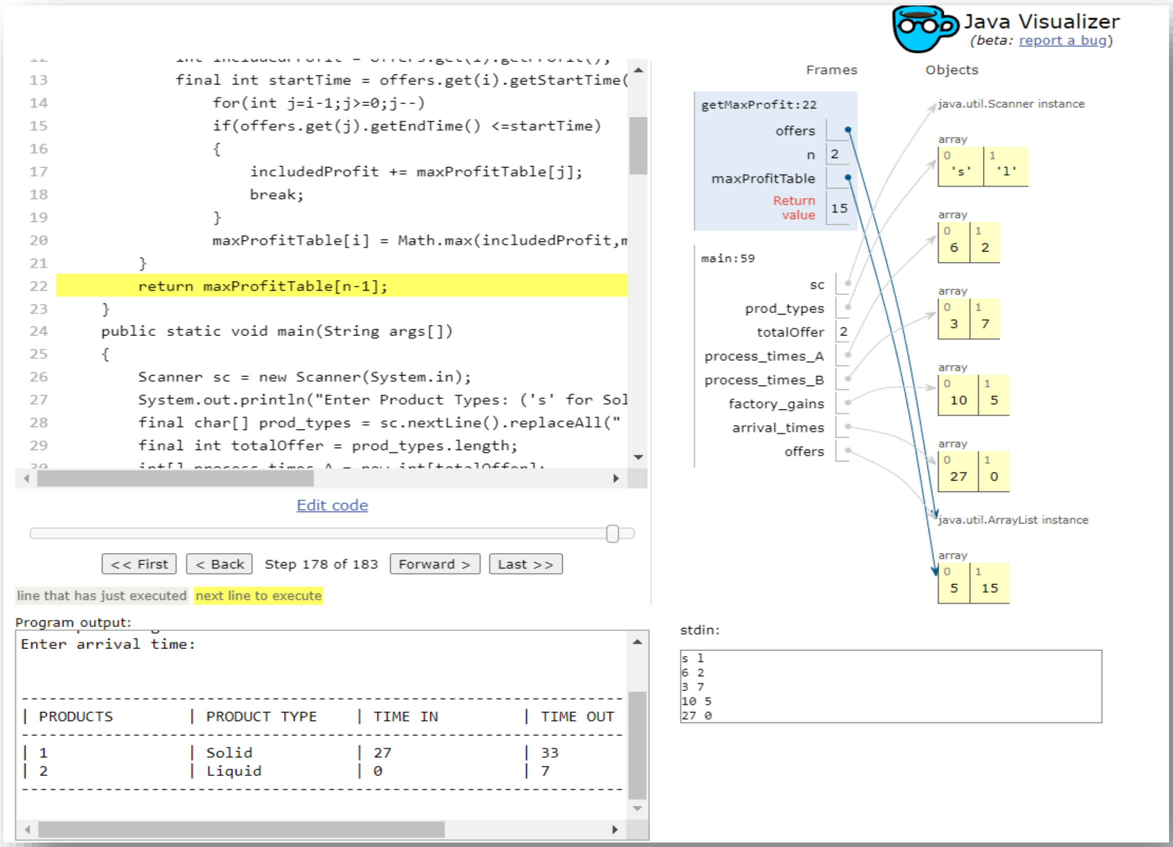
```
Enter arrival time:

-------------------------------------------------------------
| PRODUCTS          | PRODUCT TYPE      | TIME IN          | TIME OUT
-------------------------------------------------------------
| 1                 | Solid             | 27               | 33
| 2                 | Liquid            | 0                | 7
-------------------------------------------------------------
```

```
getEndTime:102
         this
 Return value  7

getMaxProfit:15
        offers
             n  2
maxProfitTable
             i  1
 includedProfit 10
      startTime 27
             j  0

main:59
             sc
     prod_types
     totalOffer  2
process_times_A
process_times_B
  factory_gains
  arrival_times
         offers
```

```
java.util.Scanner instance
array  0: 's'   1: '1'
array  0: 6     1: 2
array  0: 3     1: 7
array  0: 10    1: 5
array  0: 27    1: 0
java.util.ArrayList instance
array  0: 5     1: 0
Offer instance
  startTime  0
    endTime  7
     profit  5
productType  '1'
```

stdin:

```
s 1
6 2
```



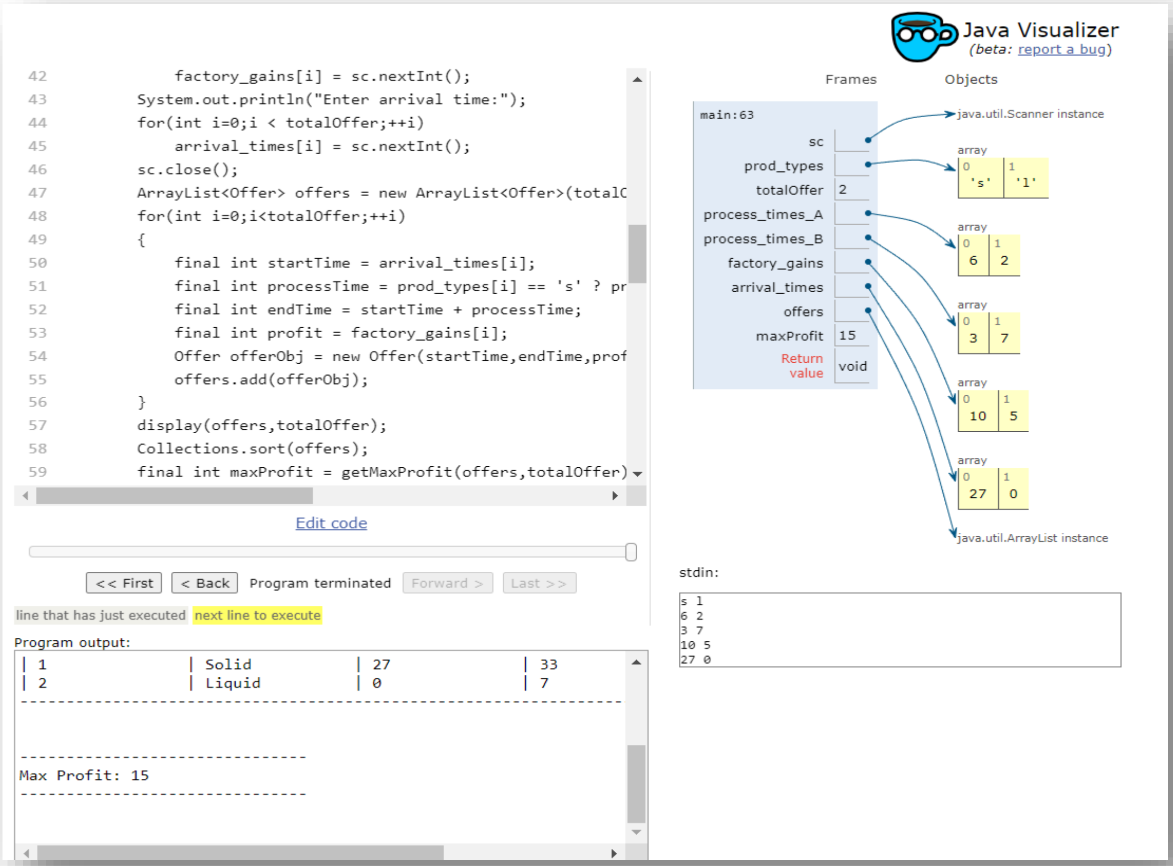
```java
            int includedProfit = offers.get(i).getProfit();
            final int startTime = offers.get(i).getStartTime(
                for(int j=i-1;j>=0;j--)
                if(offers.get(j).getEndTime() <=startTime)
                {
                    includedProfit += maxProfitTable[j];
                    break;
                }
                maxProfitTable[i] = Math.max(includedProfit,m
        }
        return maxProfitTable[n-1];
    }
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Product Types: ('s' for Sol
        final char[] prod_types = sc.nextLine().replaceAll("
        final int totalOffer = prod_types.length;
        int[] process_times_A = new int[totalOffer];
```

Program output:

```
Enter arrival time:


-----------------------------------------------------------------
| PRODUCTS          | PRODUCT TYPE      | TIME IN           | TIME OUT
-----------------------------------------------------------------
| 1                 | Solid             | 27                | 33
| 2                 | Liquid            | 0                 | 7
-----------------------------------------------------------------
```

stdin:

```
s 1
6 2
3 7
10 5
27 0
```



```java
            factory_gains[i] = sc.nextInt();
        System.out.println("Enter arrival time:");
        for(int i=0;i < totalOffer;++i)
            arrival_times[i] = sc.nextInt();
        sc.close();
        ArrayList<Offer> offers = new ArrayList<Offer>(totalO
        for(int i=0;i<totalOffer;++i)
        {
            final int startTime = arrival_times[i];
            final int processTime = prod_types[i] == 's' ? pr
            final int endTime = startTime + processTime;
            final int profit = factory_gains[i];
            Offer offerObj = new Offer(startTime,endTime,prof
            offers.add(offerObj);
        }
        display(offers,totalOffer);
        Collections.sort(offers);
        final int maxProfit = getMaxProfit(offers,totalOffer)
```

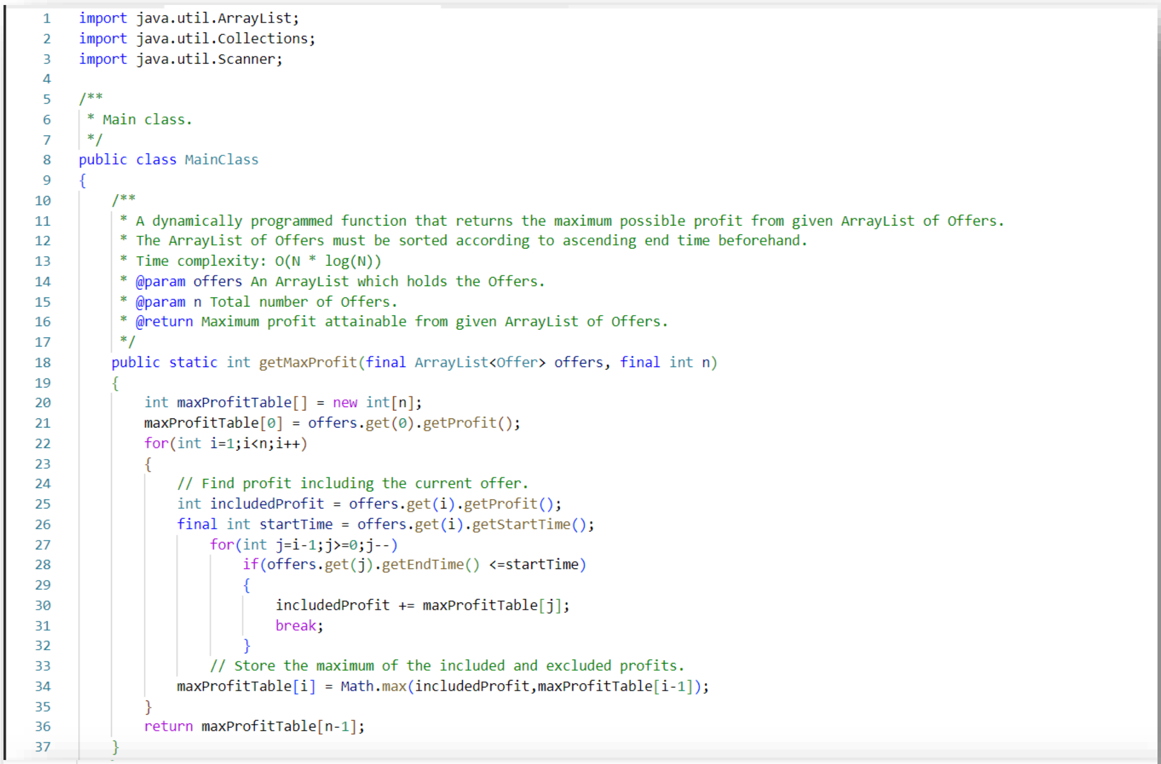
Program output:

```
| 1                 | Solid             | 27                | 33
| 2                 | Liquid            | 0                 | 7
-----------------------------------------------------------------



------------------------------
Max Profit: 15
------------------------------
```

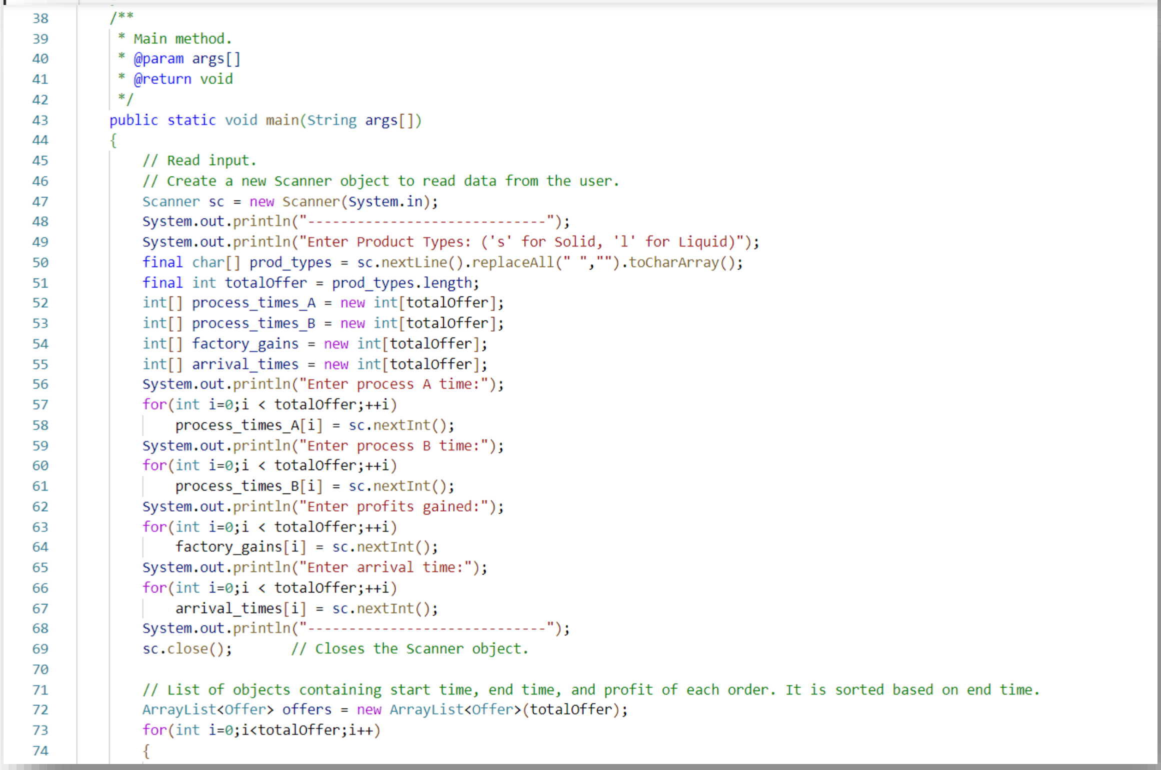
stdin:

```
s 1
6 2
3 7
10 5
27 0
```

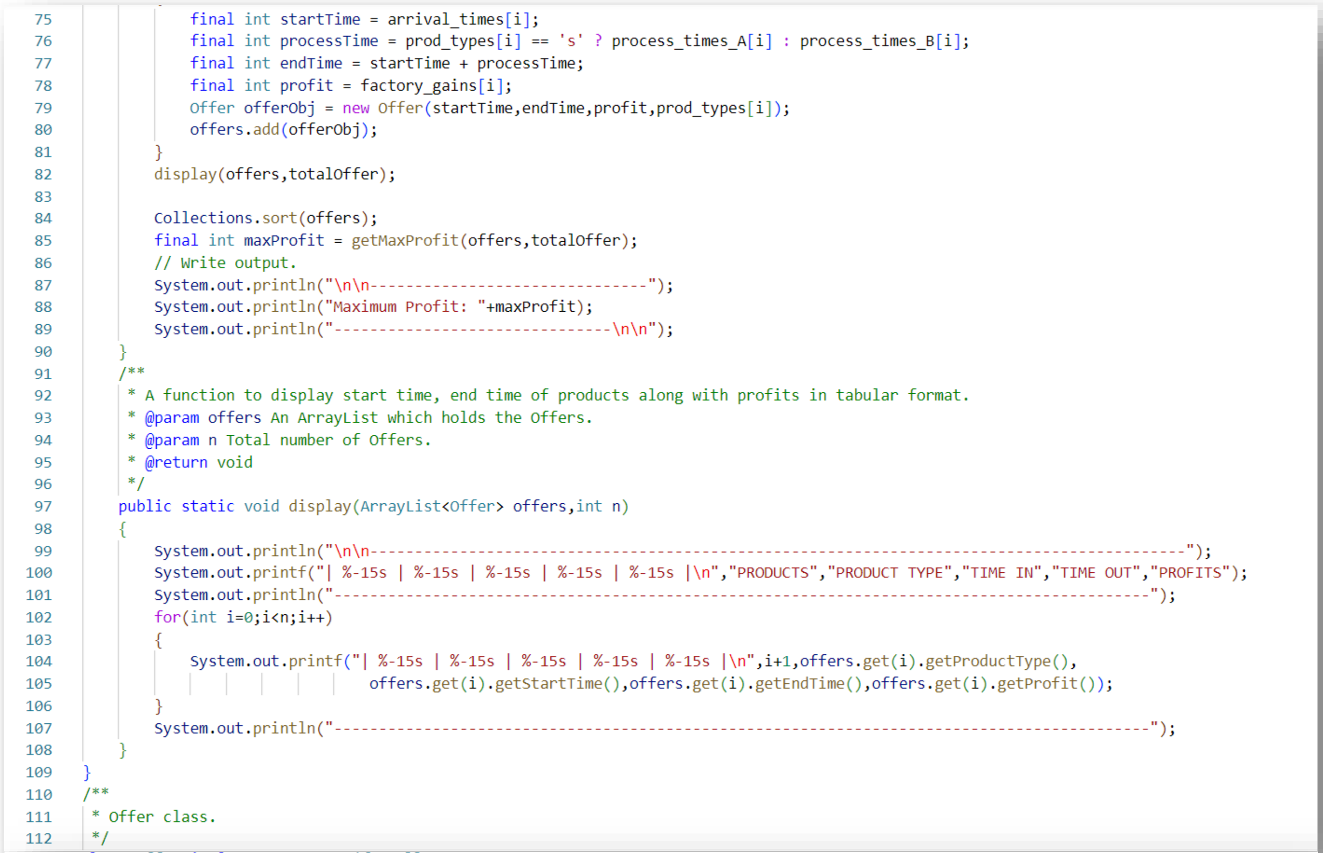
7. CODE



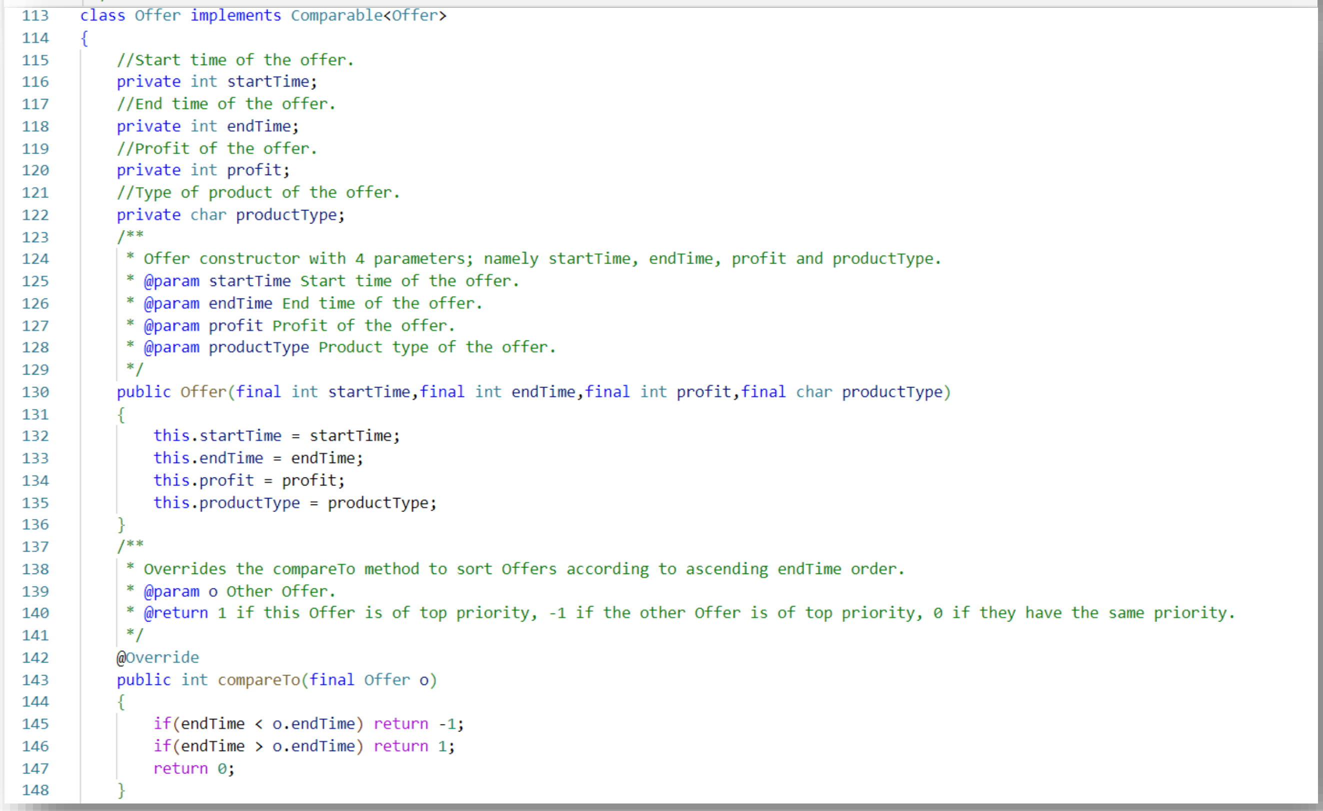
```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

/**
 * Main class.
 */
public class MainClass
{
    /**
     * A dynamically programmed function that returns the maximum possible profit from given ArrayList of Offers.
     * The ArrayList of Offers must be sorted according to ascending end time beforehand.
     * Time complexity: O(N * log(N))
     * @param offers An ArrayList which holds the Offers.
     * @param n Total number of Offers.
     * @return Maximum profit attainable from given ArrayList of Offers.
     */
    public static int getMaxProfit(final ArrayList<Offer> offers, final int n)
    {
        int maxProfitTable[] = new int[n];
        maxProfitTable[0] = offers.get(0).getProfit();
        for(int i=1;i<n;i++)
        {
            // Find profit including the current offer.
            int includedProfit = offers.get(i).getProfit();
            final int startTime = offers.get(i).getStartTime();
                for(int j=i-1;j>=0;j--)
                    if(offers.get(j).getEndTime() <=startTime)
                    {
                        includedProfit += maxProfitTable[j];
                        break;
                    }
                // Store the maximum of the included and excluded profits.
            maxProfitTable[i] = Math.max(includedProfit,maxProfitTable[i-1]);
        }
        return maxProfitTable[n-1];
    }
```



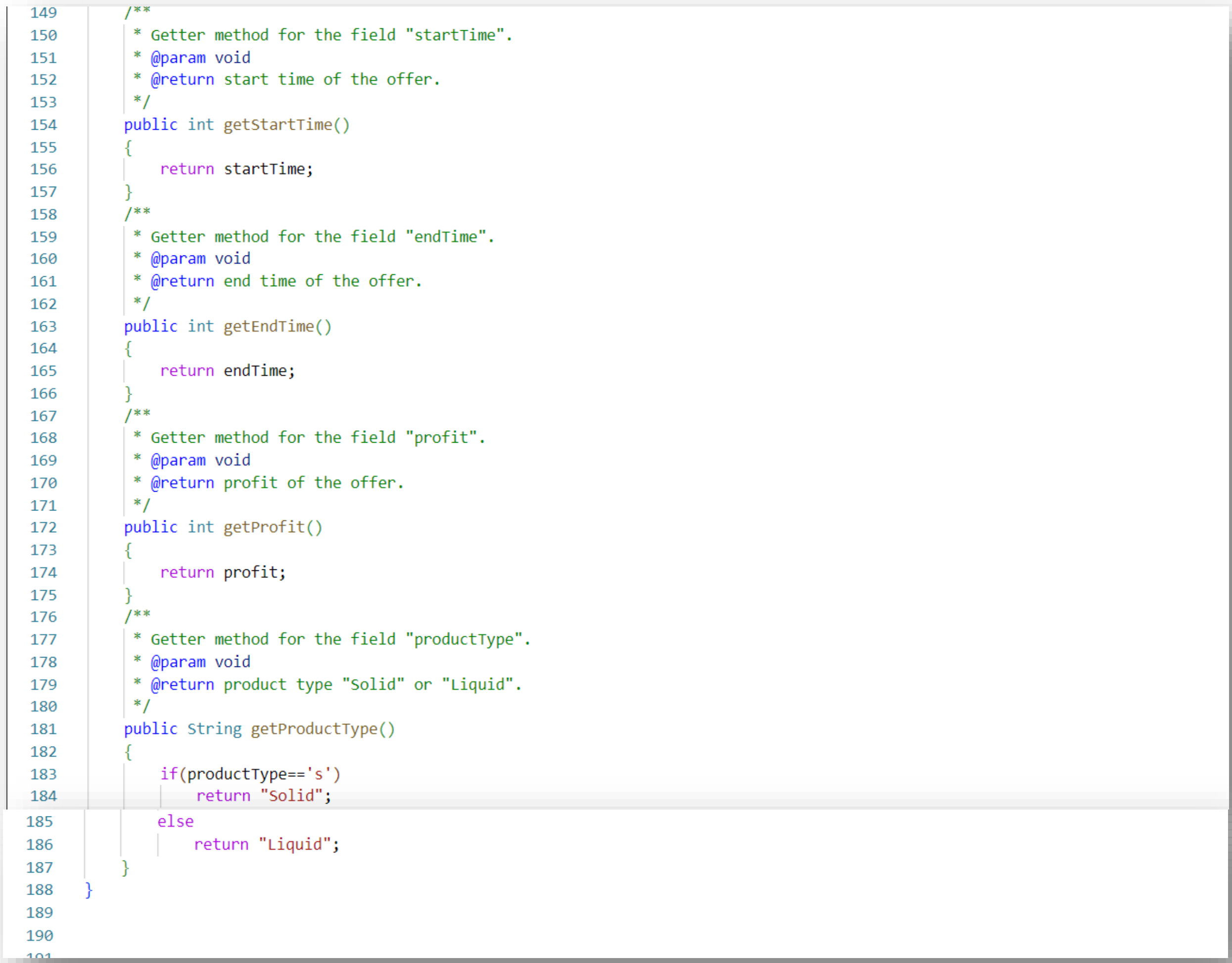
```java
    /**
     * Main method.
     * @param args[]
     * @return void
     */
    public static void main(String args[])
    {
        // Read input.
        // Create a new Scanner object to read data from the user.
        Scanner sc = new Scanner(System.in);
        System.out.println("----------------------------");
        System.out.println("Enter Product Types: ('s' for Solid, 'l' for Liquid)");
        final char[] prod_types = sc.nextLine().replaceAll(" ","").toCharArray();
        final int totalOffer = prod_types.length;
        int[] process_times_A = new int[totalOffer];
        int[] process_times_B = new int[totalOffer];
        int[] factory_gains = new int[totalOffer];
        int[] arrival_times = new int[totalOffer];
        System.out.println("Enter process A time:");
        for(int i=0;i < totalOffer;++i)
            process_times_A[i] = sc.nextInt();
        System.out.println("Enter process B time:");
        for(int i=0;i < totalOffer;++i)
            process_times_B[i] = sc.nextInt();
        System.out.println("Enter profits gained:");
        for(int i=0;i < totalOffer;++i)
            factory_gains[i] = sc.nextInt();
        System.out.println("Enter arrival time:");
        for(int i=0;i < totalOffer;++i)
            arrival_times[i] = sc.nextInt();
        System.out.println("----------------------------");
        sc.close();       // Closes the Scanner object.

        // List of objects containing start time, end time, and profit of each order. It is sorted based on end time.
        ArrayList<Offer> offers = new ArrayList<Offer>(totalOffer);
        for(int i=0;i<totalOffer;i++)
        {
```



```java
            final int startTime = arrival_times[i];
            final int processTime = prod_types[i] == 's' ? process_times_A[i] : process_times_B[i];
            final int endTime = startTime + processTime;
            final int profit = factory_gains[i];
            Offer offerObj = new Offer(startTime,endTime,profit,prod_types[i]);
            offers.add(offerObj);
        }
        display(offers,totalOffer);

        Collections.sort(offers);
        final int maxProfit = getMaxProfit(offers,totalOffer);
        // Write output.
        System.out.println("\n\n------------------------------");
        System.out.println("Maximum Profit: "+maxProfit);
        System.out.println("------------------------------\n\n");
    }
    /**
     * A function to display start time, end time of products along with profits in tabular format.
     * @param offers An ArrayList which holds the Offers.
     * @param n Total number of Offers.
     * @return void
     */
    public static void display(ArrayList<Offer> offers,int n)
    {
        System.out.println("\n\n------------------------------------------------------------------------------------------");
        System.out.printf("| %-15s | %-15s | %-15s | %-15s | %-15s |\n","PRODUCTS","PRODUCT TYPE","TIME IN","TIME OUT","PROFITS");
        System.out.println("------------------------------------------------------------------------------------------");
        for(int i=0;i<n;i++)
        {
            System.out.printf("| %-15s | %-15s | %-15s | %-15s | %-15s |\n",i+1,offers.get(i).getProductType(),
                                    offers.get(i).getStartTime(),offers.get(i).getEndTime(),offers.get(i).getProfit());
        }
        System.out.println("------------------------------------------------------------------------------------------");
    }
}
/**
 * Offer class.
 */
```

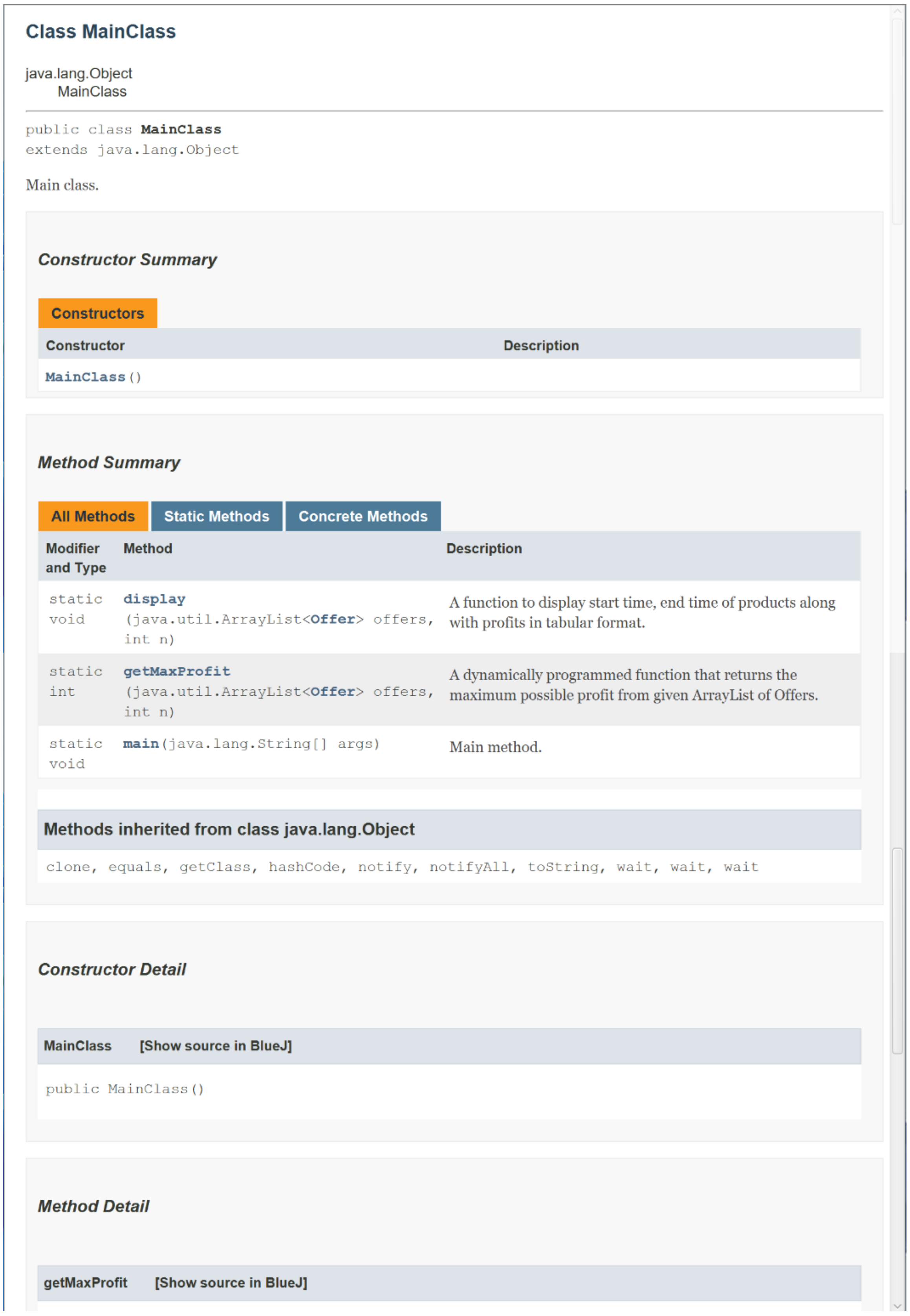


```java
class Offer implements Comparable<Offer>
{
    //Start time of the offer.
    private int startTime;
    //End time of the offer.
    private int endTime;
    //Profit of the offer.
    private int profit;
    //Type of product of the offer.
    private char productType;
    /**
     * Offer constructor with 4 parameters; namely startTime, endTime, profit and productType.
     * @param startTime Start time of the offer.
     * @param endTime End time of the offer.
     * @param profit Profit of the offer.
     * @param productType Product type of the offer.
     */
    public Offer(final int startTime,final int endTime,final int profit,final char productType)
    {
        this.startTime = startTime;
        this.endTime = endTime;
        this.profit = profit;
        this.productType = productType;
    }
    /**
     * Overrides the compareTo method to sort Offers according to ascending endTime order.
     * @param o Other Offer.
     * @return 1 if this Offer is of top priority, -1 if the other Offer is of top priority, 0 if they have the same priority.
     */
    @Override
    public int compareTo(final Offer o)
    {
        if(endTime < o.endTime) return -1;
        if(endTime > o.endTime) return 1;
        return 0;
    }
```



```java
    /**
     * Getter method for the field "startTime".
     * @param void
     * @return start time of the offer.
     */
    public int getStartTime()
    {
        return startTime;
    }
    /**
     * Getter method for the field "endTime".
     * @param void
     * @return end time of the offer.
     */
    public int getEndTime()
    {
        return endTime;
    }
    /**
     * Getter method for the field "profit".
     * @param void
     * @return profit of the offer.
     */
    public int getProfit()
    {
        return profit;
    }
    /**
     * Getter method for the field "productType".
     * @param void
     * @return product type "Solid" or "Liquid".
     */
    public String getProductType()
    {
        if(productType=='s')
            return "Solid";
        else
            return "Liquid";
    }
}
```

8. PROGRAM DOCUMENTATION



Class MainClass

java.lang.Object
    MainClass

```
public class MainClass
extends java.lang.Object
```

Main class.

Constructor Summary

Constructors

Constructor	Description
`MainClass()`	

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
static void	`display(java.util.ArrayList<Offer> offers, int n)`	A function to display start time, end time of products along with profits in tabular format.
static int	`getMaxProfit(java.util.ArrayList<Offer> offers, int n)`	A dynamically programmed function that returns the maximum possible profit from given ArrayList of Offers.
static void	`main(java.lang.String[] args)`	Main method.

Methods inherited from class java.lang.Object

`clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

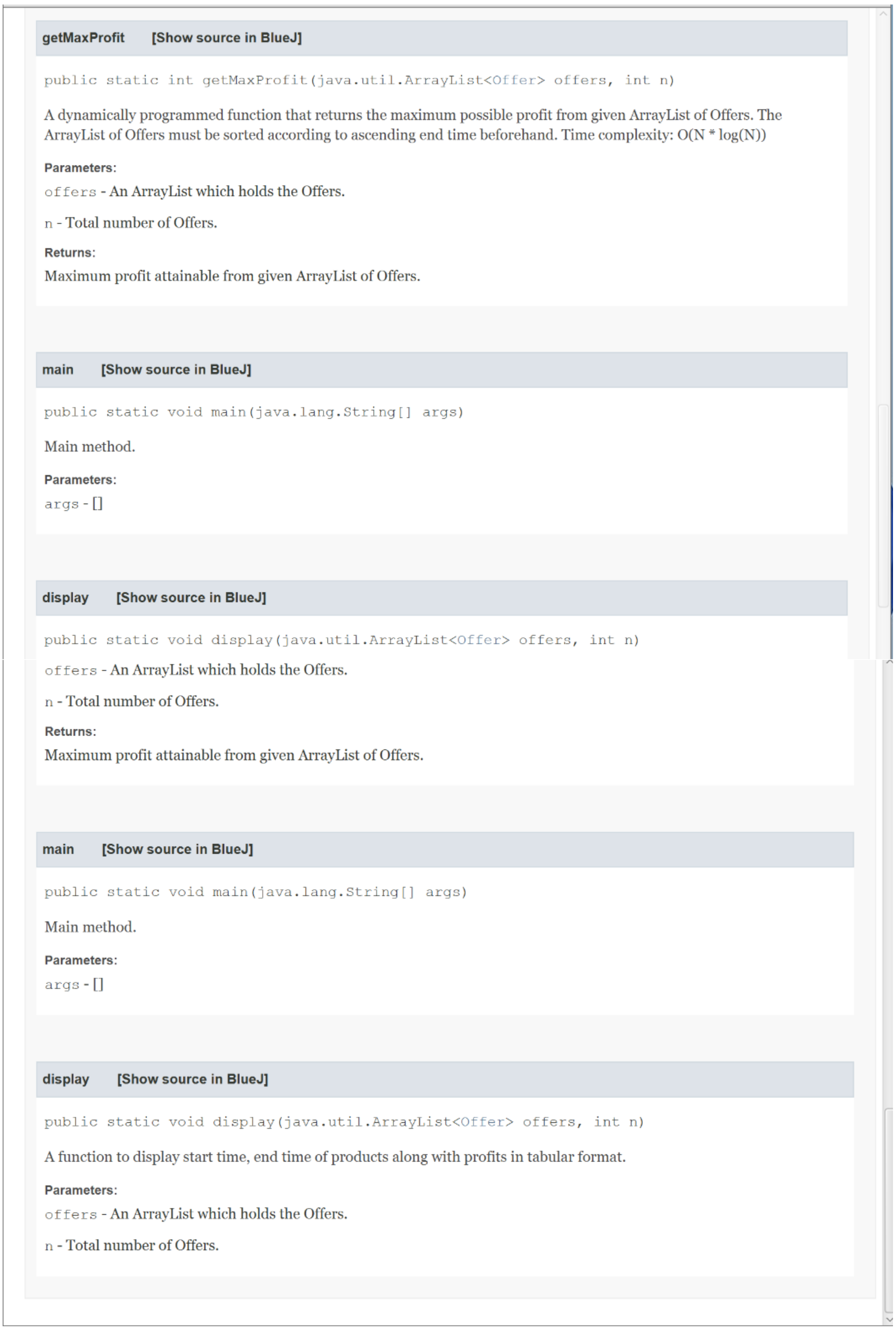
Constructor Detail

MainClass [Show source in BlueJ]

```
public MainClass()
```

Method Detail

getMaxProfit [Show source in BlueJ]



getMaxProfit **[Show source in BlueJ]**

```
public static int getMaxProfit(java.util.ArrayList<Offer> offers, int n)
```

A dynamically programmed function that returns the maximum possible profit from given ArrayList of Offers. The ArrayList of Offers must be sorted according to ascending end time beforehand. Time complexity: O(N * log(N))

Parameters:

`offers` - An ArrayList which holds the Offers.

`n` - Total number of Offers.

Returns:

Maximum profit attainable from given ArrayList of Offers.

main **[Show source in BlueJ]**

```
public static void main(java.lang.String[] args)
```

Main method.

Parameters:

`args` - []

display **[Show source in BlueJ]**

```
public static void display(java.util.ArrayList<Offer> offers, int n)
```

`offers` - An ArrayList which holds the Offers.

`n` - Total number of Offers.

Returns:

Maximum profit attainable from given ArrayList of Offers.

main **[Show source in BlueJ]**

```
public static void main(java.lang.String[] args)
```

Main method.

Parameters:

`args` - []

display **[Show source in BlueJ]**

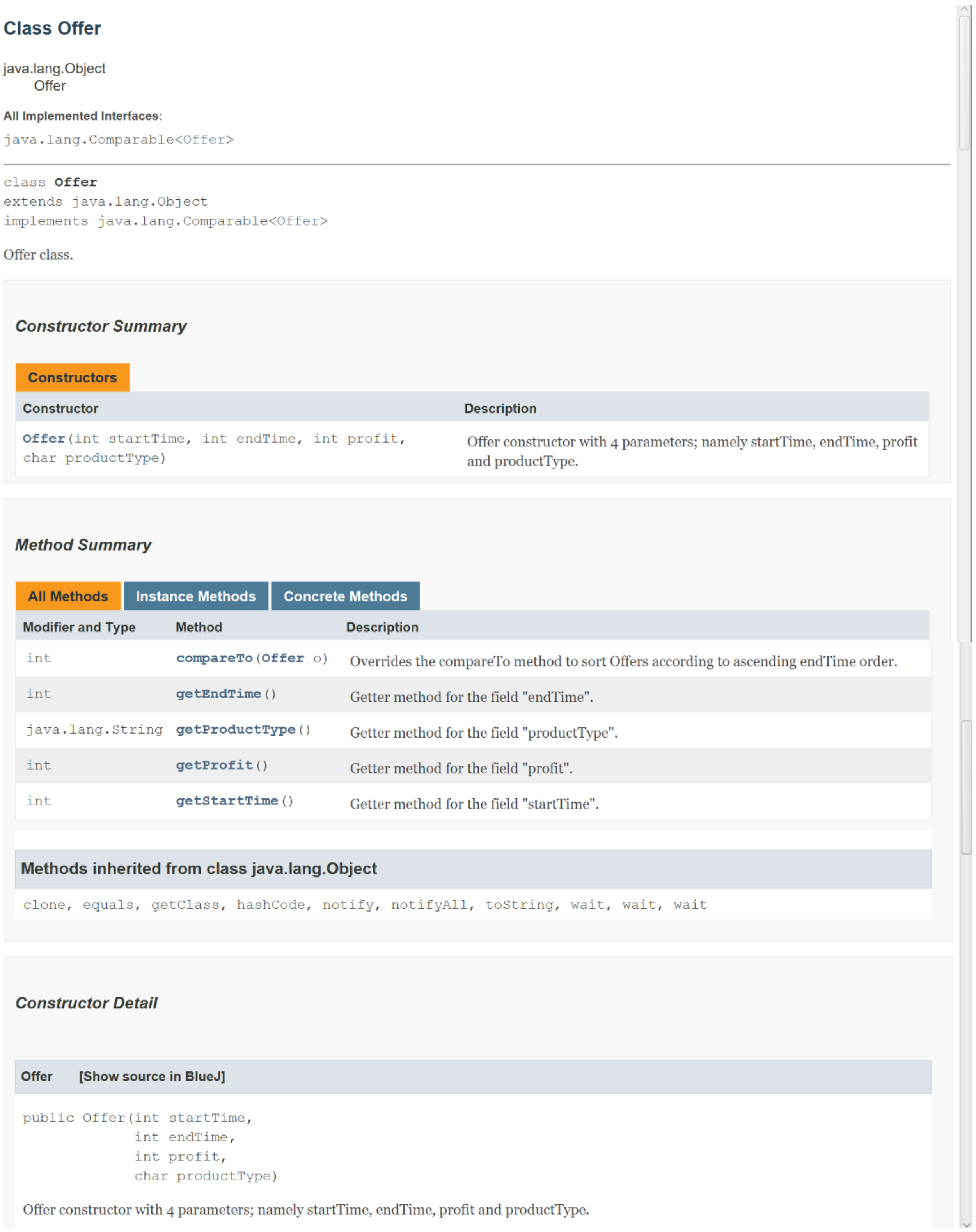
```
public static void display(java.util.ArrayList<Offer> offers, int n)
```

A function to display start time, end time of products along with profits in tabular format.

Parameters:

`offers` - An ArrayList which holds the Offers.

`n` - Total number of Offers.



Class Offer

java.lang.Object
    Offer

All Implemented Interfaces:

```
java.lang.Comparable<Offer>
```

```
class Offer
extends java.lang.Object
implements java.lang.Comparable<Offer>
```

Offer class.

Constructor Summary

Constructors

Constructor	Description
`Offer(int startTime, int endTime, int profit, char productType)`	Offer constructor with 4 parameters; namely startTime, endTime, profit and productType.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`int`	`compareTo(Offer o)`	Overrides the compareTo method to sort Offers according to ascending endTime order.
`int`	`getEndTime()`	Getter method for the field "endTime".
`java.lang.String`	`getProductType()`	Getter method for the field "productType".
`int`	`getProfit()`	Getter method for the field "profit".
`int`	`getStartTime()`	Getter method for the field "startTime".

Methods inherited from class java.lang.Object

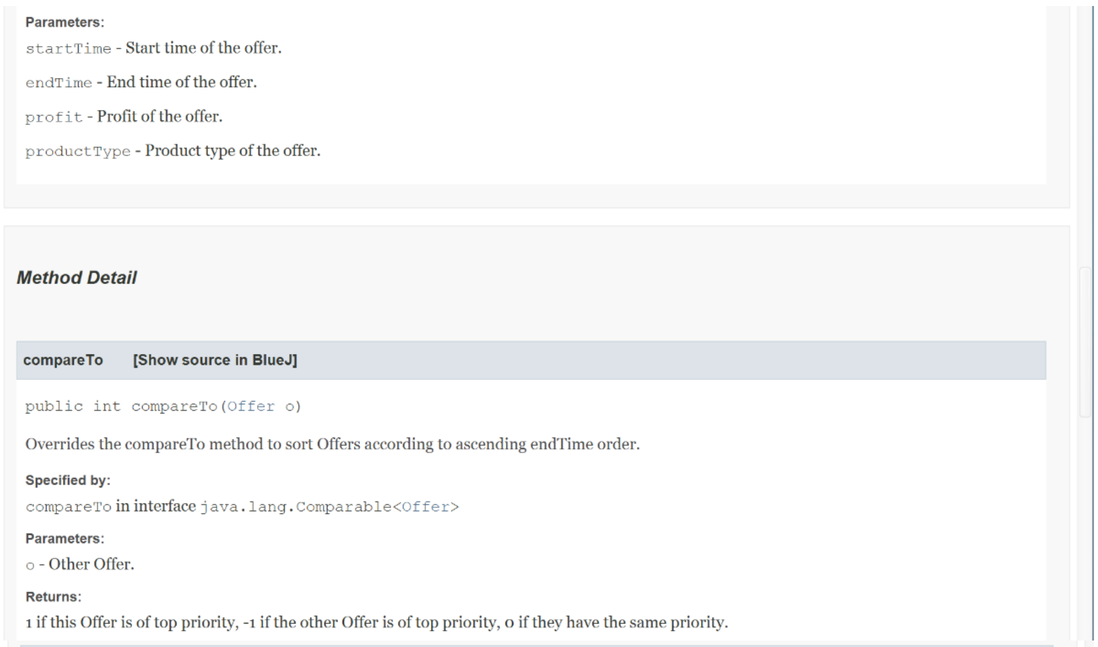
```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

Offer   **[Show source in BlueJ]**

```
public Offer(int startTime,
             int endTime,
             int profit,
             char productType)
```

Offer constructor with 4 parameters; namely startTime, endTime, profit and productType.



Parameters:

`startTime` - Start time of the offer.

`endTime` - End time of the offer.

`profit` - Profit of the offer.

`productType` - Product type of the offer.

Method Detail

compareTo **[Show source in BlueJ]**

`public int compareTo(Offer o)`

Overrides the compareTo method to sort Offers according to ascending endTime order.

Specified by:

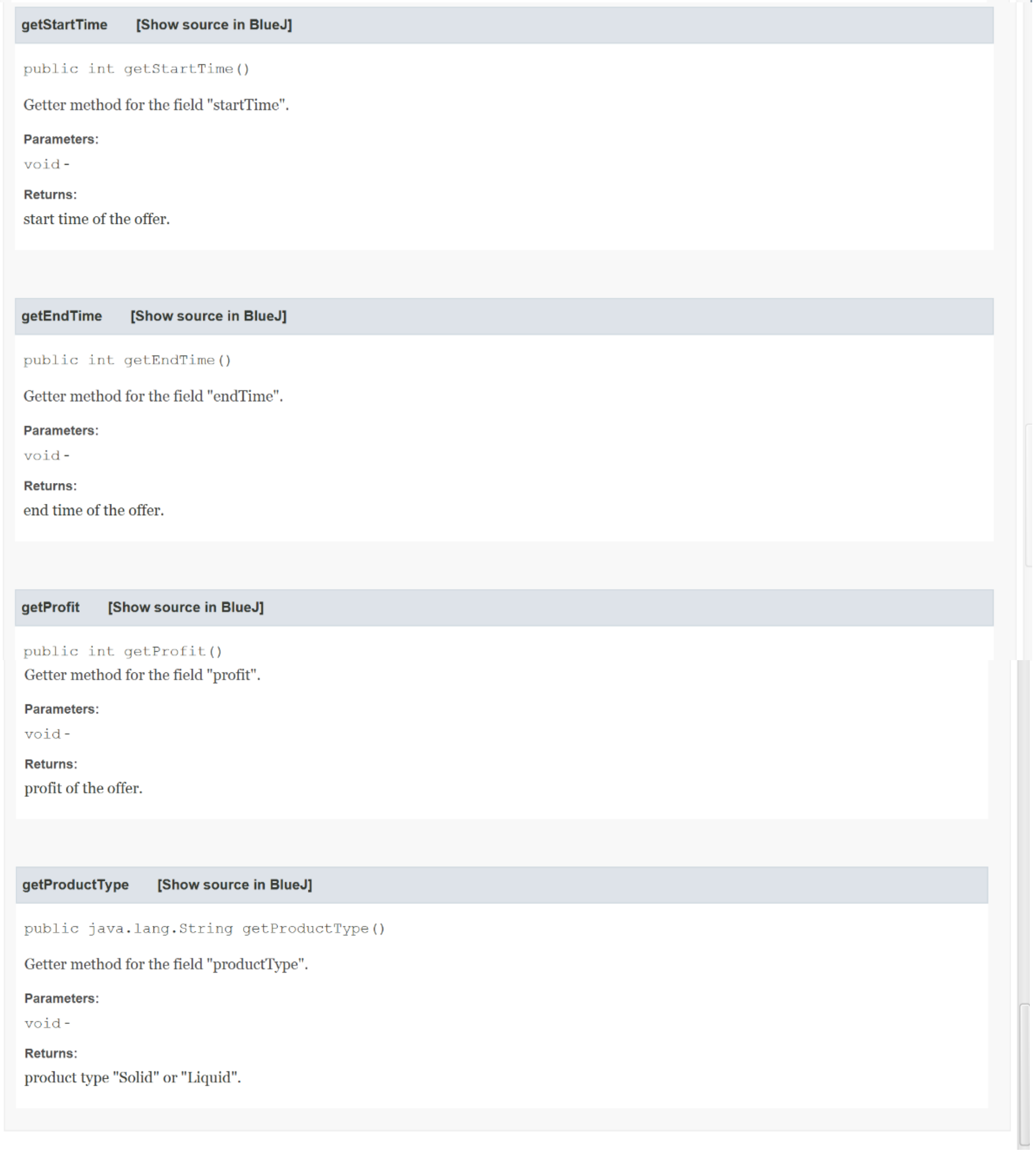
`compareTo` in interface `java.lang.Comparable<Offer>`

Parameters:

`o` - Other Offer.

Returns:

1 if this Offer is of top priority, -1 if the other Offer is of top priority, 0 if they have the same priority.



getStartTime **[Show source in BlueJ]**

`public int getStartTime()`

Getter method for the field "startTime".

Parameters:

`void` -

Returns:

start time of the offer.

getEndTime **[Show source in BlueJ]**

`public int getEndTime()`

Getter method for the field "endTime".

Parameters:

`void` -

Returns:

end time of the offer.

getProfit **[Show source in BlueJ]**

`public int getProfit()`

Getter method for the field "profit".

Parameters:

`void` -

Returns:

profit of the offer.

getProductType **[Show source in BlueJ]**

`public java.lang.String getProductType()`

Getter method for the field "productType".

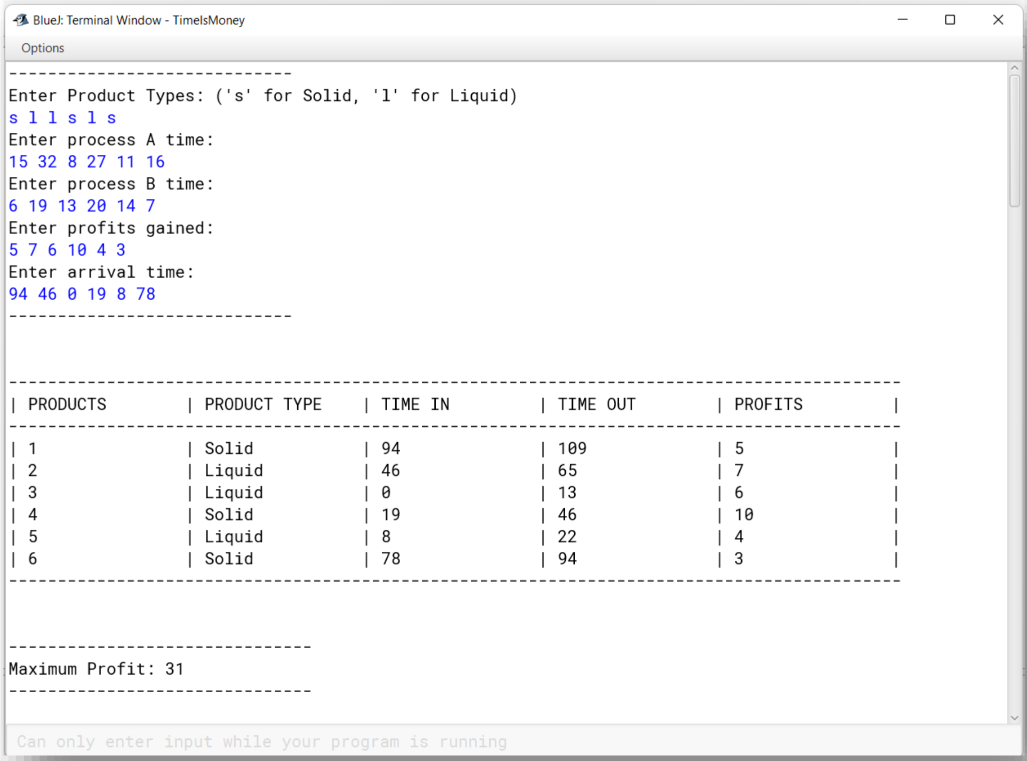
Parameters:

`void` -

Returns:

product type "Solid" or "Liquid".

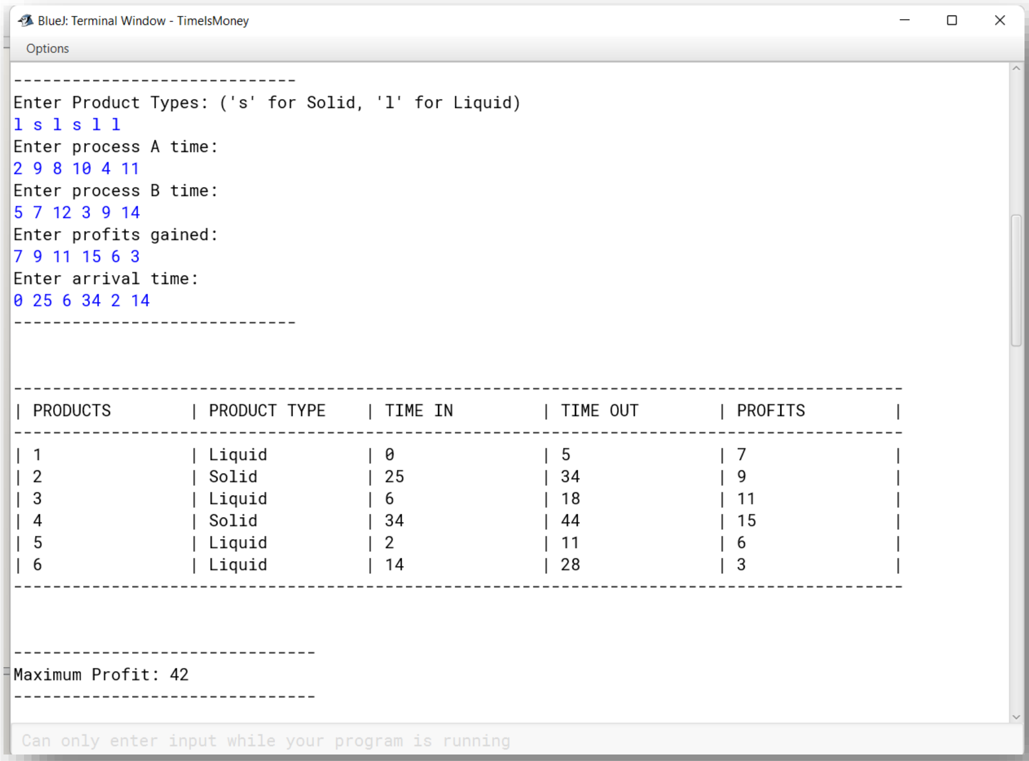
9. TEST CASES


BlueJ: Terminal Window - TimeIsMoney

Options

```
----------------------------
Enter Product Types: ('s' for Solid, 'l' for Liquid)
s l l s l s
Enter process A time:
15 32 8 27 11 16
Enter process B time:
6 19 13 20 14 7
Enter profits gained:
5 7 6 10 4 3
Enter arrival time:
94 46 0 19 8 78
----------------------------


---------------------------------------------------------------------------------------------
| PRODUCTS         | PRODUCT TYPE     | TIME IN          | TIME OUT         | PROFITS          |
---------------------------------------------------------------------------------------------
| 1                | Solid            | 94               | 109              | 5                |
| 2                | Liquid           | 46               | 65               | 7                |
| 3                | Liquid           | 0                | 13               | 6                |
| 4                | Solid            | 19               | 46               | 10               |
| 5                | Liquid           | 8                | 22               | 4                |
| 6                | Solid            | 78               | 94               | 3                |
---------------------------------------------------------------------------------------------


------------------------------
Maximum Profit: 31
------------------------------
```

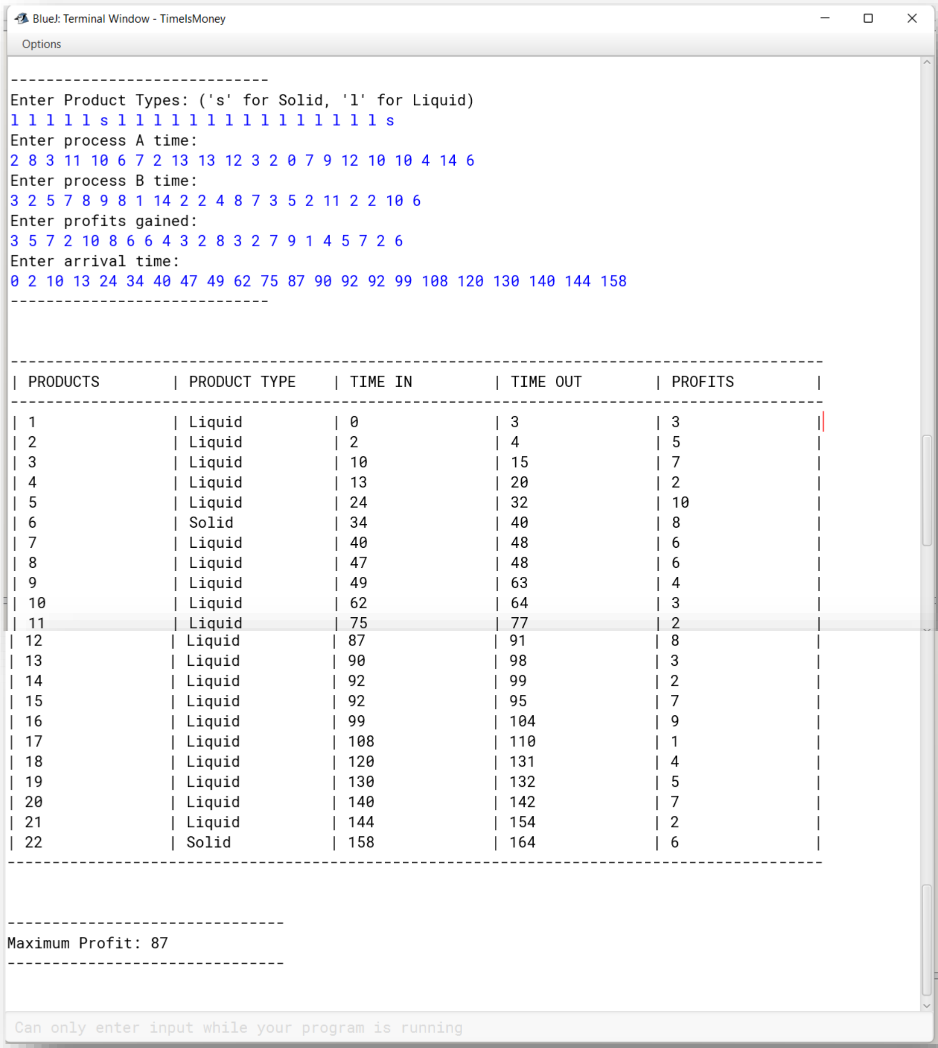
Can only enter input while your program is running


BlueJ: Terminal Window - TimeIsMoney

Options

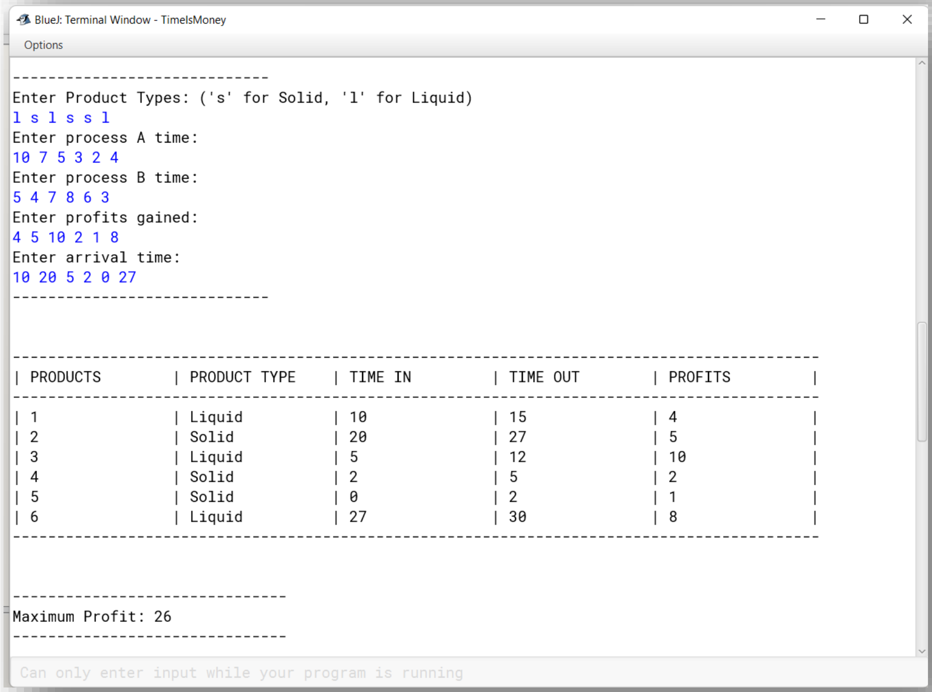
```
----------------------------
Enter Product Types: ('s' for Solid, 'l' for Liquid)
l s l s l l
Enter process A time:
2 9 8 10 4 11
Enter process B time:
5 7 12 3 9 14
Enter profits gained:
7 9 11 15 6 3
Enter arrival time:
0 25 6 34 2 14
----------------------------


---------------------------------------------------------------------------------------------
| PRODUCTS         | PRODUCT TYPE     | TIME IN          | TIME OUT         | PROFITS          |
---------------------------------------------------------------------------------------------
| 1                | Liquid           | 0                | 5                | 7                |
| 2                | Solid            | 25               | 34               | 9                |
| 3                | Liquid           | 6                | 18               | 11               |
| 4                | Solid            | 34               | 44               | 15               |
| 5                | Liquid           | 2                | 11               | 6                |
| 6                | Liquid           | 14               | 28               | 3                |
---------------------------------------------------------------------------------------------


------------------------------
Maximum Profit: 42
------------------------------
```

Can only enter input while your program is running



```
----------------------------
Enter Product Types: ('s' for Solid, 'l' for Liquid)
l l l l l s l l l l l l l l l l l l l l l s
Enter process A time:
2 8 3 11 10 6 7 2 13 13 12 3 2 0 7 9 12 10 10 4 14 6
Enter process B time:
3 2 5 7 8 9 8 1 14 2 2 4 8 7 3 5 2 11 2 2 10 6
Enter profits gained:
3 5 7 2 10 8 6 6 4 3 2 8 3 2 7 9 1 4 5 7 2 6
Enter arrival time:
0 2 10 13 24 34 40 47 49 62 75 87 90 92 92 99 108 120 130 140 144 158
----------------------------


------------------------------------------------------------------------------------------
| PRODUCTS          | PRODUCT TYPE     | TIME IN          | TIME OUT         | PROFITS          |
------------------------------------------------------------------------------------------
| 1                 | Liquid           | 0                | 3                | 3                |
| 2                 | Liquid           | 2                | 4                | 5                |
| 3                 | Liquid           | 10               | 15               | 7                |
| 4                 | Liquid           | 13               | 20               | 2                |
| 5                 | Liquid           | 24               | 32               | 10               |
| 6                 | Solid            | 34               | 40               | 8                |
| 7                 | Liquid           | 40               | 48               | 6                |
| 8                 | Liquid           | 47               | 48               | 6                |
| 9                 | Liquid           | 49               | 63               | 4                |
| 10                | Liquid           | 62               | 64               | 3                |
| 11                | Liquid           | 75               | 77               | 2                |
| 12                | Liquid           | 87               | 91               | 8                |
| 13                | Liquid           | 90               | 98               | 3                |
| 14                | Liquid           | 92               | 99               | 2                |
| 15                | Liquid           | 92               | 95               | 7                |
| 16                | Liquid           | 99               | 104              | 9                |
| 17                | Liquid           | 108              | 110              | 1                |
| 18                | Liquid           | 120              | 131              | 4                |
| 19                | Liquid           | 130              | 132              | 5                |
| 20                | Liquid           | 140              | 142              | 7                |
| 21                | Liquid           | 144              | 154              | 2                |
| 22                | Solid            | 158              | 164              | 6                |
------------------------------------------------------------------------------------------


-------------------------------
Maximum Profit: 87
-------------------------------
```

Can only enter input while your program is running


BlueJ: Terminal Window - TimeIsMoney
Options

```
----------------------------
Enter Product Types: ('s' for Solid, 'l' for Liquid)
l s l s s l
Enter process A time:
10 7 5 3 2 4
Enter process B time:
5 4 7 8 6 3
Enter profits gained:
4 5 10 2 1 8
Enter arrival time:
10 20 5 2 0 27
----------------------------


------------------------------------------------------------------------------------------
| PRODUCTS          | PRODUCT TYPE     | TIME IN          | TIME OUT         | PROFITS          |
------------------------------------------------------------------------------------------
| 1                 | Liquid           | 10               | 15               | 4                |
| 2                 | Solid            | 20               | 27               | 5                |
| 3                 | Liquid           | 5                | 12               | 10               |
| 4                 | Solid            | 2                | 5                | 2                |
| 5                 | Solid            | 0                | 2                | 1                |
| 6                 | Liquid           | 27               | 30               | 8                |
------------------------------------------------------------------------------------------


-------------------------------
Maximum Profit: 26
-------------------------------
```

Can only enter input while your program is running

--END OF REPORT--