**DATA STRUCTURE**

**PROGRAMS**

PROGRAM 1(Stack)

**Algorithm for a Push in a Stack**

Step 1: item=size, top=-1 /\* shift the elements downward to create space for new entrant \*/

Step 2: if top = item.length-1

{

Step 3: print “Stack is full”

}

Step 4: else {

Step 5: item[++top] = data /\* Push the new entrant ITEM \*/

}

**Algorithm for a pop in a stack.**

1.Step 1: if top < 0 { // when stack is empty

2.Step 2: print “Stack underflow”

}

3. Step 3: else {

4. Step 4: print ”pop item” //the item which is deleted

5 Step 5: return item[top--]

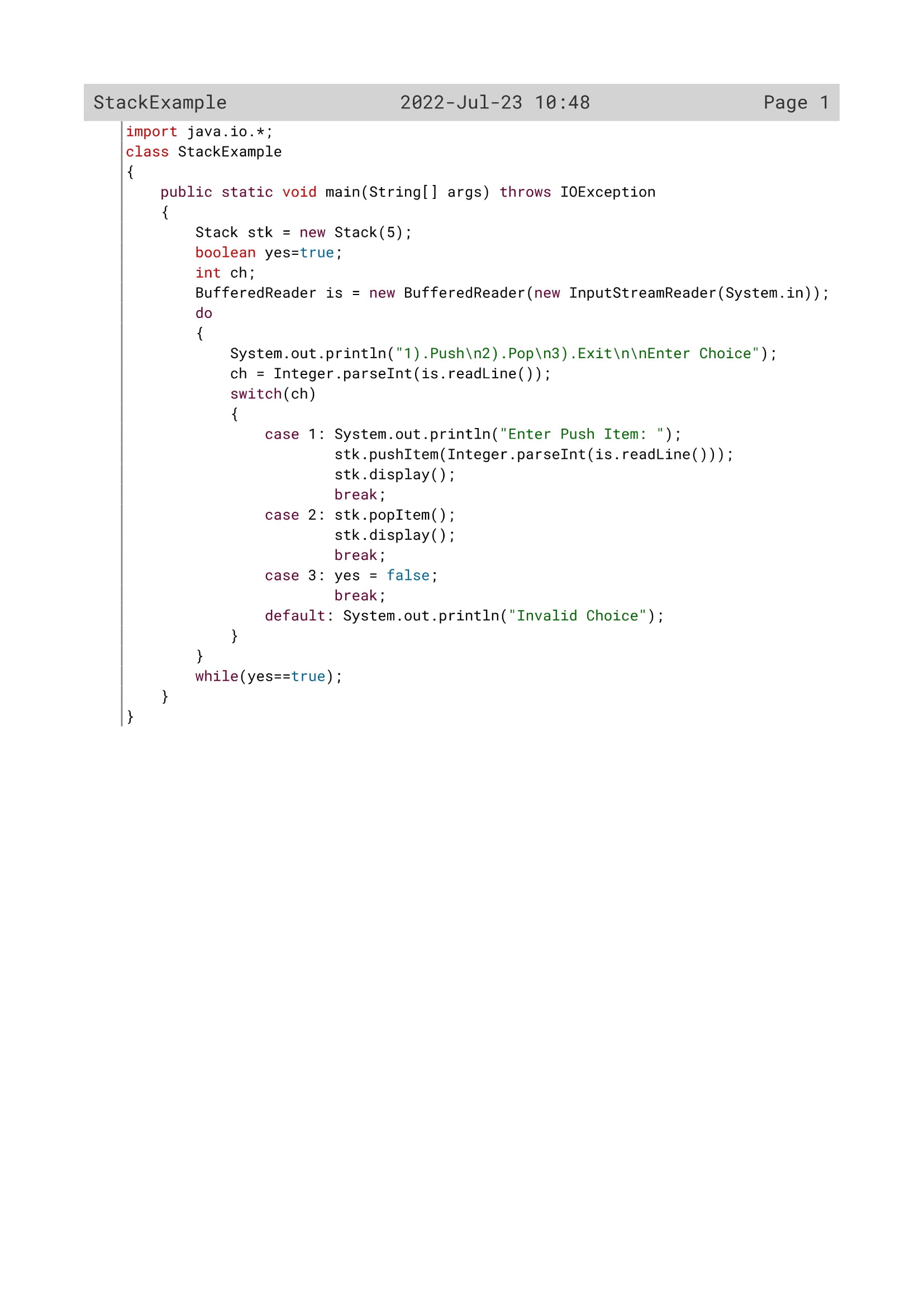
}

204

Source Code



205



206

Variable Description

|  |  |  |
| --- | --- | --- |
| Variable name | Data type | Description |
| i | int | Loop variable |
| top | int | To act as stack pointer |
| Item[][] | Int | To insert store stack values |
| size | Int | To store array size |
| data | int | Parameter for pushItem() |

For stackExample

|  |  |  |
| --- | --- | --- |
| Variable name | Data type | Description |
| yes | boolean | Flag variable |
| ch | int | To insert the elements |

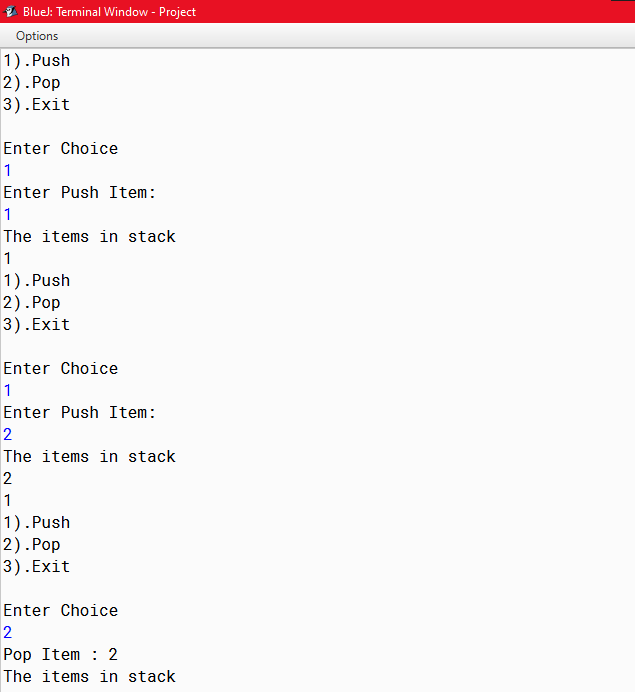
207

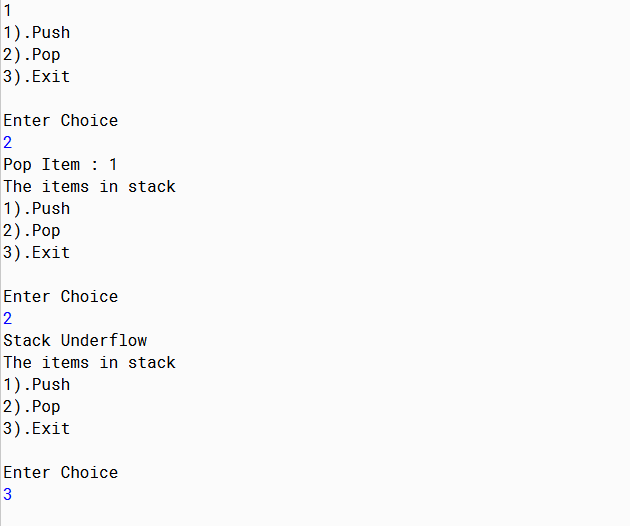
Function Description

|  |  |  |
| --- | --- | --- |
| Function name | Data type | description |
| pushItem() | void | To store items in the stack |
| popItem() | int | To pop the element and display it |
| display() | void | To display all stack elements |

208

OUTPUT





209

PROGRAM 2(Queue)

**ALGORITHM FOR AN INSERTION IN AN ARRAY QUEUE**

//accept the number in v

1. if rear = size - 1 then

2. print "OVERFLOW"

3. else if front= -1 then

4. Q [++front]= v

5. else

6. rear++

7. Q[rear]=v

**ALGORITHM FOR AN DELETION IN AN ARRAY QUEUE**

1. if front = -1 then

2. print "UNDERFLOW"

3. return -999

4. else {

5. val= Q[front]

6. if front=rear then {

7. front =-1

8. rear = -1

}

9. else

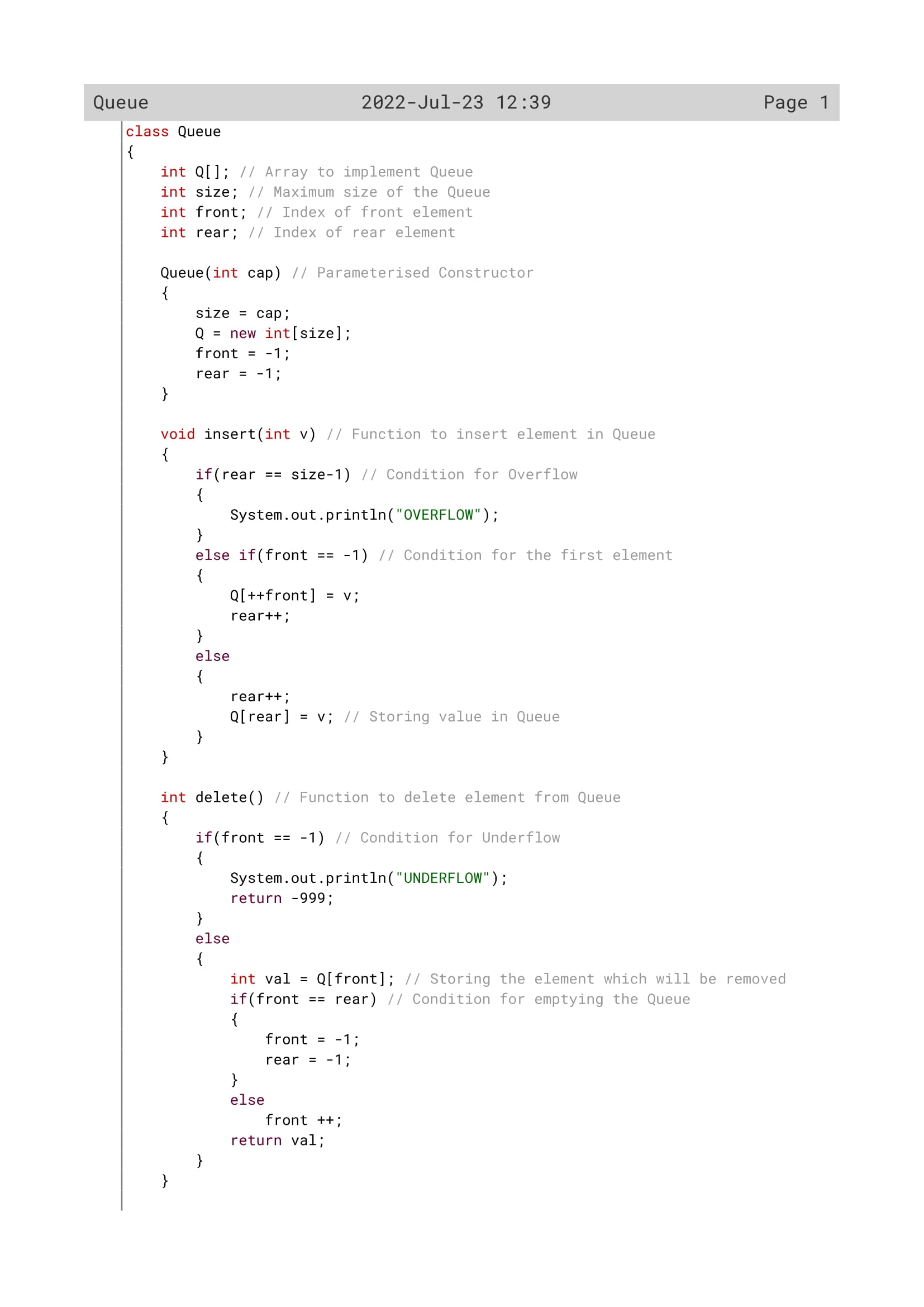
10. front++

11. return val

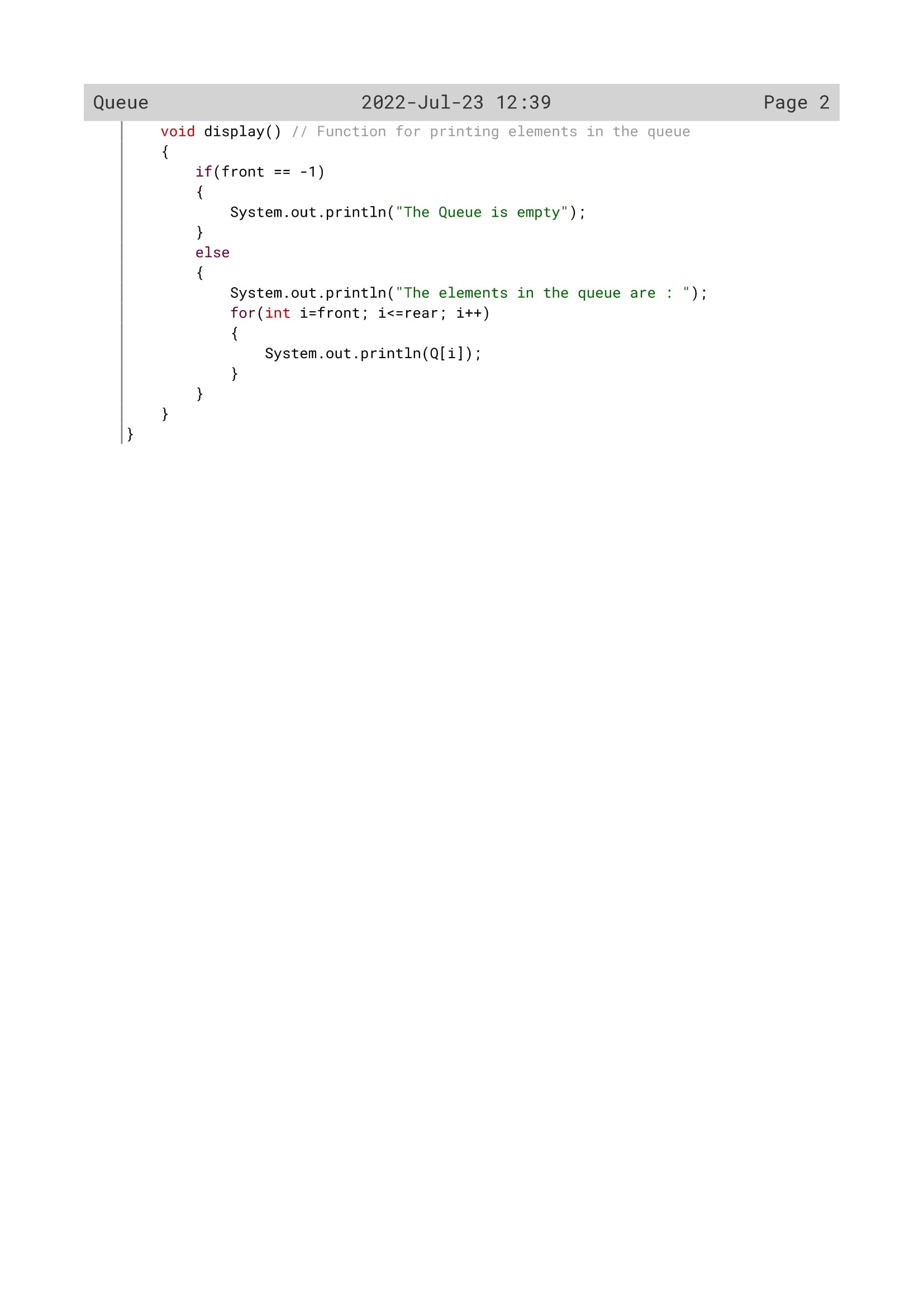
}

210

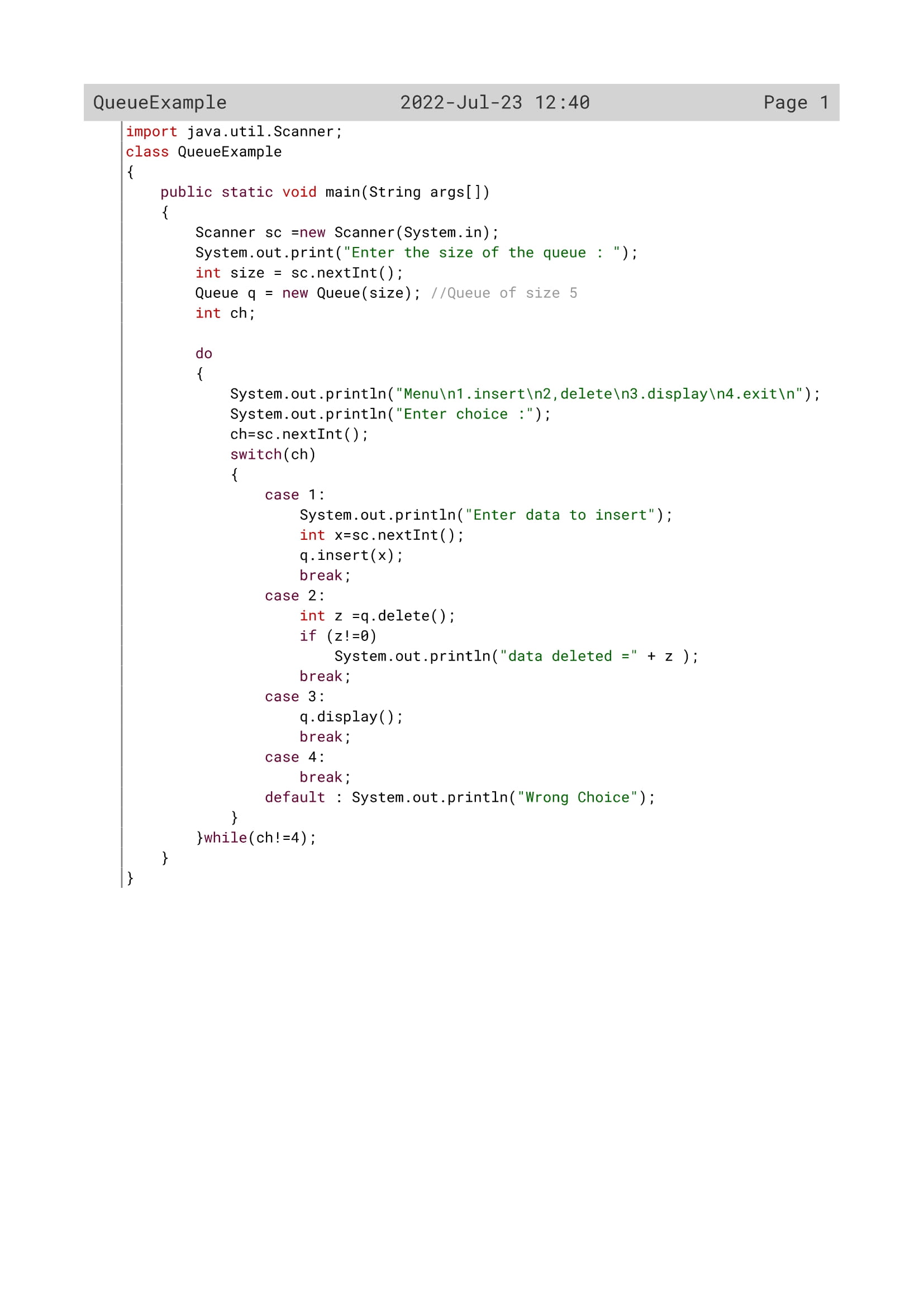
Source Code



211



212



213

Variable Description

|  |  |  |
| --- | --- | --- |
| Variable name | Data type | Description |
| Q[] | Int | Array to implement Queue |
| size | Int | Maximum size of the Queue |
| front | Int | Index of front element |
| rear | Int | Index of rear element |
| cap | Int | To accept the elements in queue |
| v | Int | To insert element in queue |
| val | int | For Storing element which will be removed |
| i | Int | Loop variable |

For queueExample

|  |  |  |
| --- | --- | --- |
| Variable name | Data type | description |
| size | Int | To enter the size of queue |
| ch | Int | To enter the choice by the user |
| x | Int | To enter the data to insert |
| z | Int | To delete |
| q | Int | Queue of size 5 |

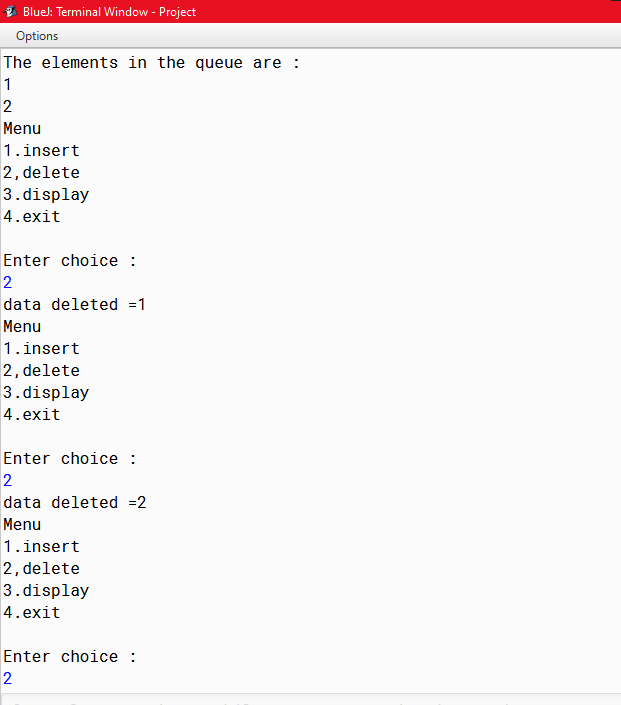
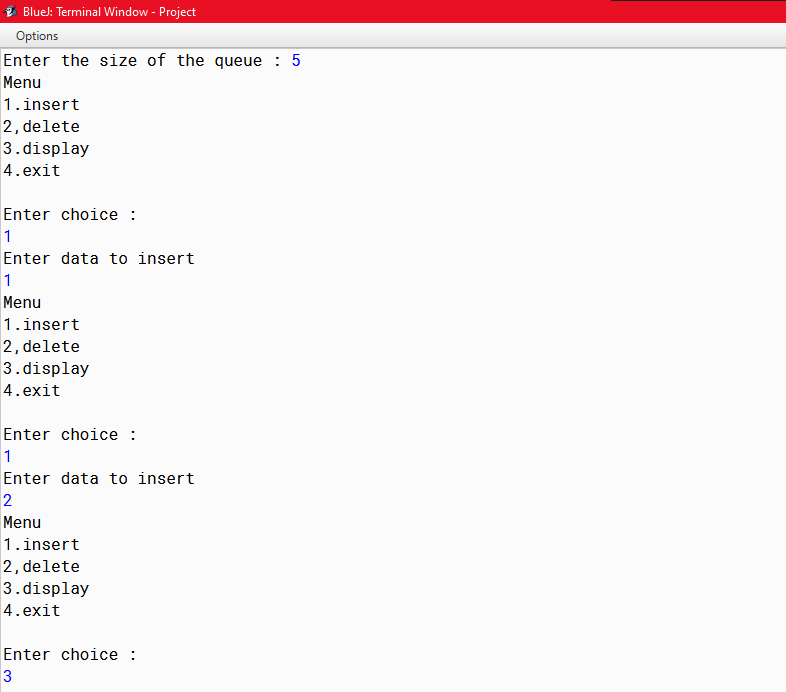
214

Function Description

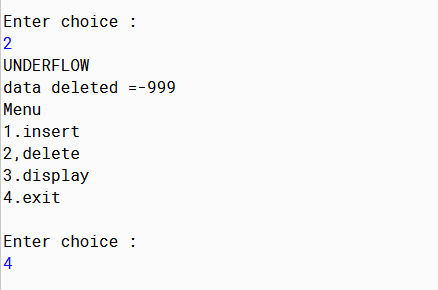
|  |  |  |
| --- | --- | --- |
| Function name | Data type | description |
| insert() | void | To store items in the queue |
| delete() | int | To delete the element and display it |
| display() | void | To display all queue elements |

215

OUTPUT



216



217

PROGRAM 3(SinglyLinkedList)

**Algorithm to Insert a Node in a Sorted Lists**

This Algorithm deals with all three cases of insertion 1.e., in the beginning, in the end and in between "/

/\*Initialize the pointer\*/

1. ptr – START // START denotes the first node of the list

/\*ALLocate memory for the new node \*/

2. NEWPTR = new Node

3. If NEWPTR = NULL // If sufficient memory is not available

4. print "No Space Available ! Aborting !!"

5. else

{

6. NEWPTR.INFO = ITEM /\* put information in new node \*/

7. NEWPTR.LINK= NUL // Initialise the reference

8. IF START = NULL then

9. START-NEWTR //insert in the empty.list

10. else if ITEM< START.INFO then

11. Save = START /\* Save Start's value \*/

12. START = NEWPTR /\* Assign NEWPTR to START \*/

13. NEWPTR.LINK= Save /\* Make NEWPTR point to previous START \*/

}

/\* Find appropriate position for ITEM for insertion in the middle \*/

14. Save = Start

15. Repeat steps 16 through 22 until ptr = NULL

218

16. if NEWPTR.INFO > ptr.INFO then

{

17. Save = ptr

18. ptr = ptr.LINK

}

19. else /\* insert in between two nodes \*/

{

20. Save.LINK = NEWPTR

21. NEWPTR.LINK = ptr

22. break

} /\*end of if\*/ /\*jump out of loop\*/

/\* End of Repeat \*/

23. if ptr = NULL then

{

24. Save.LINK = NEWPTR

25. NEWPTR.LINK = NULL

}

} /\* of step 3 \*/

26. END.

**Algorithm to delete a node in a list**

1. save start, ptr = start /\* Initialise references /

2. Repeat step 3 through 7 until ptr = NULL /\* Search for ITEM in the List

3. If ptr.INFO== ITEM then

4. break /\* jump out of the Loop/

5. else

{

6. save = ptr

219

7. ptr = ptr.LINK

} // end of step 3

8. if ptr NULL then

9. print Item "not found !!"

10. else

{

11. if ptr == start then. /\* check whether first node is to be delete

12. start= ptr.LINK

13. else

14. Save.LINK= ptr.LINK

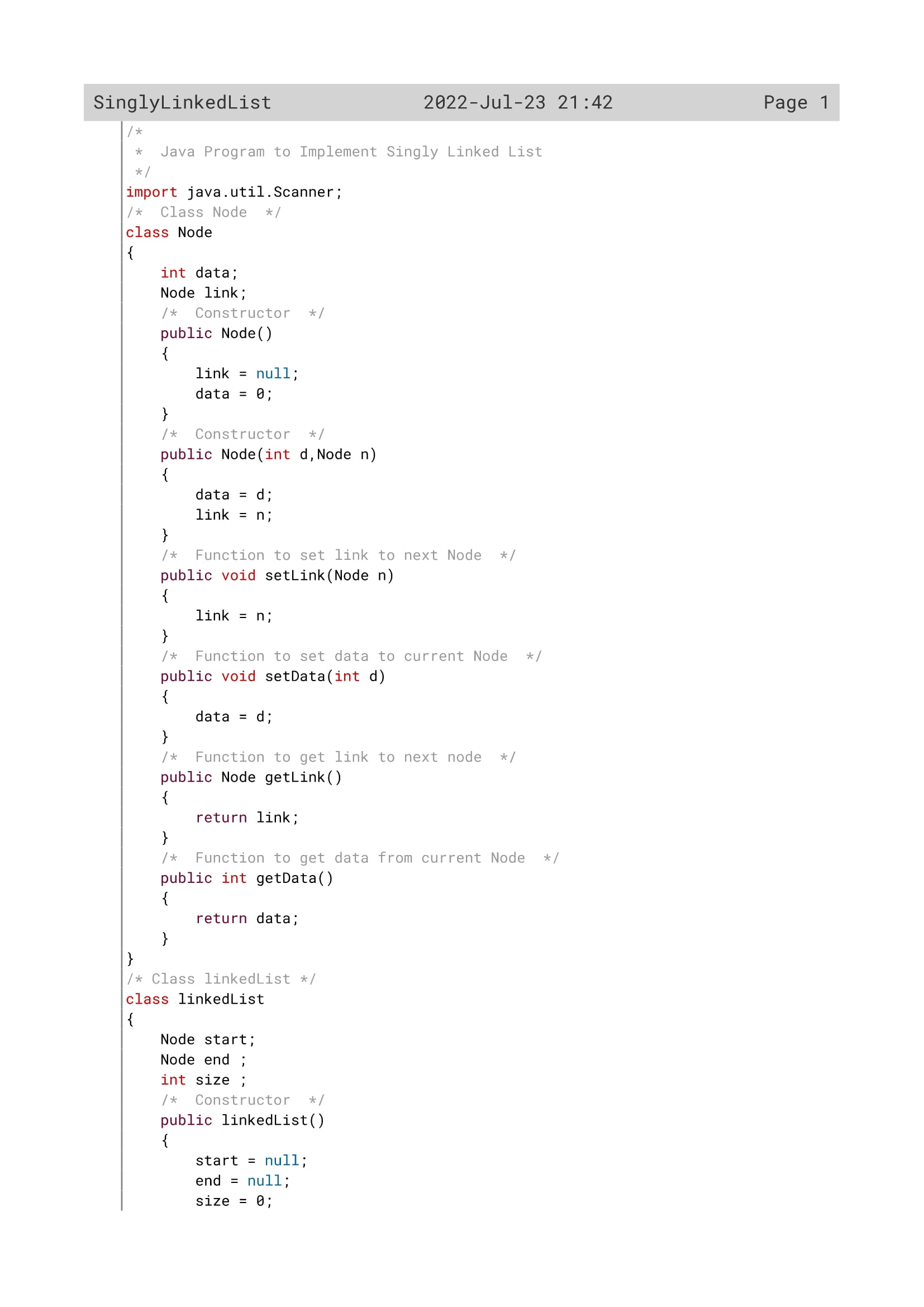
// end of step 11

} // end of step 8

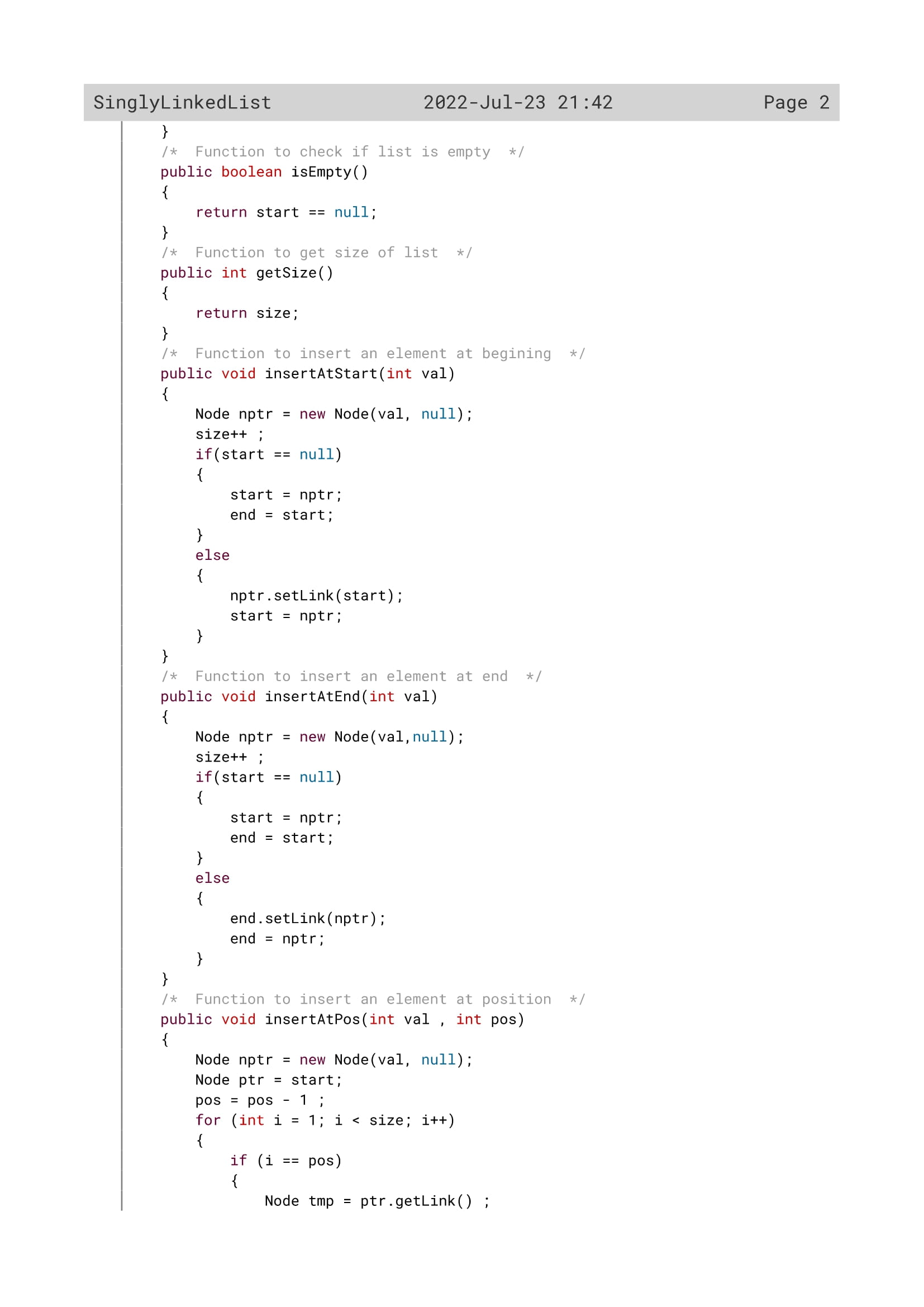
15. END.

220

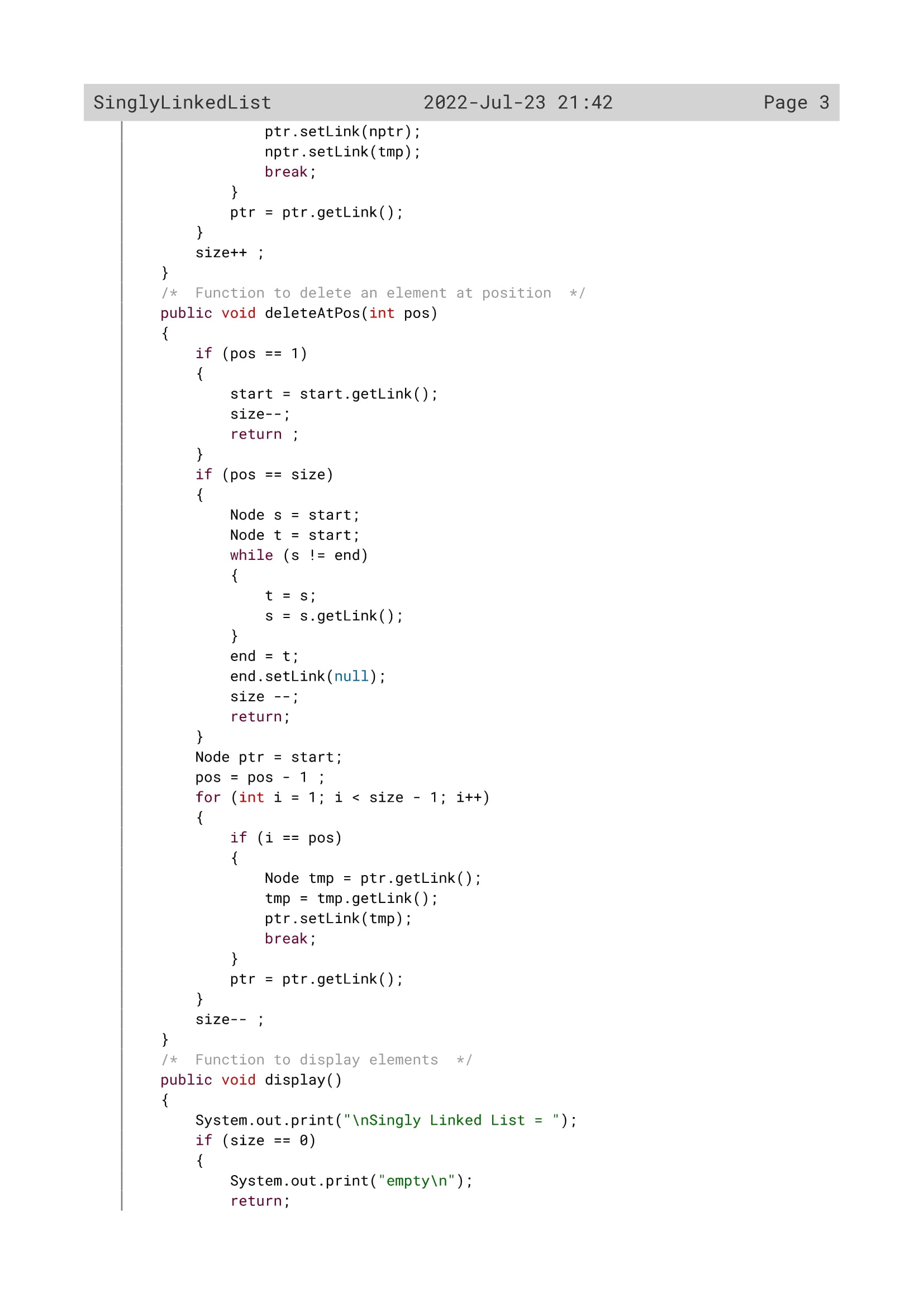
Source Code



221



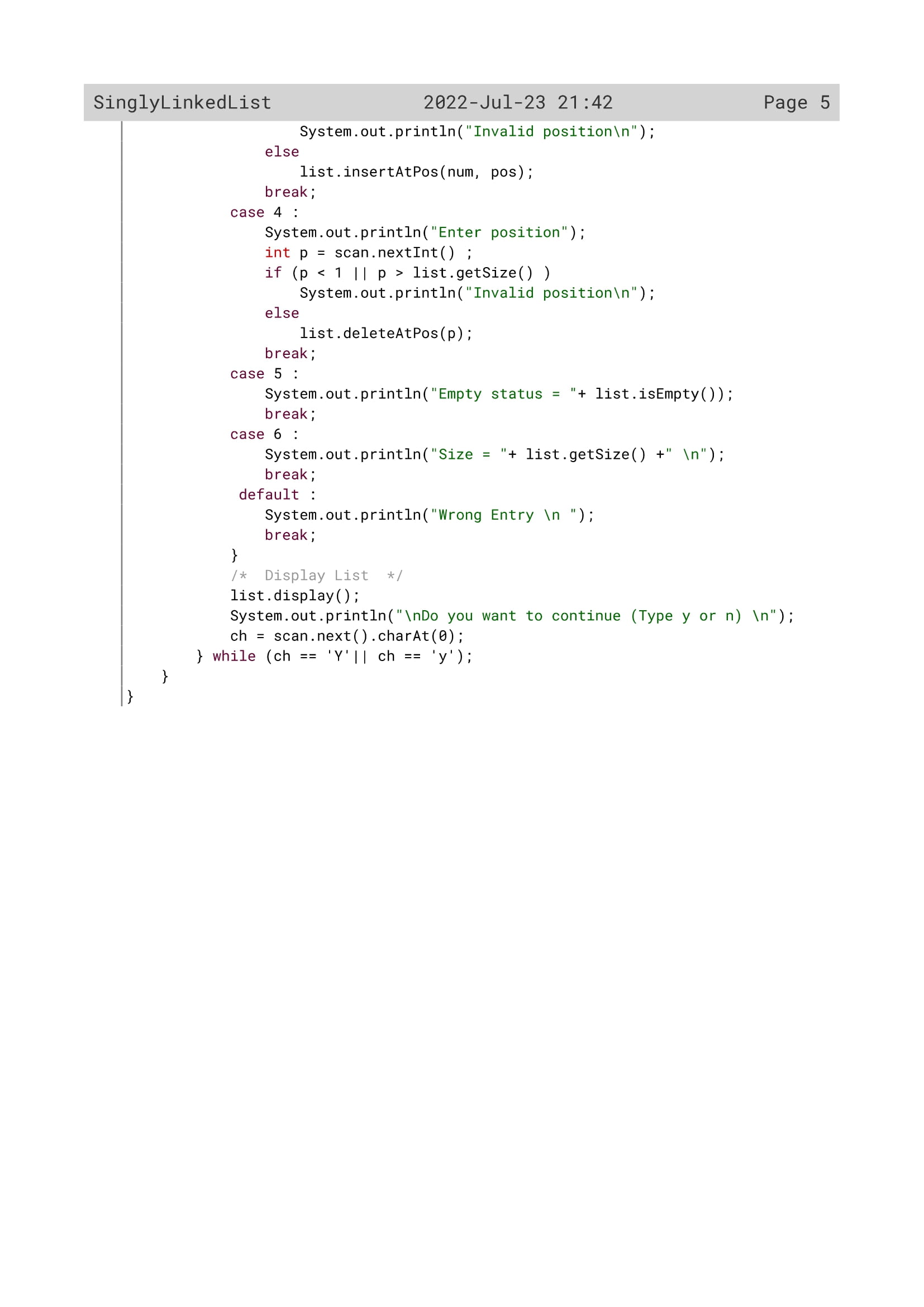
222



223



224



225

Variable Description

|  |  |  |
| --- | --- | --- |
| VARIABLE NAME | DATA TYPE | DESCRYPTION |
| ch | int | To accept choice from user |
| num | int | To accept the number to be inserted |
| pos | int | To accept the position where the number is to be inserted |
| data | int | To store the element |
| link | int | To store null value |
| d | int | To receive the element |
| n | int | To receive the null value |
| size | int | To store the total number of the elements |
| start | int | To store the number from  *nptr* |
| end | int | To store the number from *start* |
| i | int | Loop variable |

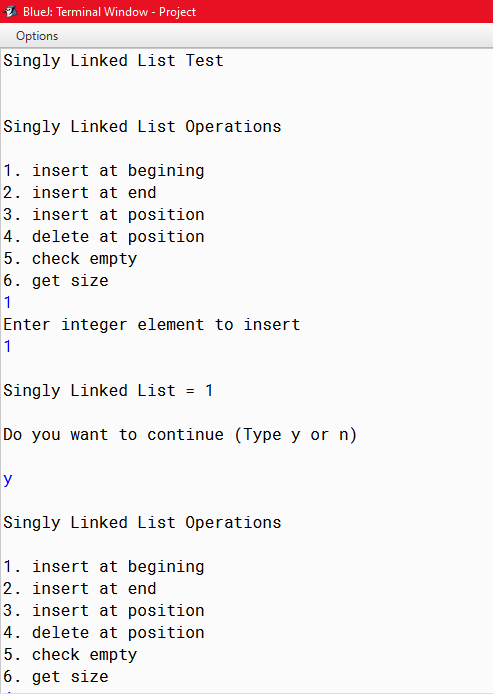
226

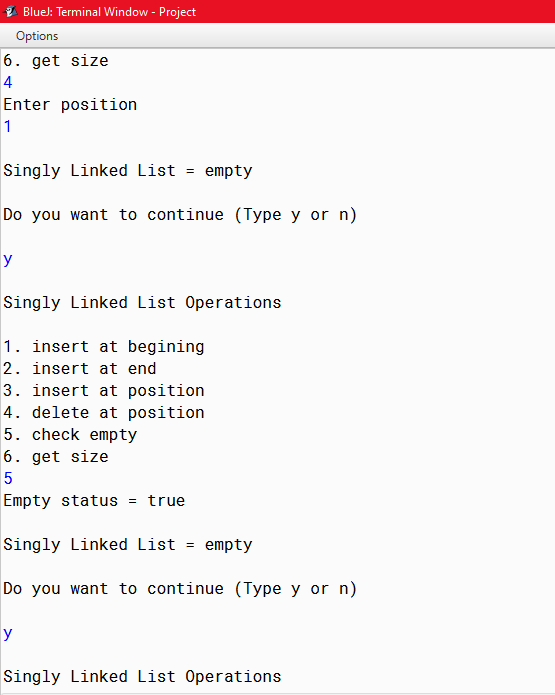
Function Description

|  |  |  |  |
| --- | --- | --- | --- |
| FUNCTION NAME | RETURN TYPE | PARAMETERS | Description |
| Node | null | int | To initialize *d* to *data* and *n* to *link* |
| setLink | void | null | To store *n* to *link* |
| setData | void | int | To store value of *d* to *data* |
| getLink | null | null | Constructor to return *link* |
| getData | int | null | To return *data* |
| linkedList | null | null | To initialize *null* to *start, null* to *end, 0* to *size.* |
| isEmpty | boolean | null | To return *true* if *start* is equal to *null* |
| getSize | int | null | To return size |
| insertAtStart | void | int | To insert value at the beginning |
| insertAtEnd | void | int | To insert value at the end |
| insertAtPos | void | int, int | To insert value at any position |
| deleteAtPos | void | int | To delete value at any position |
| display | void | null | To display elements |
| main | void | null | To call the above functions by creating object and to accept the choice from the user, where to insert and delete elements |

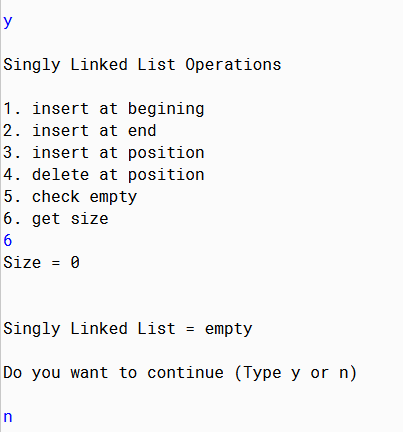
227

OUTPUT





228



229

PROGRAM 4(Bubble Sort)

ALGORITHM FOR main()

Step 1: start

Step 2: Scanner class object declared

Step 3: an array arr[] is initialised and declared to store 10 numbers. The numbers are accepted from user in arr[] variable

Step 4: an object BubbleSort is created. The object calls the functions bubbleSort() and printArray()

to display the sorted array.

Step 5: Stop.

ALGORITHM FOR bubbleSort()

Step 1: start

Step 2: the elements are received in arr[] variable and the total number of elements in arr[] is stored in n variable.

Step 3: A for loop i runs from 0 to less than (n-1) and inner loop j runs from 0 to less than (n-i-1). Based on ith value and jth value the numbers in arr[j] and arr[j+1] are compared. If the numbers in arr[j] is greater than arr[j+1] then the numbers will be swapped up. The smaller number will be stored first then then bigger number. In this way all the numbers will to compared and will be arranged in ascending order.

Step 4: Stop.

ALGORITHM FOR printArray()

Step 1: start

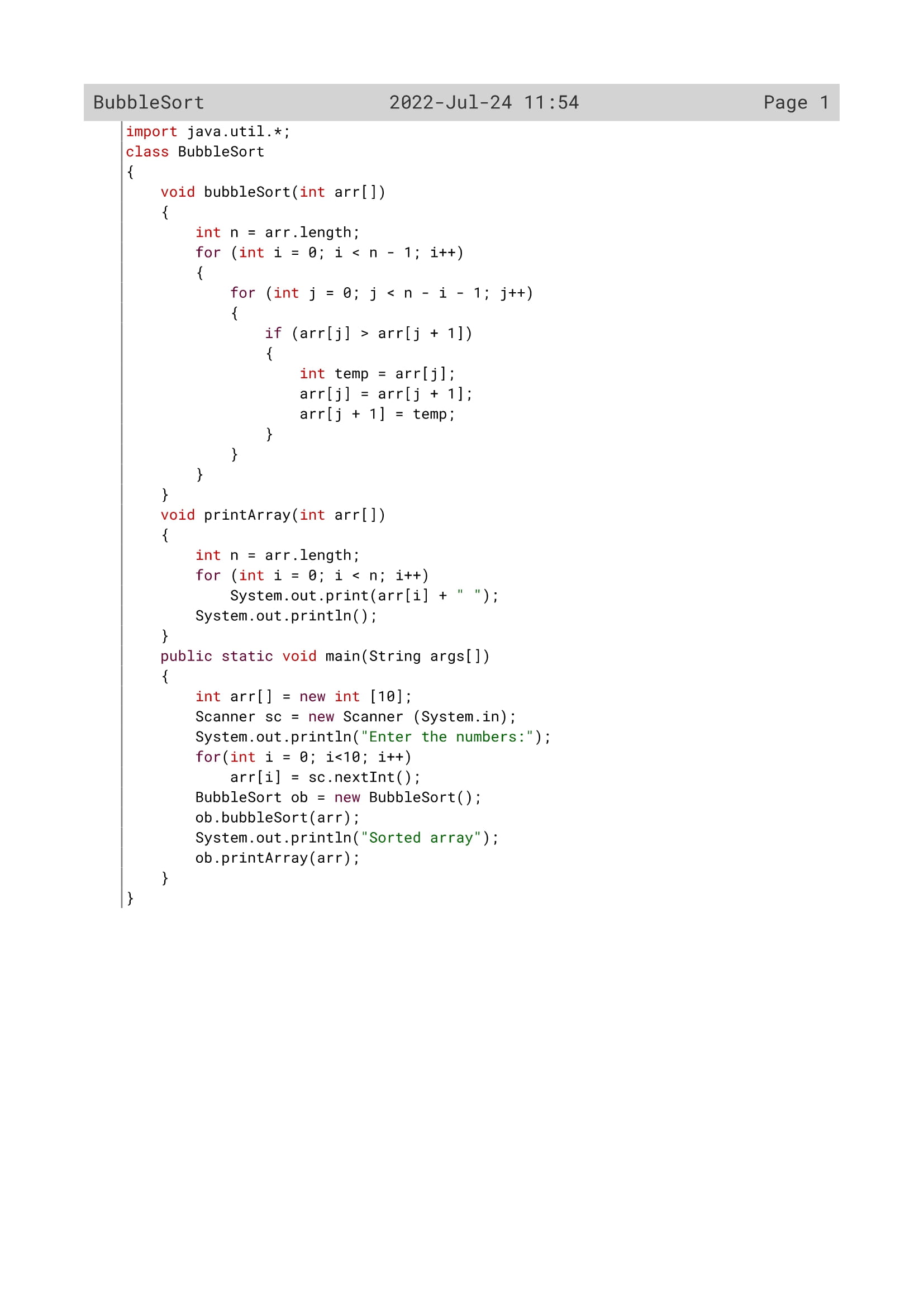
Step 2: the elements are received in arr[] variable and the total number of elements in arr[] is stored in n variable.

Step 3: by using a loop variable i the numbers in arr[] will be displayed in ascending order.

Step 4: Stop.

230

Source Code



231

Variable Description

|  |  |  |
| --- | --- | --- |
| Variable name | Data type | Description |
| i | Int | Loop variable |
| j | int | Loop variable |
| temp | int | To sort the array in ascending order |
| arr[] | int | To store the array and display it |
| n | int | To accept the numbers from the user |

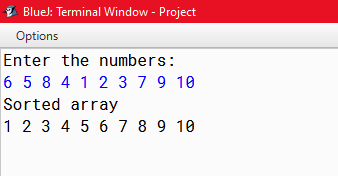
232

Function Description

|  |  |  |
| --- | --- | --- |
| Function name | Data type | description |
| bubblesort() | Void | To sort the numbers in the array in ascending order using bubblesort |
| printArray() | void | To print the sorted array and display it to the user |
| main() | void | To input the elements into the array |

233

OUTPUT



234

PROGRAM 5(Insertion Sort)

ALGORITHM FOR main()

Step 1: start

Step 2: Scanner class object declared

Step 3: an array arr[] is initialised and declared to store 10 numbers. The numbers are accepted from user in arr[] variable

Step 4: an object Insertion\_Sort is created. The object calls the functions Sort() and printArray()

to display the sorted array.

Step 5: Stop

ALGORITHM FOR Sort()

Step 1: start

Step 2: the elements are received in arr[] variable and the total number of elements in arr[] is stored in n variable.

Step 3: A for loop i runs from 1 to less than n executing the following steps.

Step 3.1: In variable key each elements will be stored and in j variable (i-1) is stored.

Step 3.2: a while loop will be executed when j is greater than or equal to 0 and when the number stored in arr[j] is greater then key.

Step 3.2.1: the element stored in arr[j] will be stored in arr[j+1].The bigger number will be stored after the smaller number.

Step 3.2.2: decrease the value of j by 1.

Step 3.3: after the execution of while loop execute arr[j+1]=key. In this way all the numbers will to compared and will be arranged in ascending order.

Step 4: Stop.

ALGORITHM FOR printArray()

Step 1: start

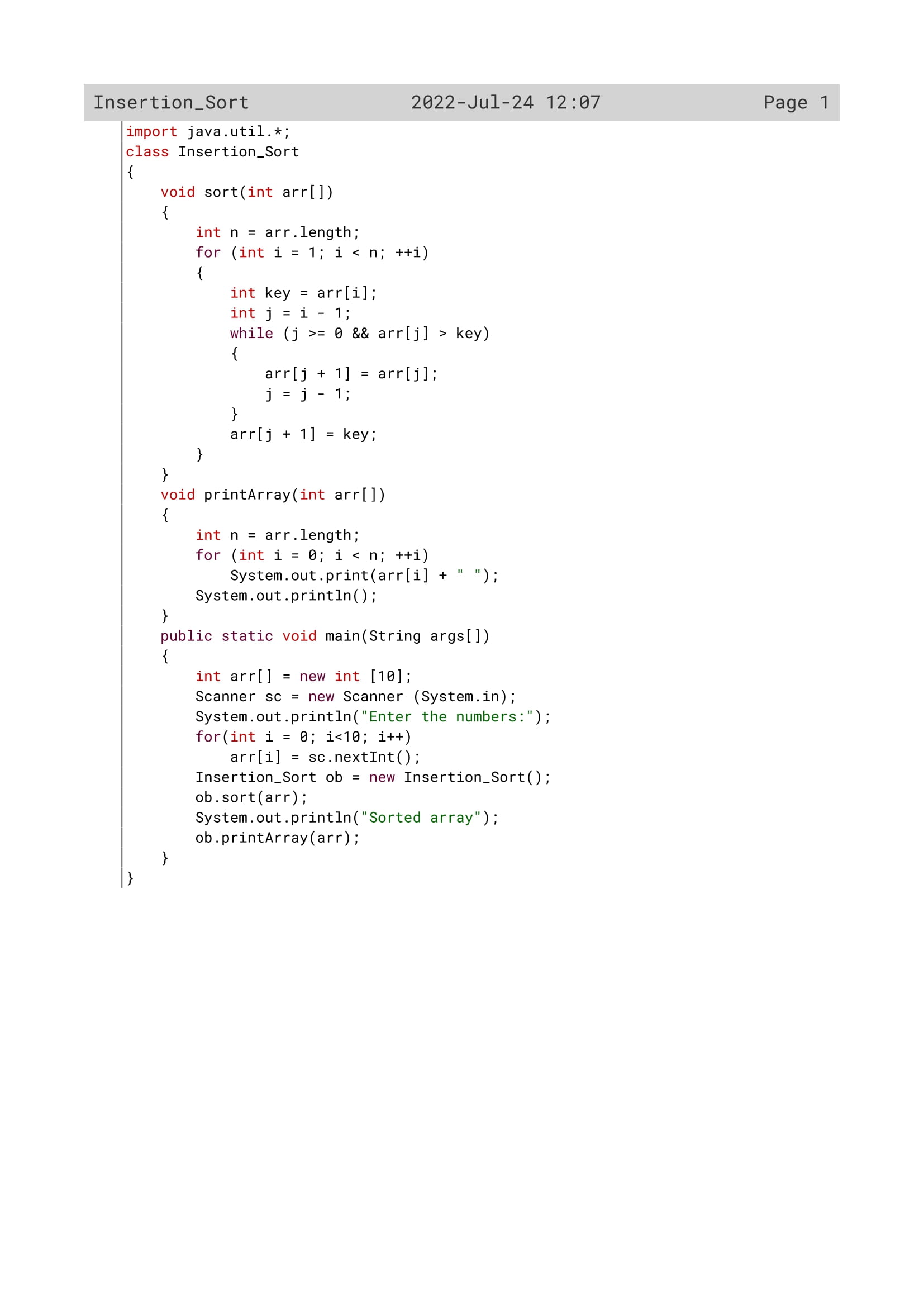
Step 2: the elements are received in arr[] variable and the total number of elements in arr[] is stored in n variable.

Step 3: by using a loop variable i the numbers in arr[] will be displayed in ascending order.

Step 4: Stop.

235

Source Code



236

Variable Description

|  |  |  |
| --- | --- | --- |
| Variable name | Data type | Description |
| n | Int | To find the length of the array |
| i | Int | Loop variable |
| arr[] | Int | To store the numbers in the array |
| j | Int | To sort the array in ascending order |
| key | int | To sort the array in ascending order using insertion sort |

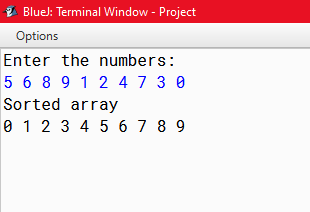
237

Function Description

|  |  |  |
| --- | --- | --- |
| Function name | Data type | description |
| sort() | Void | To sort the numbers in the array in ascending order using Insertionsort |
| printArray() | void | To print the sorted array and display it to the user |
| main() | void | To input the elements into the array |

238

OUTPUT



239

PROGRAM 6(Selection Sort)

ALGORITHM FOR main()

Step 1: start

Step 2: Scanner class object declared

Step 3: an array arr[] is initialised and declared to store 10 numbers. The numbers are accepted from user in arr[] variable

Step 4: an object SelectionSort is created. The object calls the functions Sort() and printArray()

to display the sorted array.

Step 5: Stop

ALGORITHM FOR Sort()

Step 1: start

Step 2: the elements are received in arr[] variable and the total number of elements in arr[] is stored in n variable.

Step 3: A for loop i runs from 1 to less than n-1 executing the following steps.

Step 3.1: In variable min\_idx elements will be stored one by one.

Step 3.2: Another for loop j will be executed from i+1 to less than n

Step 3.2.1:If the value of arr[j] is less than arr[min\_idx], then store jth value to min\_idx.

Step 3.3: now swap up the number in temp, arr[min\_idx], and arr[i] variables. In this way the smaller number will be stored first then then bigger number and all the numbers will to compared with one another and will be arranged in ascending order.

Step 4: Stop.

ALGORITHM FOR printArray()

Step 1: start

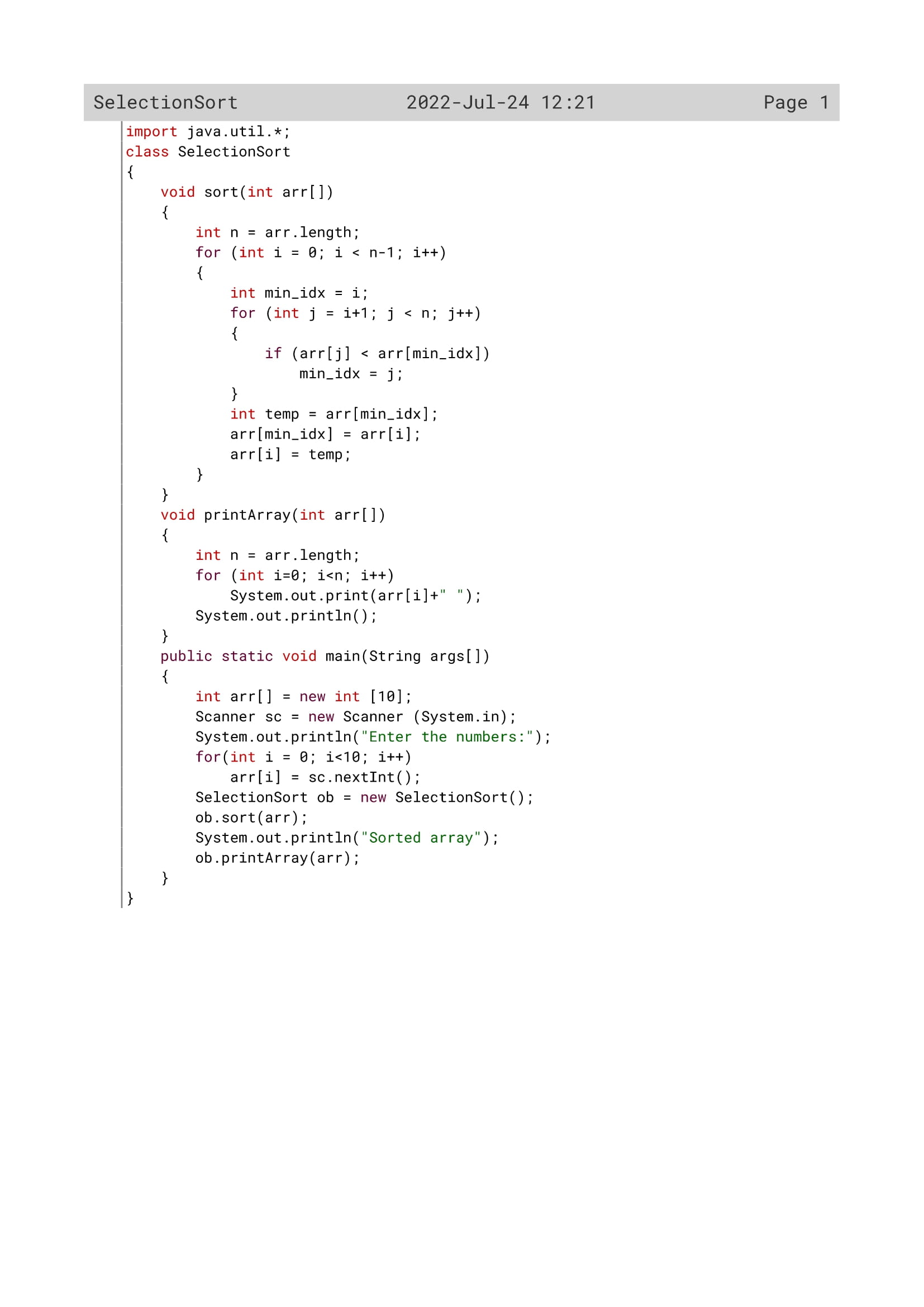
Step 2: the elements are received in arr[] variable and the total number of elements in arr[] is stored in n variable.

Step 3: by using a loop variable i the numbers in arr[] will be displayed in ascending order.

Step 4: Stop.

240

Source Code



241

Variable Description

|  |  |  |
| --- | --- | --- |
| Variable name | Data type | Description |
| arr[] | Int | To create an array and store the numbers in it |
| i | Int | Loop variable |
| min\_idx | Int | To sort the numbers in ascending order using selection sort |
| j | int | Loop variable |
| temp | Int | To sort the numbers in ascending order using selection sort |
| n | Int | To insert the elements in the array by the user |

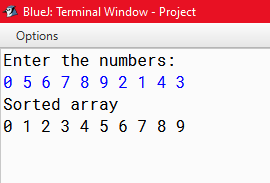
242

Function Description

|  |  |  |
| --- | --- | --- |
| Function name | Data type | description |
| sort() | void | To sort the numbers in the array in ascending order using SelectionSort |
| printArray() | void | To print the sorted array and display it to the user |
| main() | void | To input the elements into the array |

243

OUTPUT



244