**CORE JAVA TRAINING - INDEX**

**Date:05/08/2024**

1. Language And Applications
2. JAVA Features

* Why Java is Independent?
* Oops
* Exception Handling
* Multi threading
* Web Application
* Open Source
* Security
* Support Networking
* Memory Management

1. JDK,JRE,JVM
2. Basic Java Programming
3. Packages

**Date:06/08/2024**

**Morning(11:00 am)**

1. Nested Loops
2. One Dimensional Array
3. Two Dimensional Array
4. Logical Programming

**After Noon(3:30 pm)**

1. SwitchCase
2. Scanner Class
3. Java.lang

* Object Class Methods

1. Enum
2. Event Management Application

**Date:07/08/2024**

**Mrng(11:00 am)**

1.oops

* Encapsulation

Programs

Calculation

Person

Method Flow

**After Noon(3:30 pm)**

1. Inheritance
2. Polymorphism

**.**Method overloading

**.**Method Overriding

1. Abstraction
2. IS-A (Inheritance)
3. HAs- A (Object Creation).

**Date:08/08/24**

**Constructor**

i. Class name and constructor name should be same

ii. There are 2 types of constructors

a. Default Constructor

b. Parameterized Constructor

iii. We can access constructor while creation of object

iv. Constructors are mainly for initializing

v. Constructor doesn’t have any return type not even void. If you declare as a void the compiler will consider as a method not a constructor

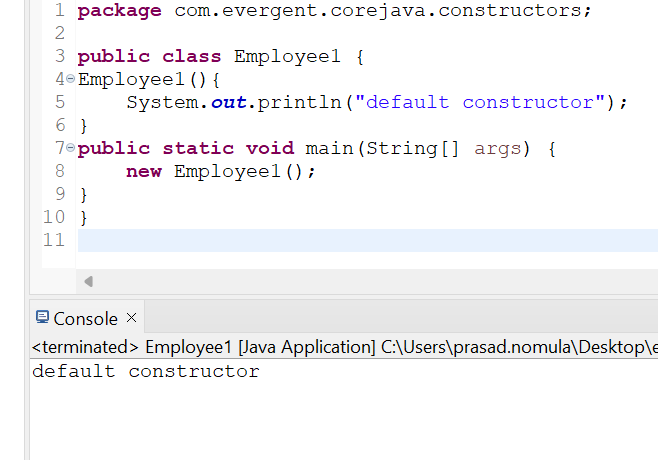
vi. Every class needs atleast 1 default constructor

vii. this, super This

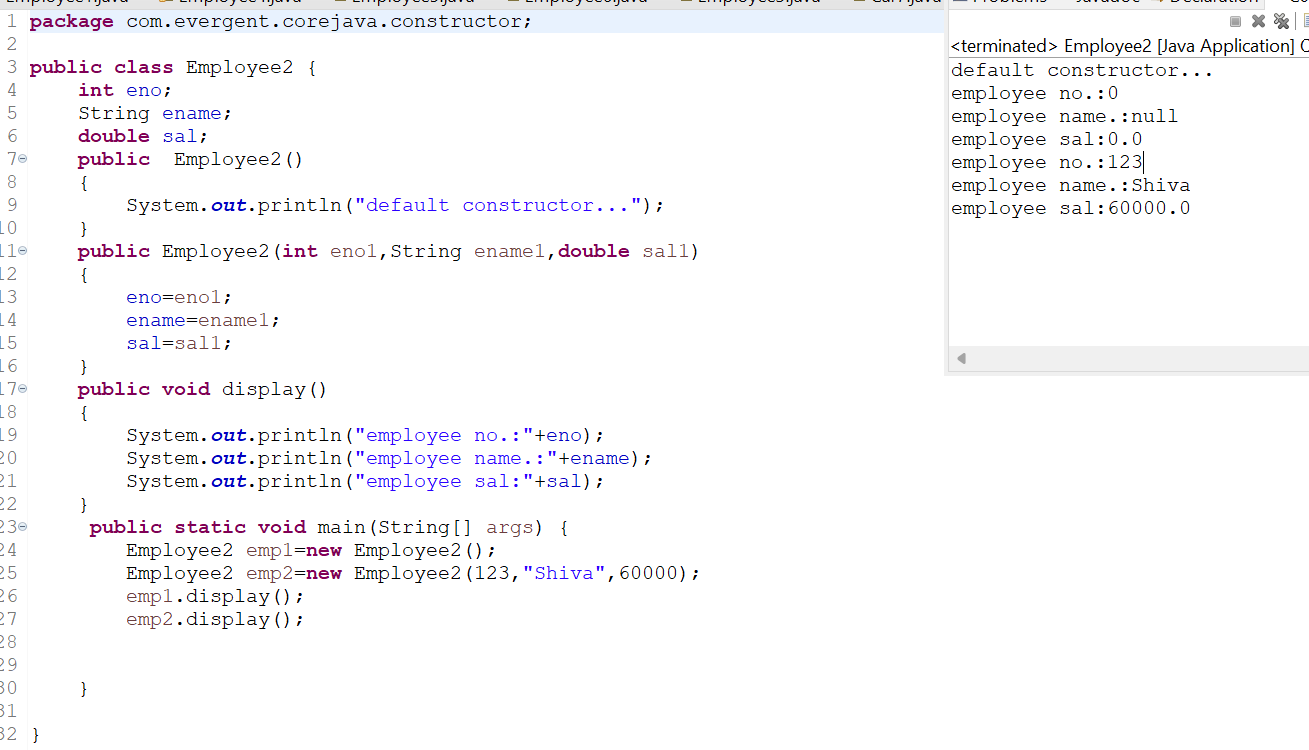
--> this is a keyword always refers to instance variables

viii. Always constructor are overloaded

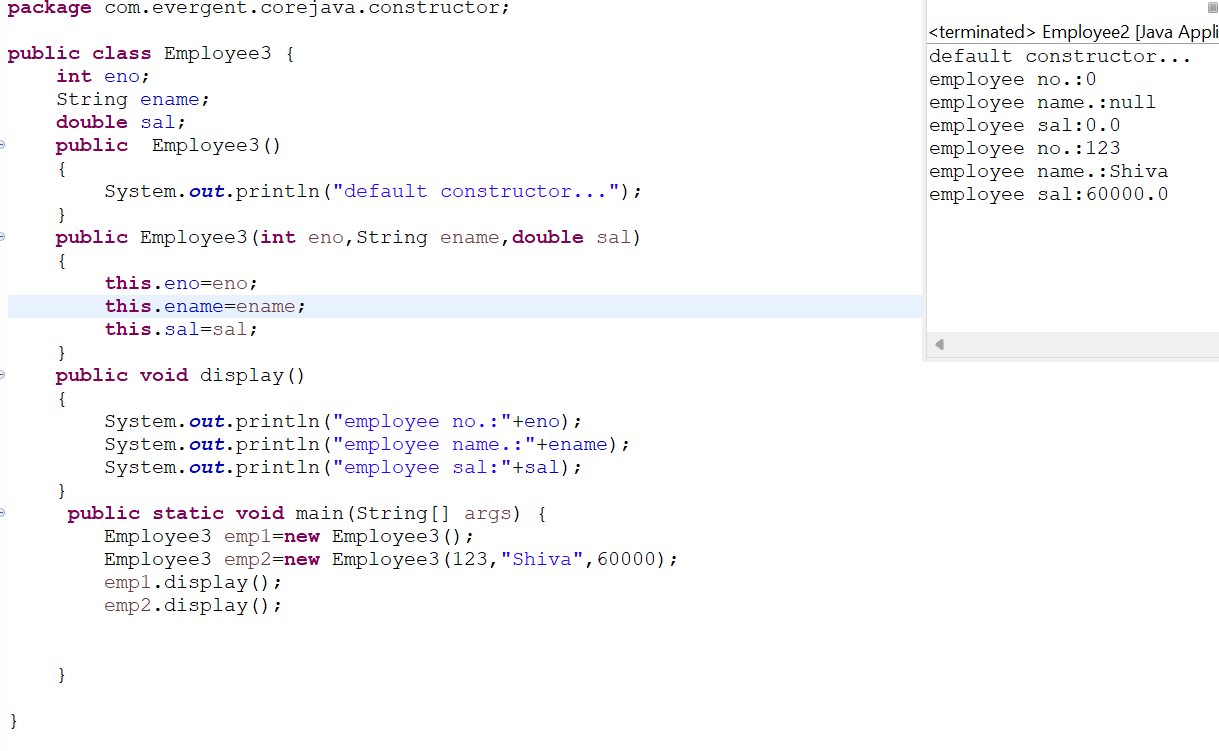
**Program1:**



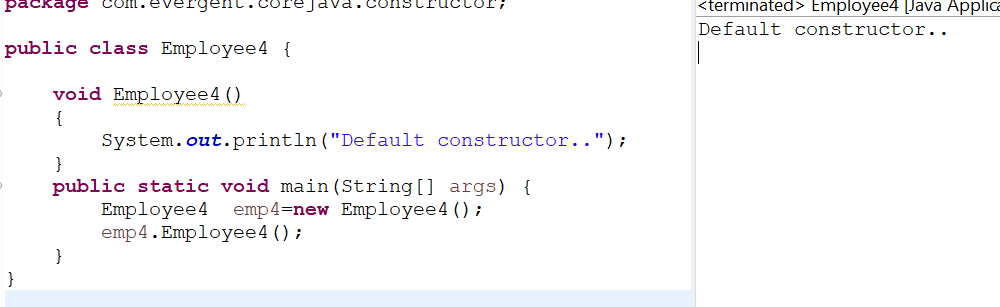
**Program2:**

****

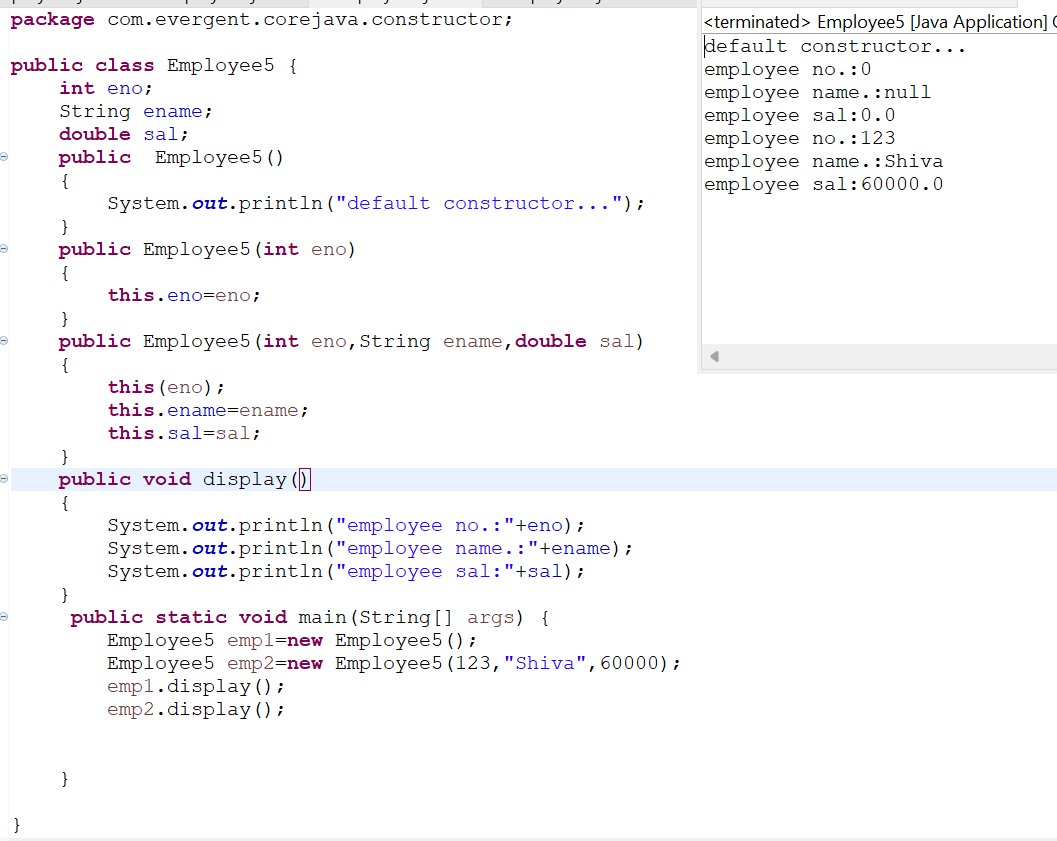
**Program3:**

****

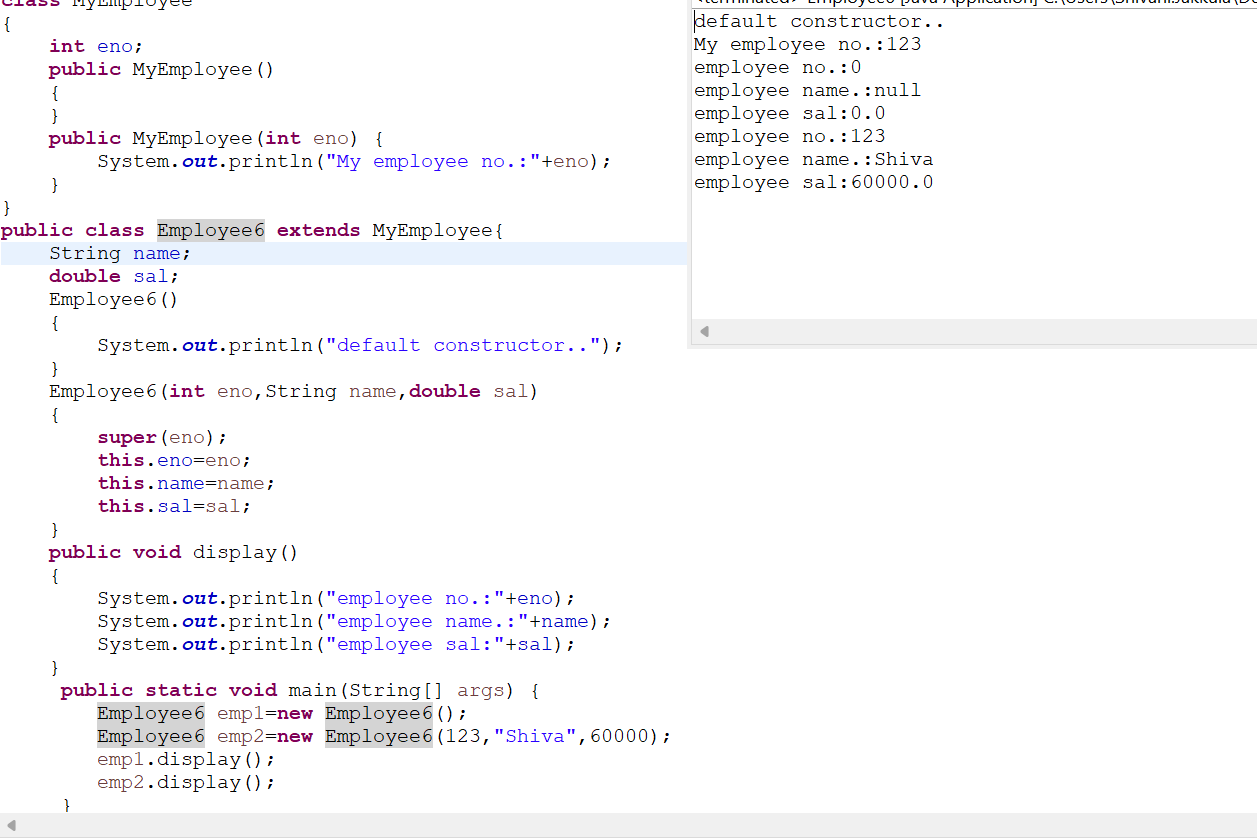
**Program4:**

****

**Program5:**

****

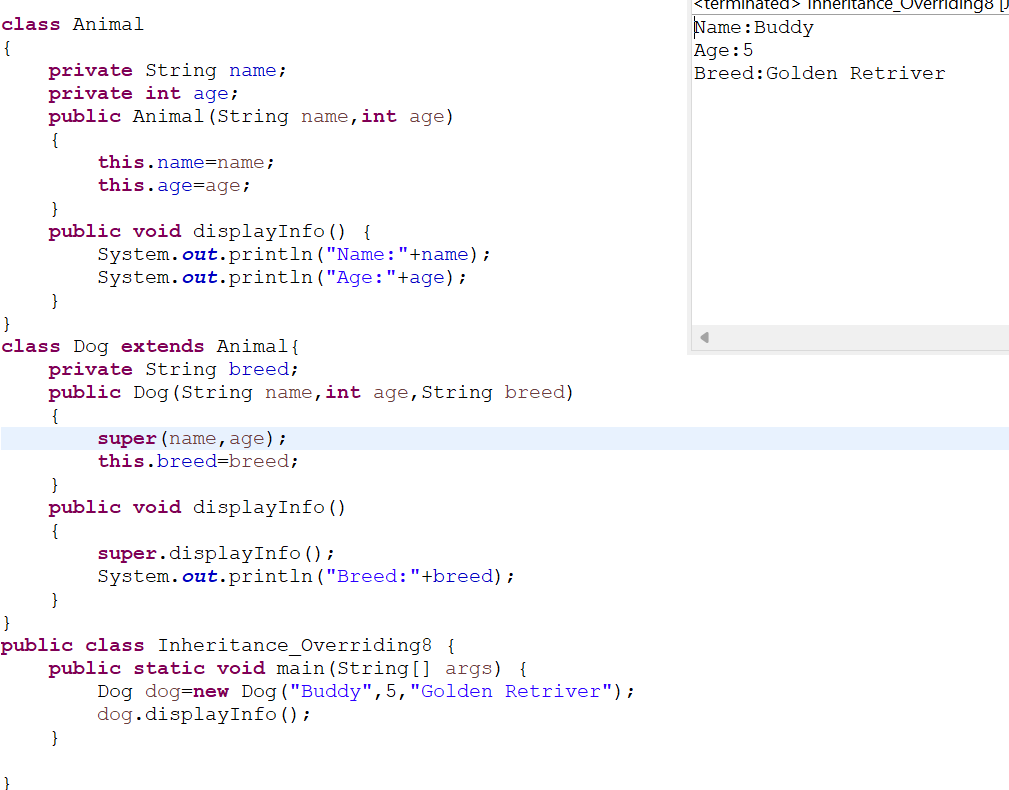
**Program 6:**

****

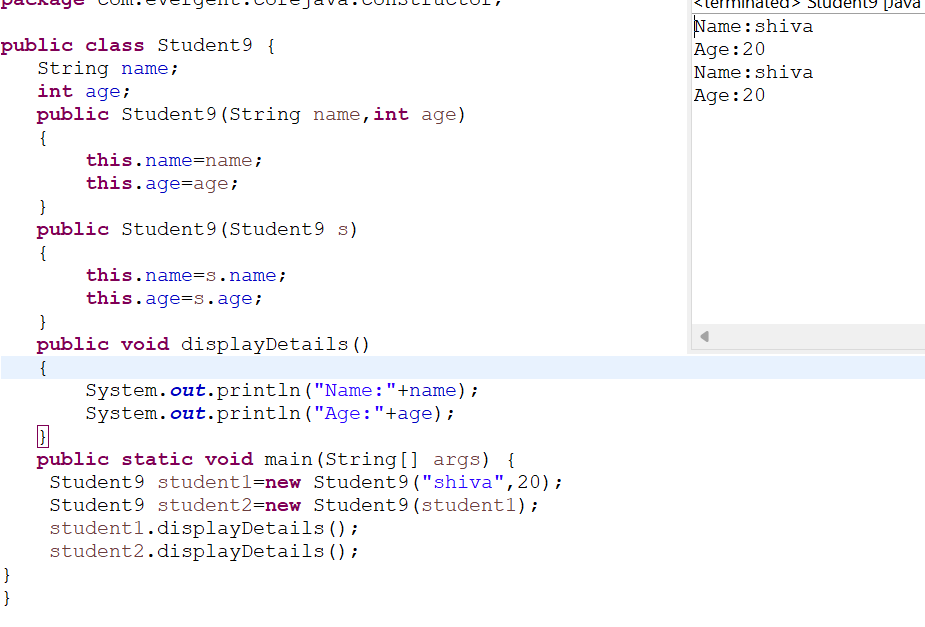
**Program 7:**

****

**Program8:**

****

**Program9:**

****

**Date:09/08/2024 - Day5**

1. **Static**

a. Static is a keyword

b. We can declare variables and methods as static

c. We can access static variables and static methods directly through calssname.methodname and classname.variablename respectively.

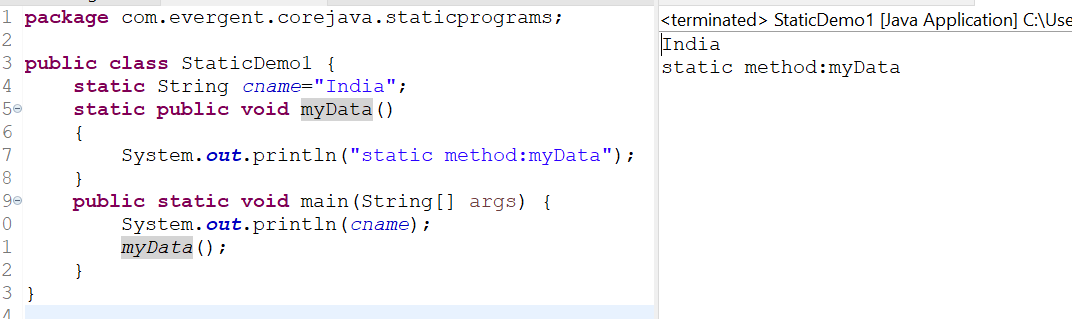
d. Static methods can access static methods and static variables only.

e. Static methods cannot access non static methods and non static variables.

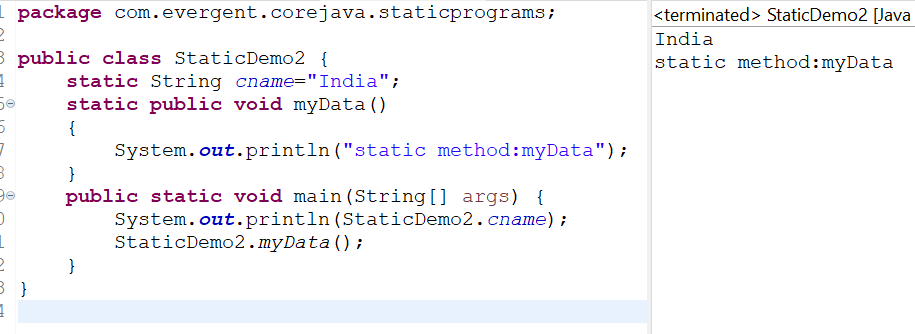
f. Non static methods can access static methods and static variables.

g. Static block- whenever class is loaded inside the JVM at that time static block is initiated.

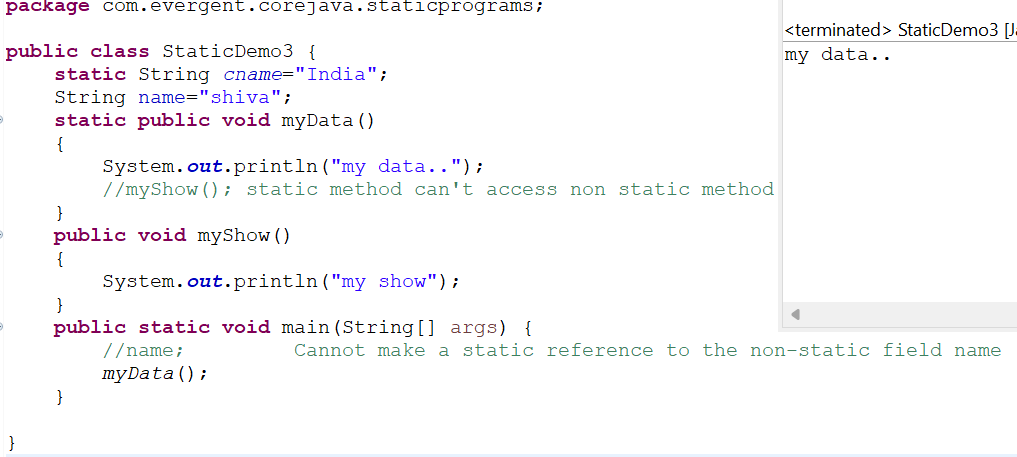
**Program1:**

****

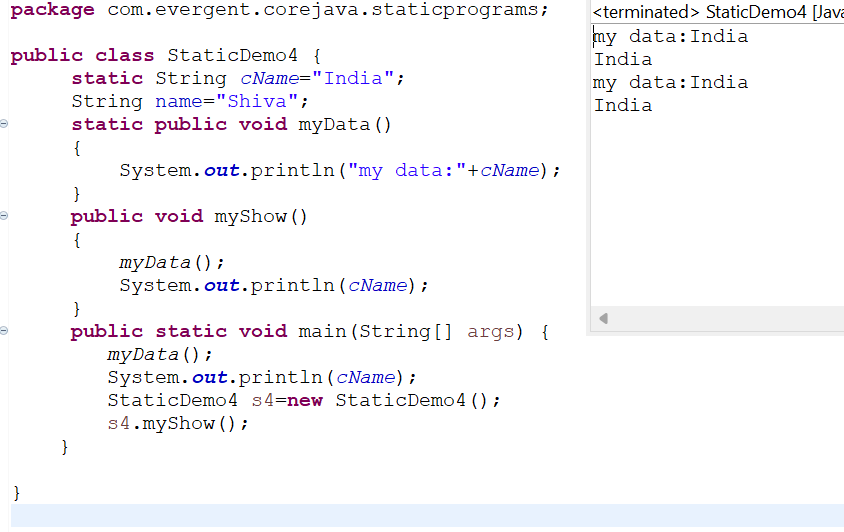
**Program2:**

****

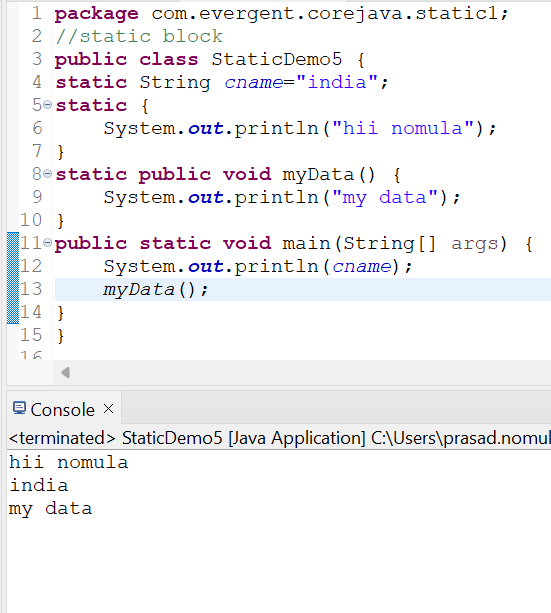
**Program3:**

****

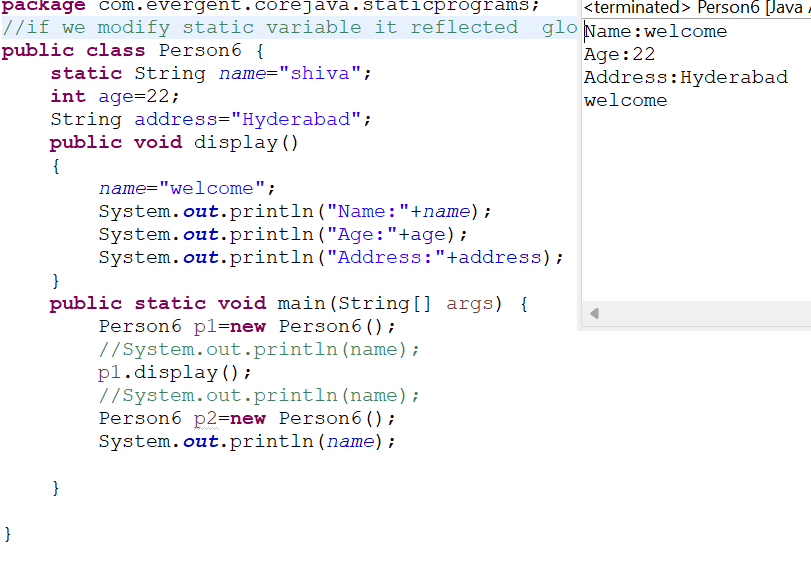
**Program4:**

****

**Program5:**



**Program6:**

****

**2. Final**

a. Final is a Keyword.

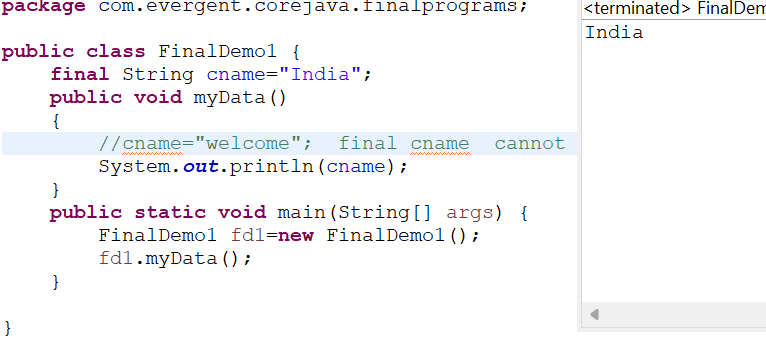
b. We can declare a variable, method, or a class as final.

c. Final variable cannot be modified.

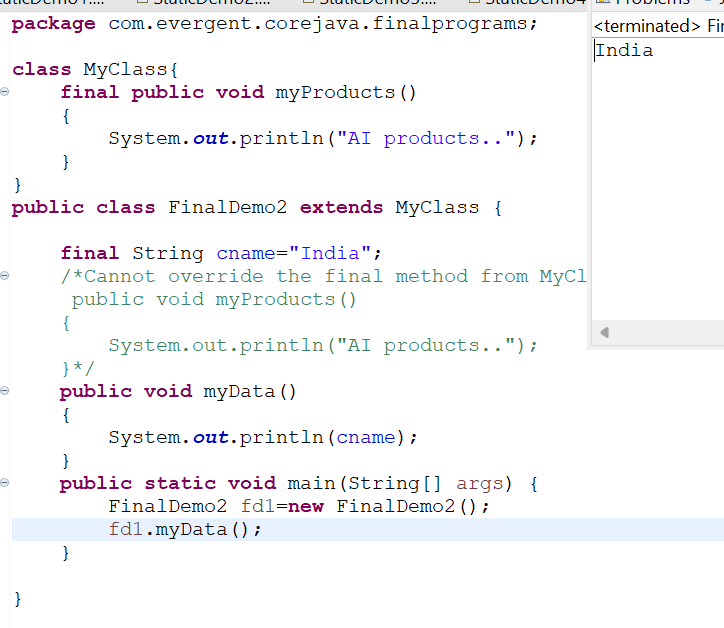
d. Final Method cannot be overrided.

e. Final class cannot be inherited.

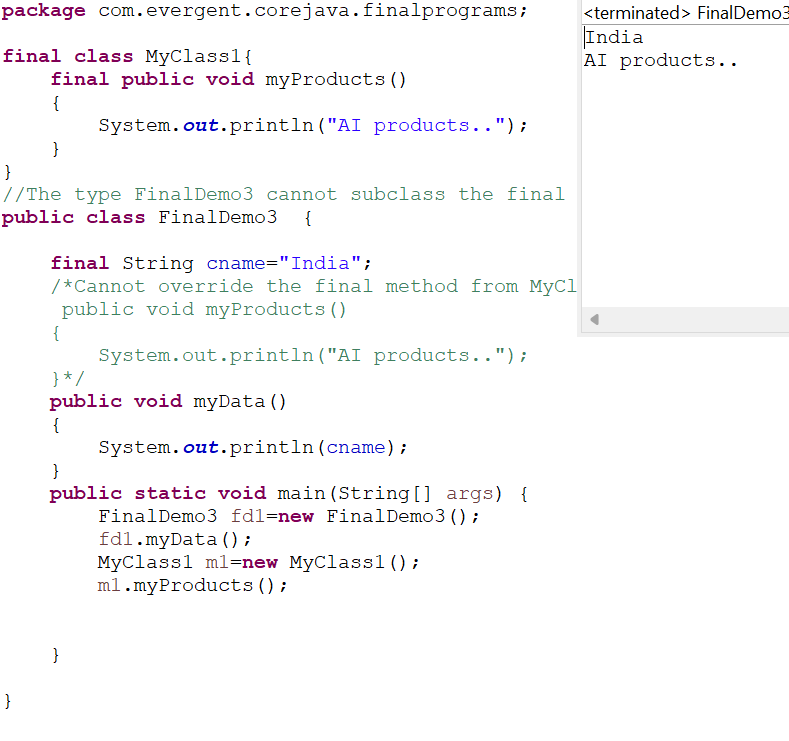
**Program1:**

****

**Program2:**

****

**Program3:**

****

**12/08/24 :**

**Strings:**

**-Why string is immutable?**

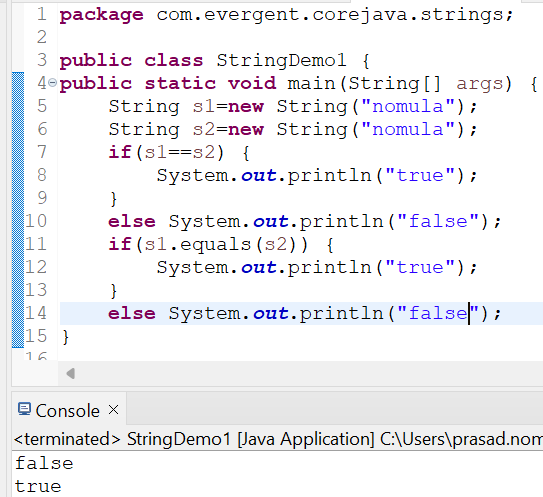
1. String is a final class
2. Strings are immutable
3. Strings having methods
4. All methods are non-synchronized

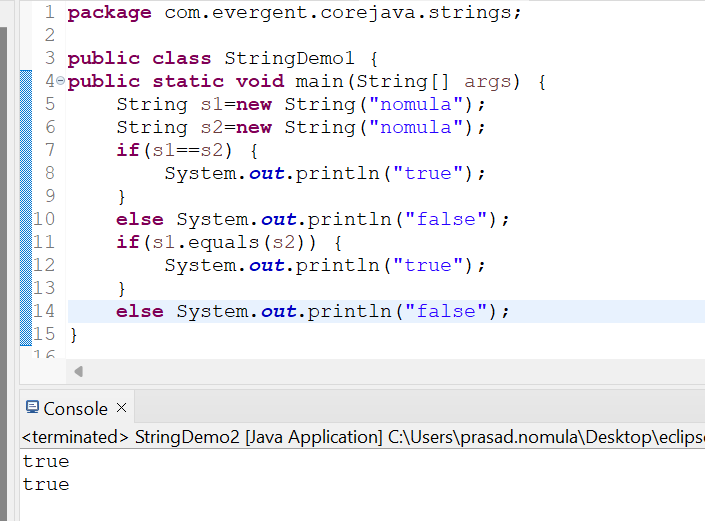
**StringBuffer:**

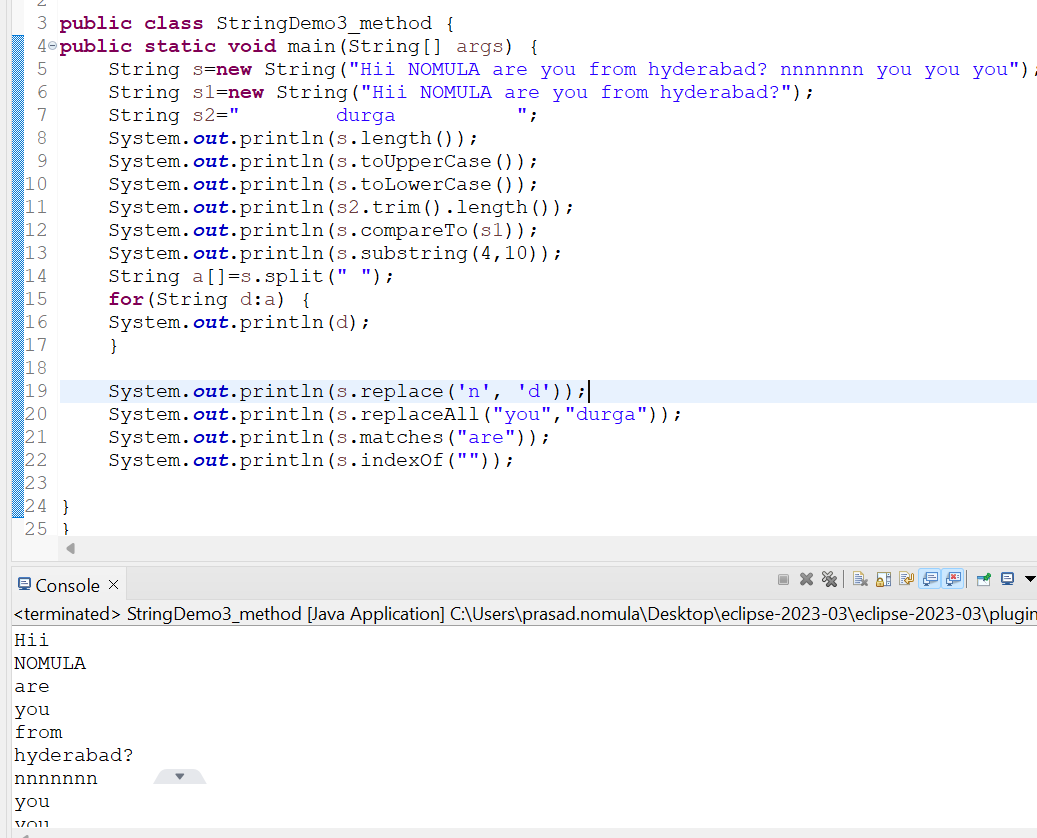
1. String buffer is a final class
2. String buffer is mutable
3. String buffer having methods
4. All methods are synchronized

**String Builder:**

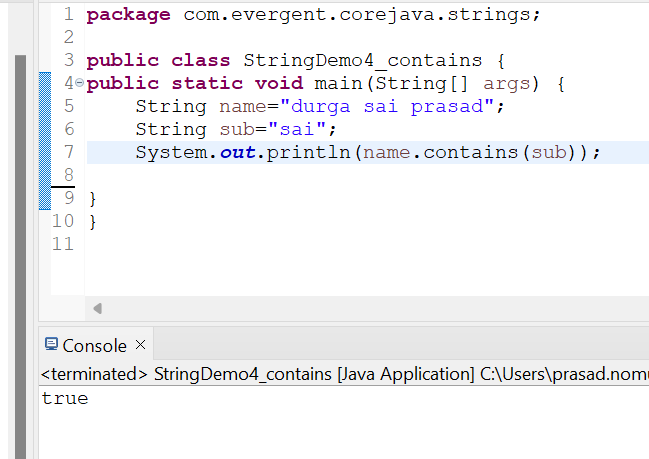
1. String builder is a final class
2. String builder is mutable
3. String builder having methods
4. All methods are non-synchronized



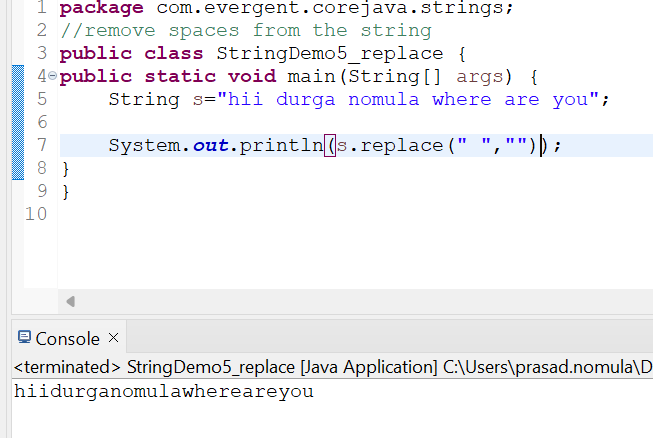




1.create a java program that creates a string and checks if it contains specific subString and then prints out the result

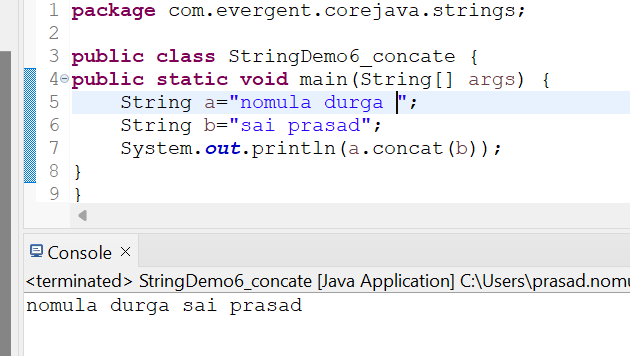


1. Write a java prgm to create a String ,remove all spaces from the string and then print out the result

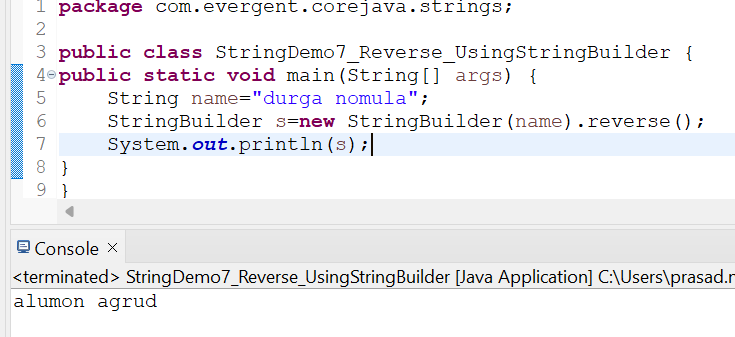


String concatination:

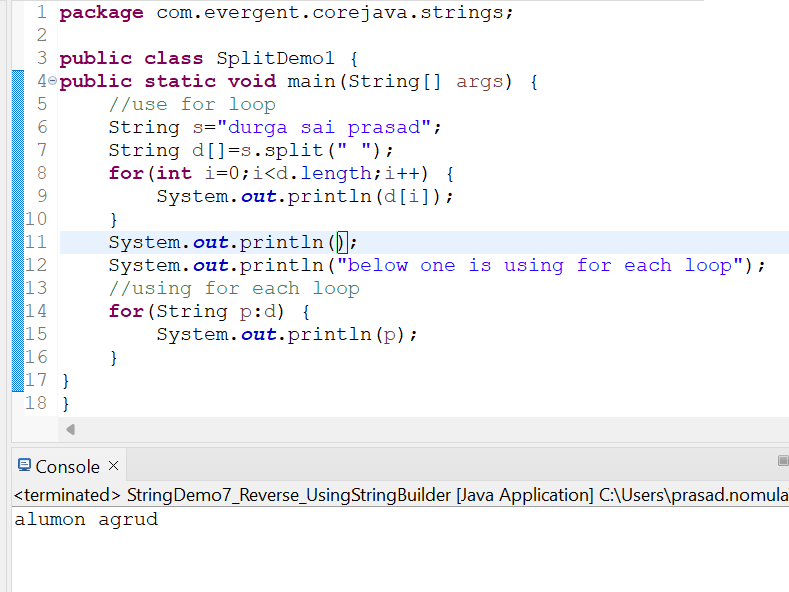
Strings can be concatenated using + operator (or) concat.



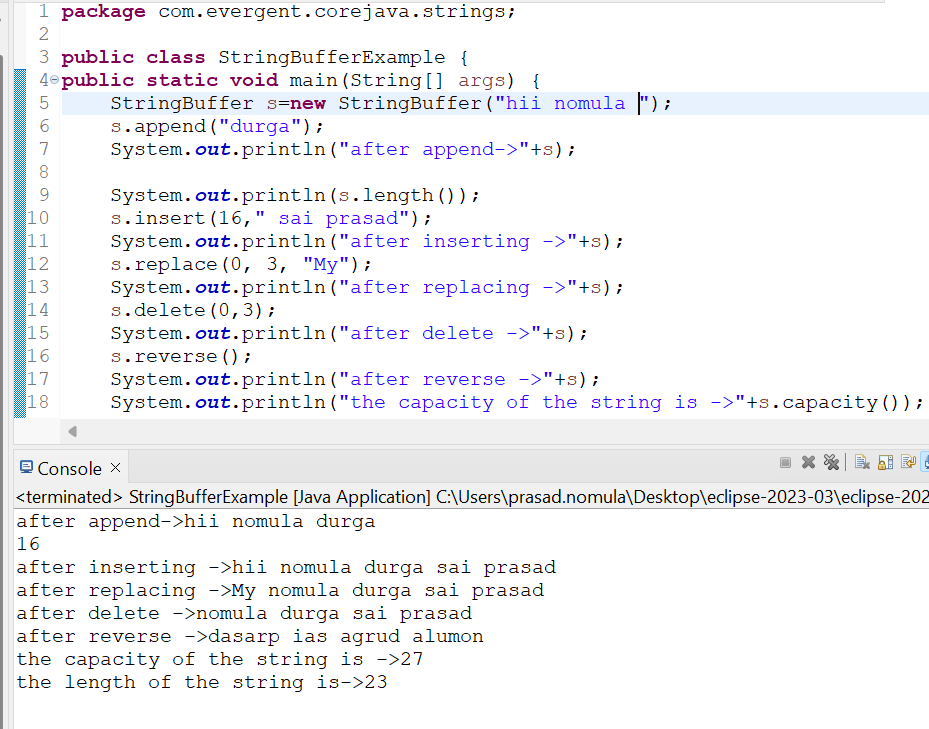
Reverse of a String:



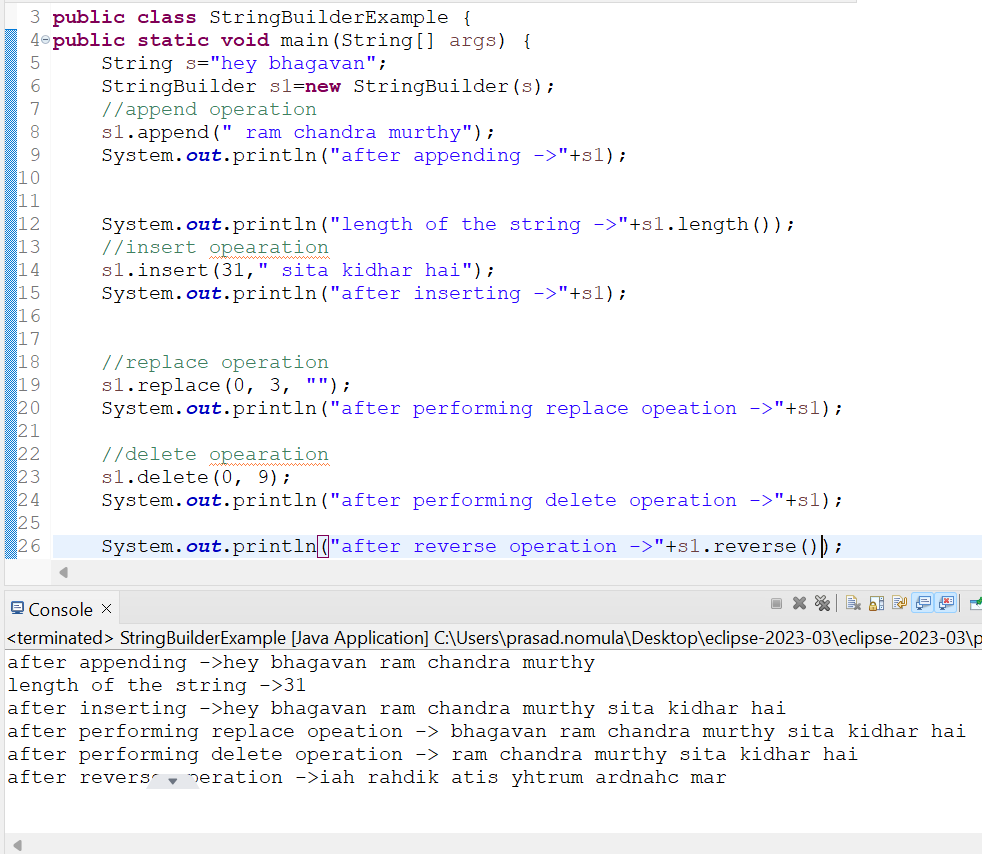
Splitting of any sentence (or) text on spaces and using for each loop



**METHODS IN STRINGBUFFER**



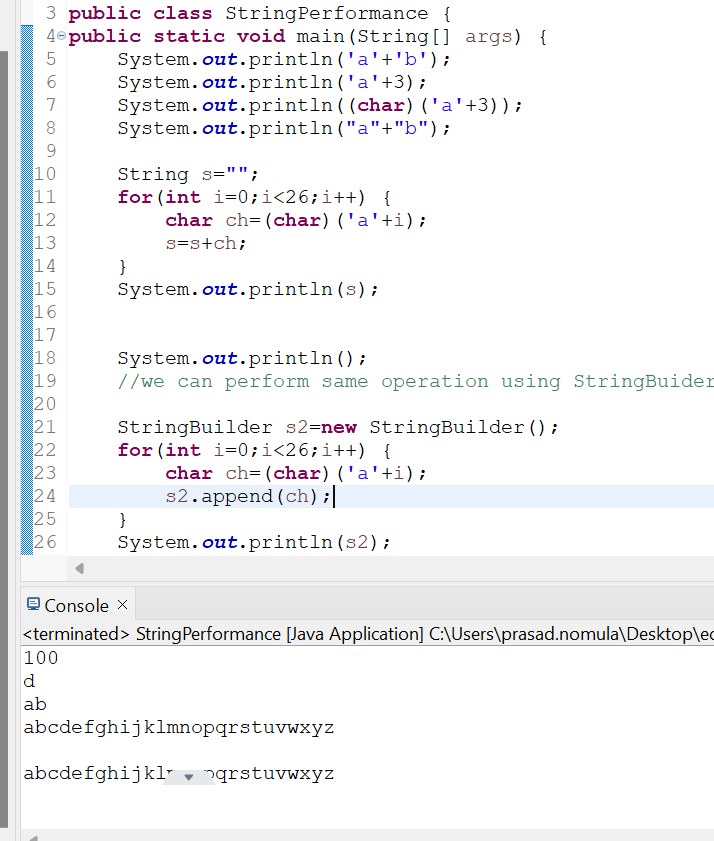
**METHODS IN STRING BUILDER**

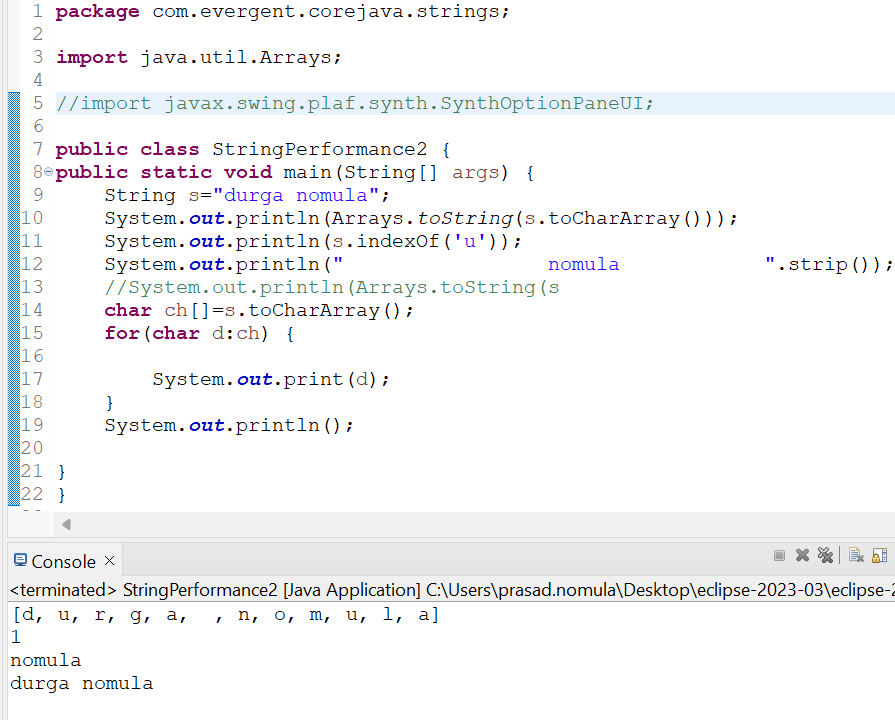


**String class important points:**

1. In java a string is a sequence of characters ,often used to represent text.
2. Strings are objects in java and are instances of the string class,which is part of the java java.lang package
3. Key features of strings in java:
4. Immutable:once a string object is created ,it cannot be changed .
5. Java optimizes memory usage by storing strings in special area of memory as string pool
6. If two strings have the same value and are created without using new keyword they will refer to same object in the stringpool.
7. We can create a string in java in multiple ways:
8. Using string literals :str=”hello world”;
9. Using the new keyword

String str=new String(“hello, world”);





**13/08/24-Day7**

1. **Can you make class as immutable?**

We can make our class as immutable by declaring class as final and attributes are private and final

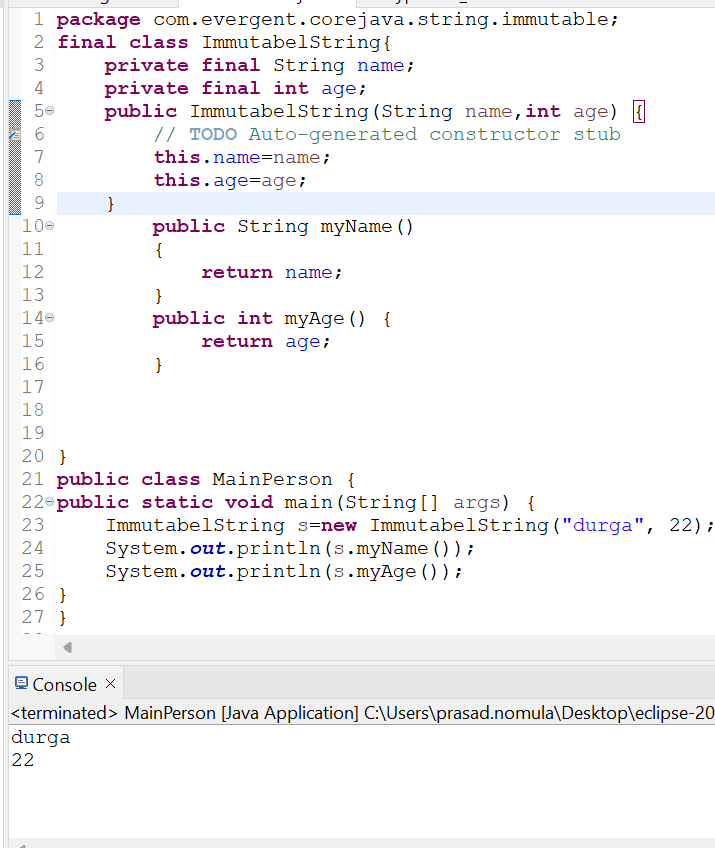
We can declare class as our own immutable class

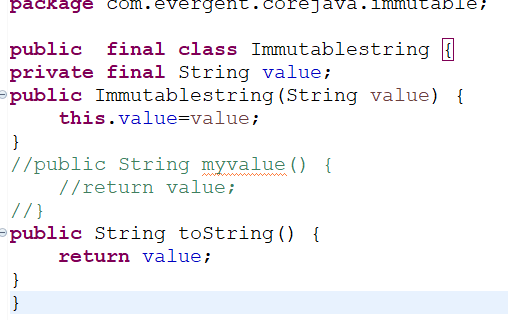
1. we can declare class as final
2. The classs is declared as final so that it cannot be subclass
3. Private Final class:
4. The fields name and age are private and final

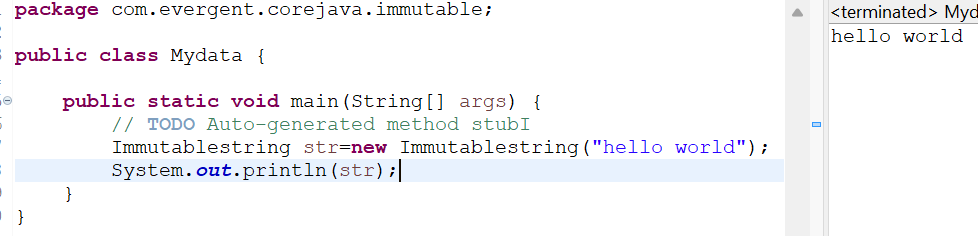
Constructor:

1. The constructor initialies the final fields when a person object is ceated

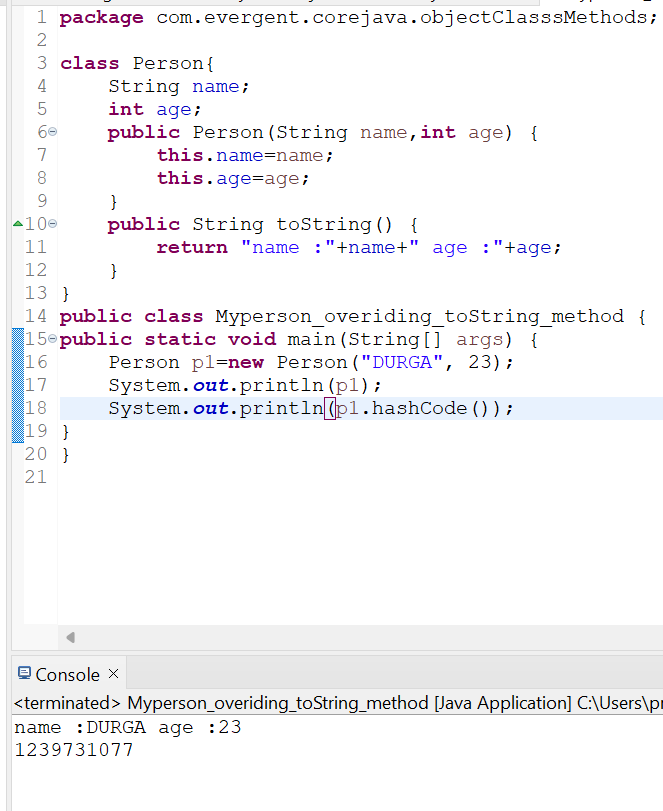
**Immutable class:**

****



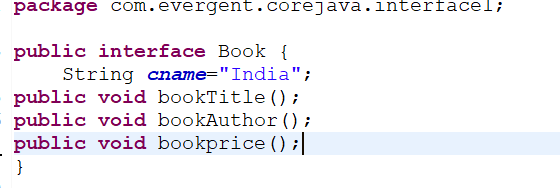


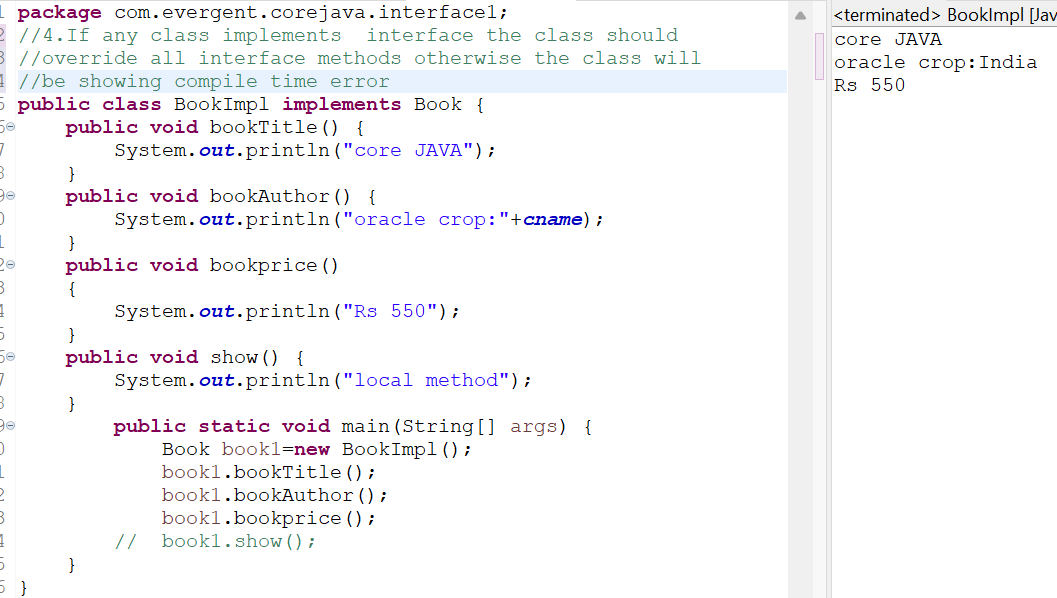
**toString and objectclassmethods:**

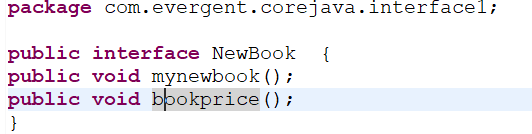


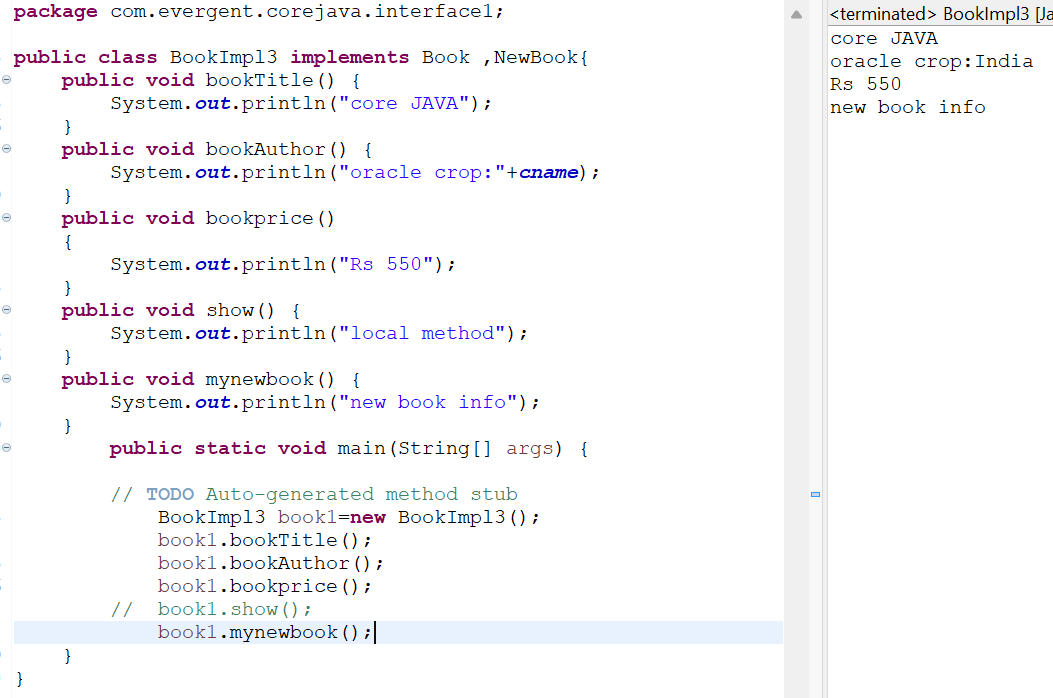
**Interfaces:**

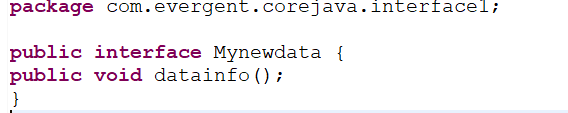
1. Interface is a keyword
2. We can declare method signature only but not implementation
3. By default all interfaces methods are abstract
4. We cannot create object but we can create reference to interface
5. By default variables inside interface is public static final
6. Java will support multiple inheritance through interfaces
7. One class can implements multiple interfaces
8. One interface can extend other interface
9. In interface we can create complete methods also but we cant create object to that methods we can call that method with the help of other class object only for that purpose we want to impletments with another class

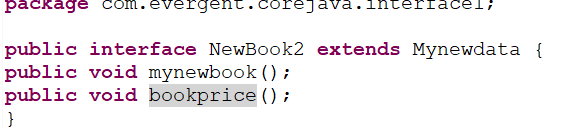


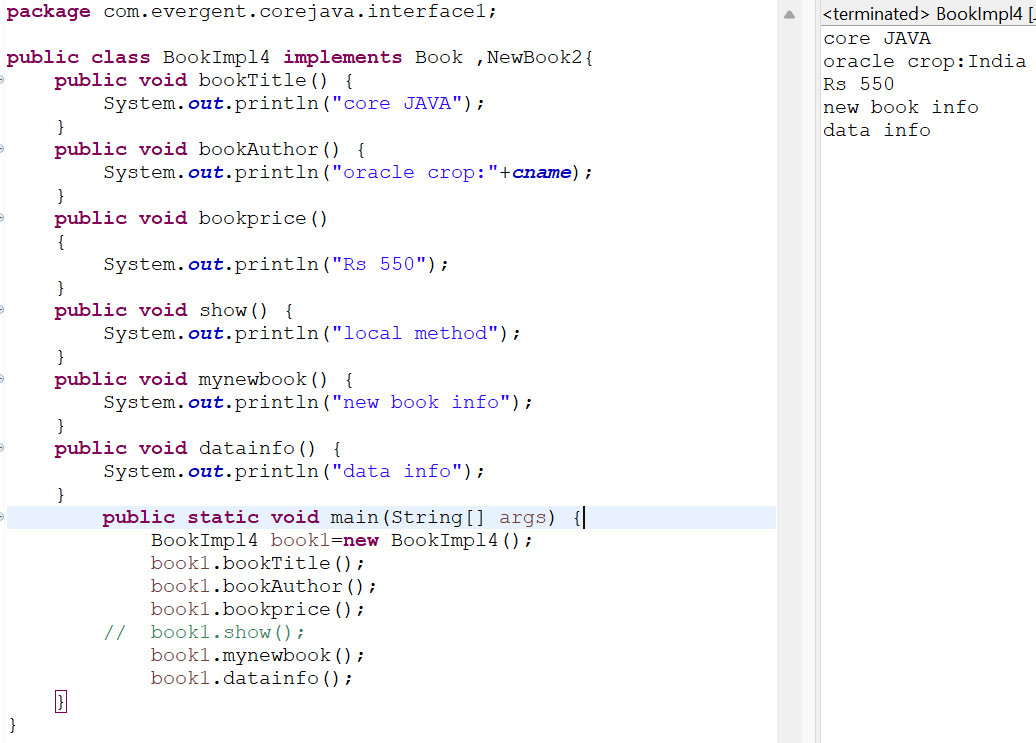










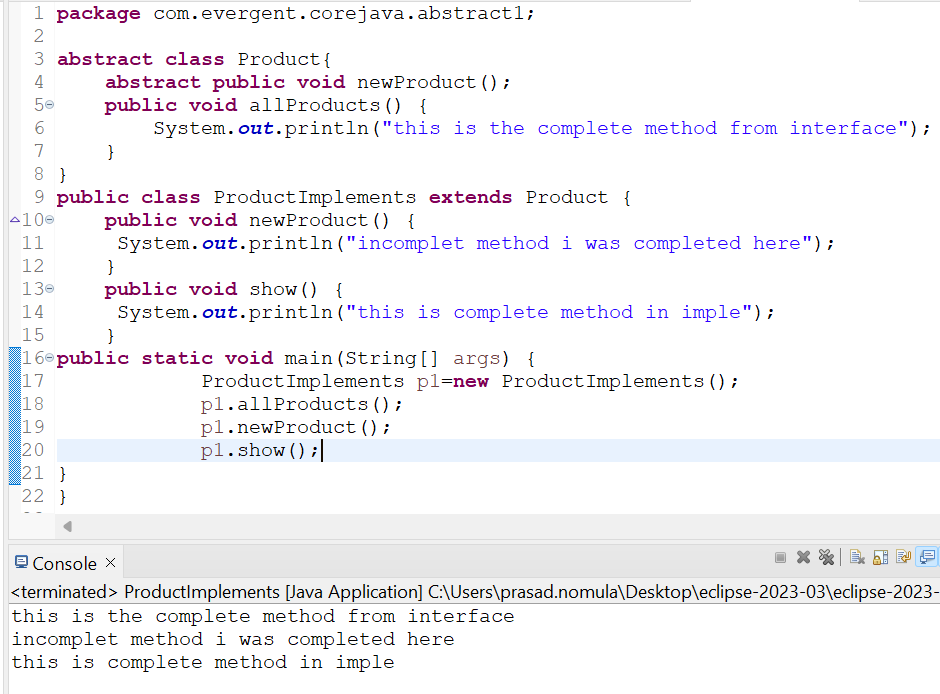


**14-08-2024(DAY-08)WED**

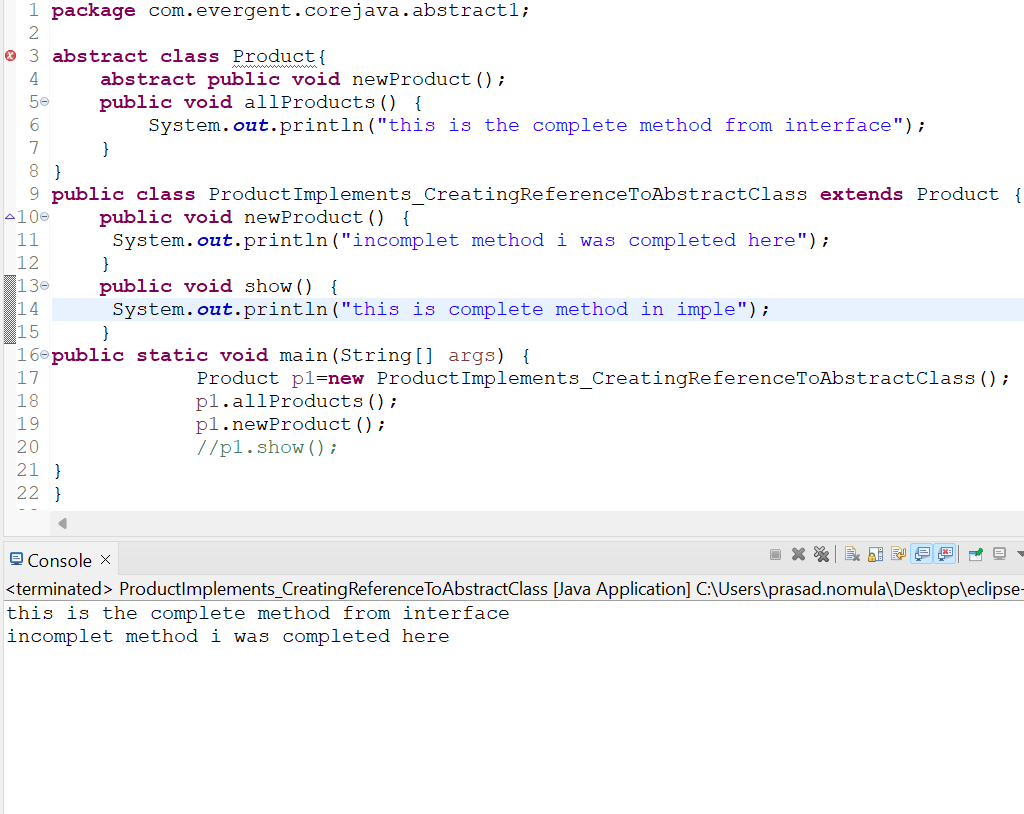
**ABSTRACT CLASS**

1. abstract is a keyword
2. Abstract class have abstract methods and concrete methods(implemented).
3. If any class have abstract method that class should declared as abstract class otherwise the class will be showing compile time error
4. If any class extends abstract class that class should be overide all abstract methods otherwise the class will showing error
5. We cant create object to abstract class but we can create reference to abstract class
6. If we declare abstract to class eventhough it doesn’t contain abstract class then we cant create object ot this class.this process we use to do secure purpose
7. We can create constructor to abstract class
8. We can access abstract class constructor through subclass object creation

PROGRAM-1



PROGRAM-2



**16/08/2024 DAY-09(FRI)**

* Conducted exam based on oops concept
* **D**iscussed some questions based on oops concept
* Explained Application-3 and analyzed that code.

**19/08/2024 DAY-10(MON)**

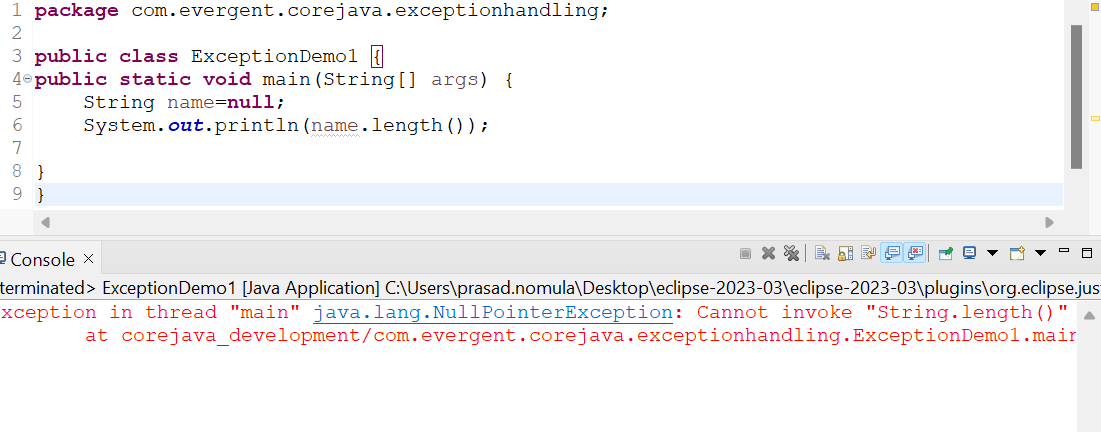
**EXCEPTION HANDLING**

* Exception handle is a mechanism.
* Exception are inbuilt mechnasims of java
* All exceptions are excuted with with abnormal conditions only
* Normal flow it wont excute any exceptions
* Once any exception are occuring in java the remaining lines of code is unreachable
* Java.lang.Throwable is a super class for exception and errors
* They are 2 types of exception.

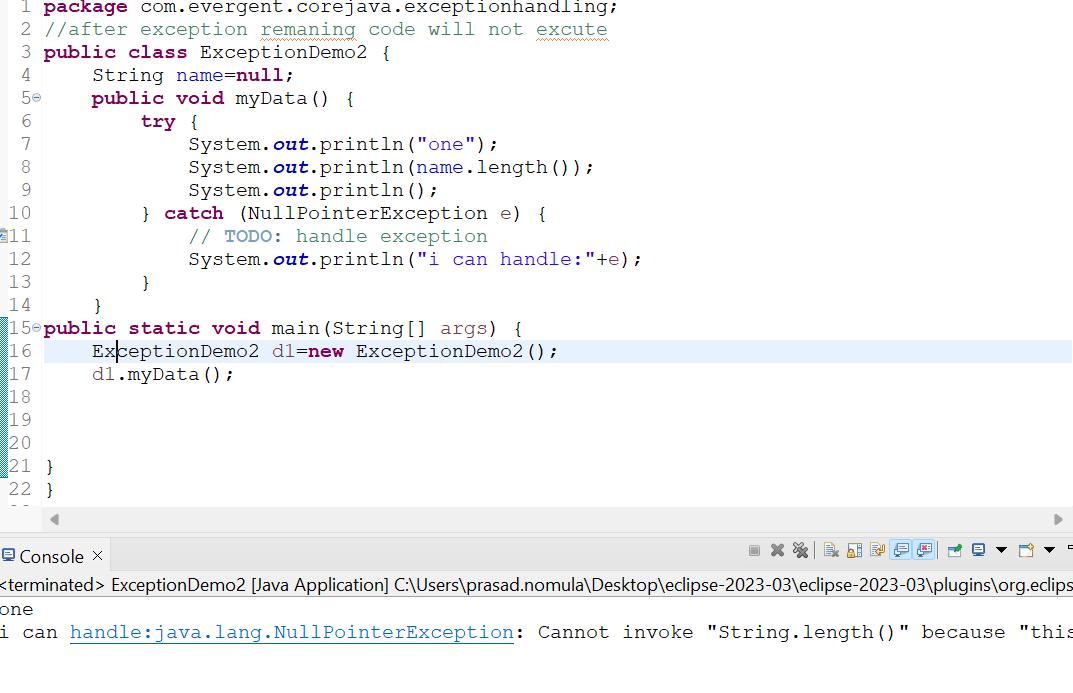
1. checked exception
2. Unchecked exception

* All checked exceptions are compile time exceptions
* All runtime exceptions are runtime exceptions
* They are 5 keywords
  + Try
  + Catch
  + Finally
  + Throw
  + Throws
* Try is for business logic
* Catch is for handling exceptions
* Finally is block,if exception is occurs or not, finally block will excuted
* Throws an exception will be excuted by method by method
* Throw is for runtime and will call predefined exception or user defined exceptions
* Try follows catch or finally block
* We should follow exceptional hierarchical
* We can create our own exceptions(user defined)
* Our own exception extends exception or runtime exception
* All exceptions classes are into java.lang package
* There is 2 exceptions in class,develper should be handled one after one
* Developer cant handle error
* If we write system.exit(0) in this finally block will not excutes

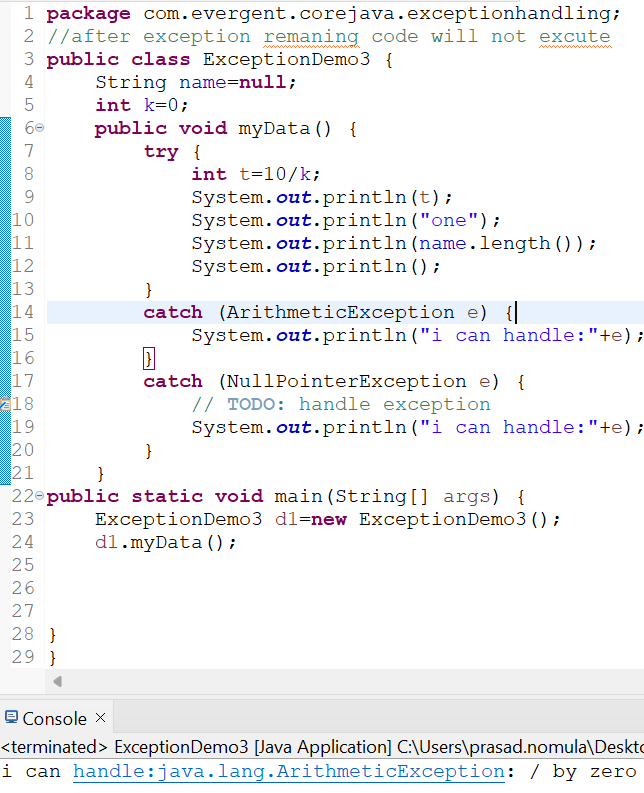
PROGRAM-1



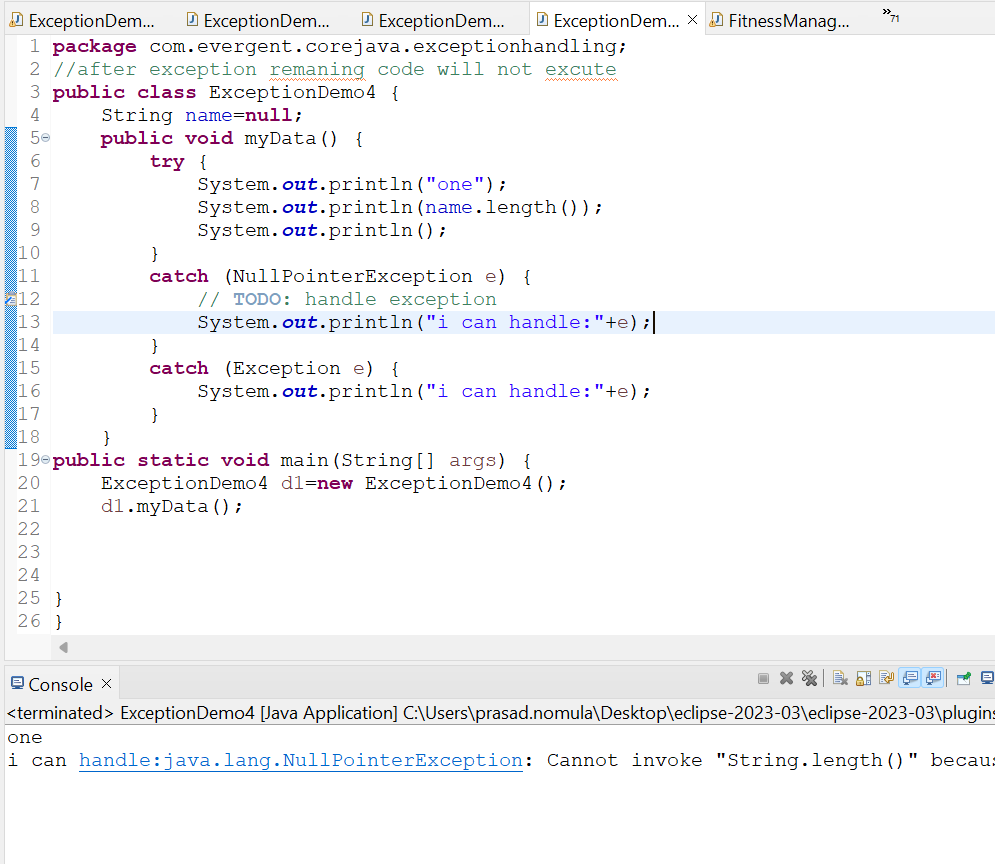
PROGRAM-2



PROGRAM-3



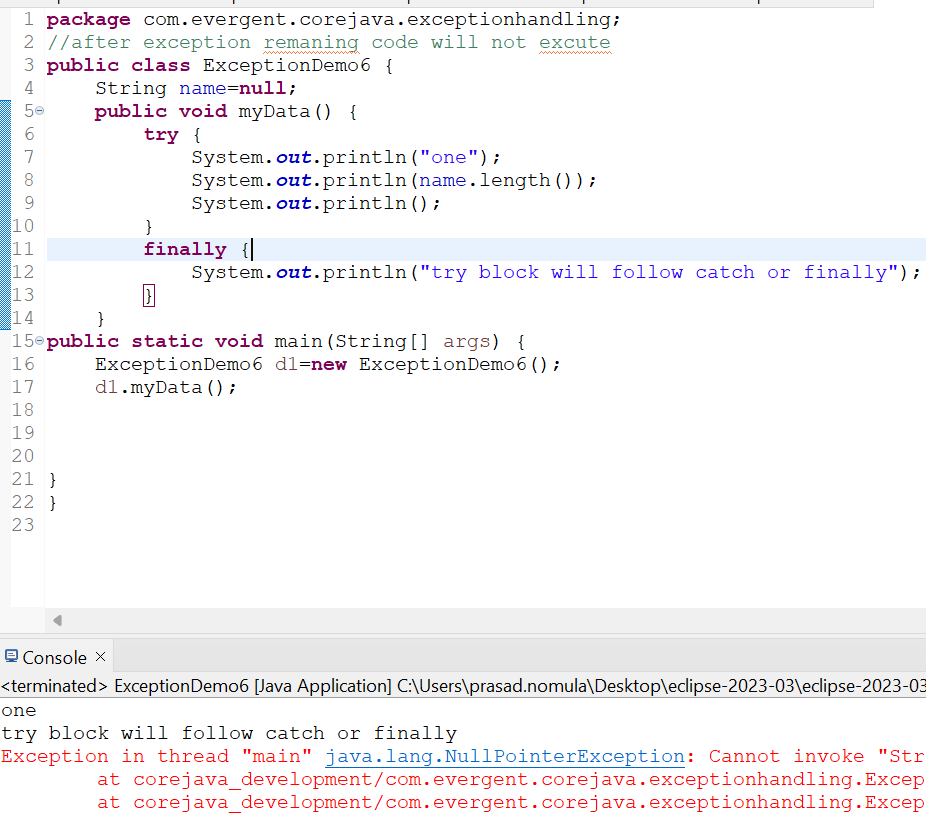
PROGRAM-4



PROGRAM-5



PROGRAM-6



**DAY-11(TUESDAY)20/08/2024**

**MRNG**

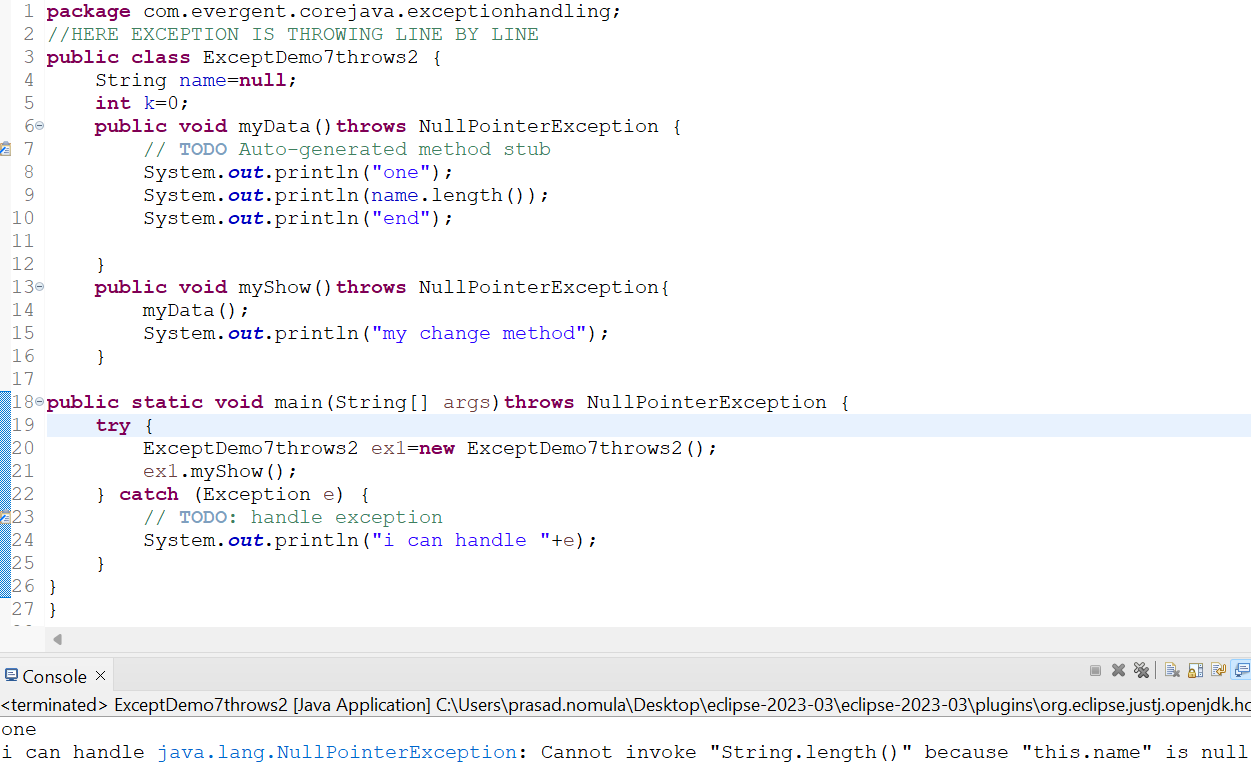
Discussesd throw and throws keyword

Solved some problems

PROGRAM-1



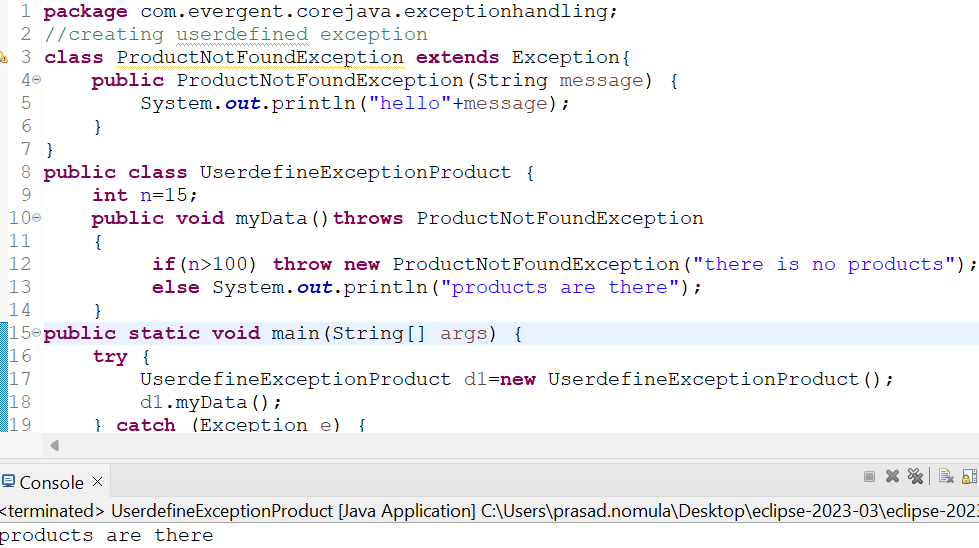
PROGRAM-2



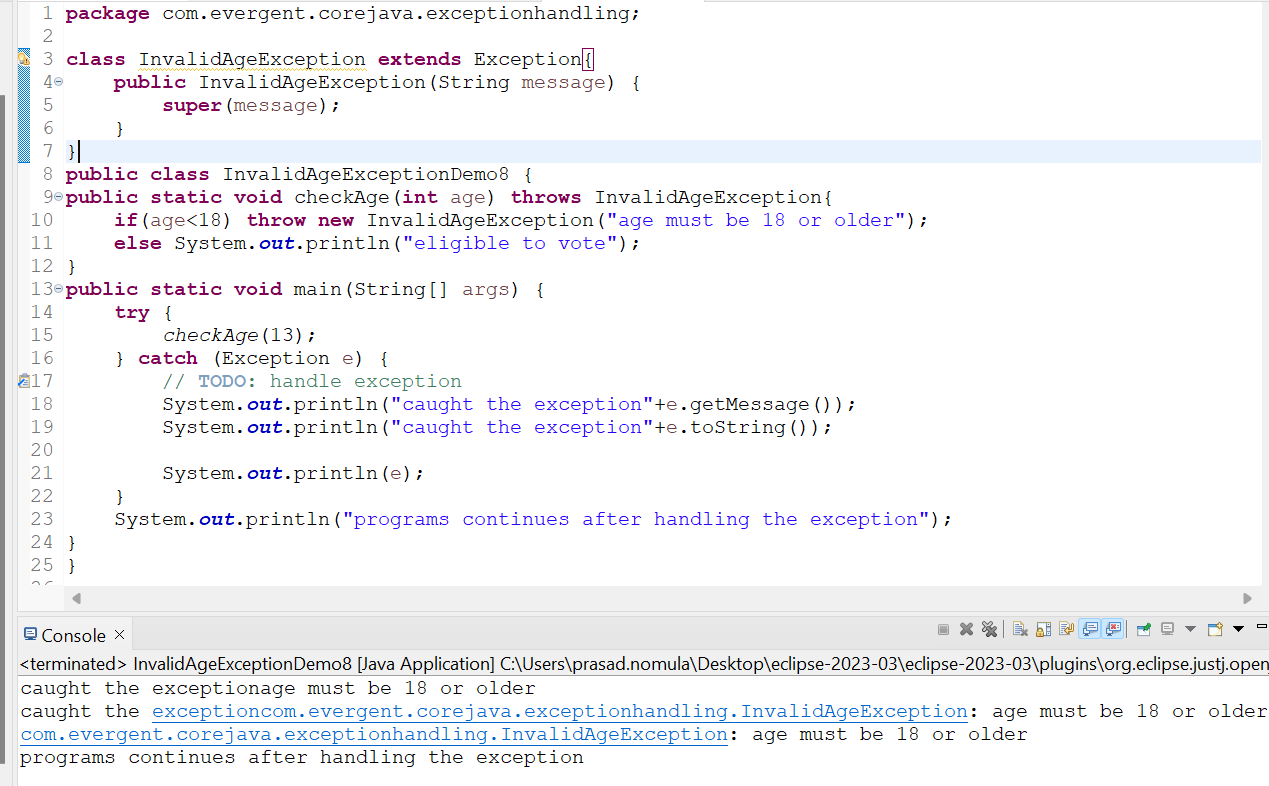
**DAY-12 21/08/2024 Wednesday**

* Discuessed userdefined exceptions using THROW keyword

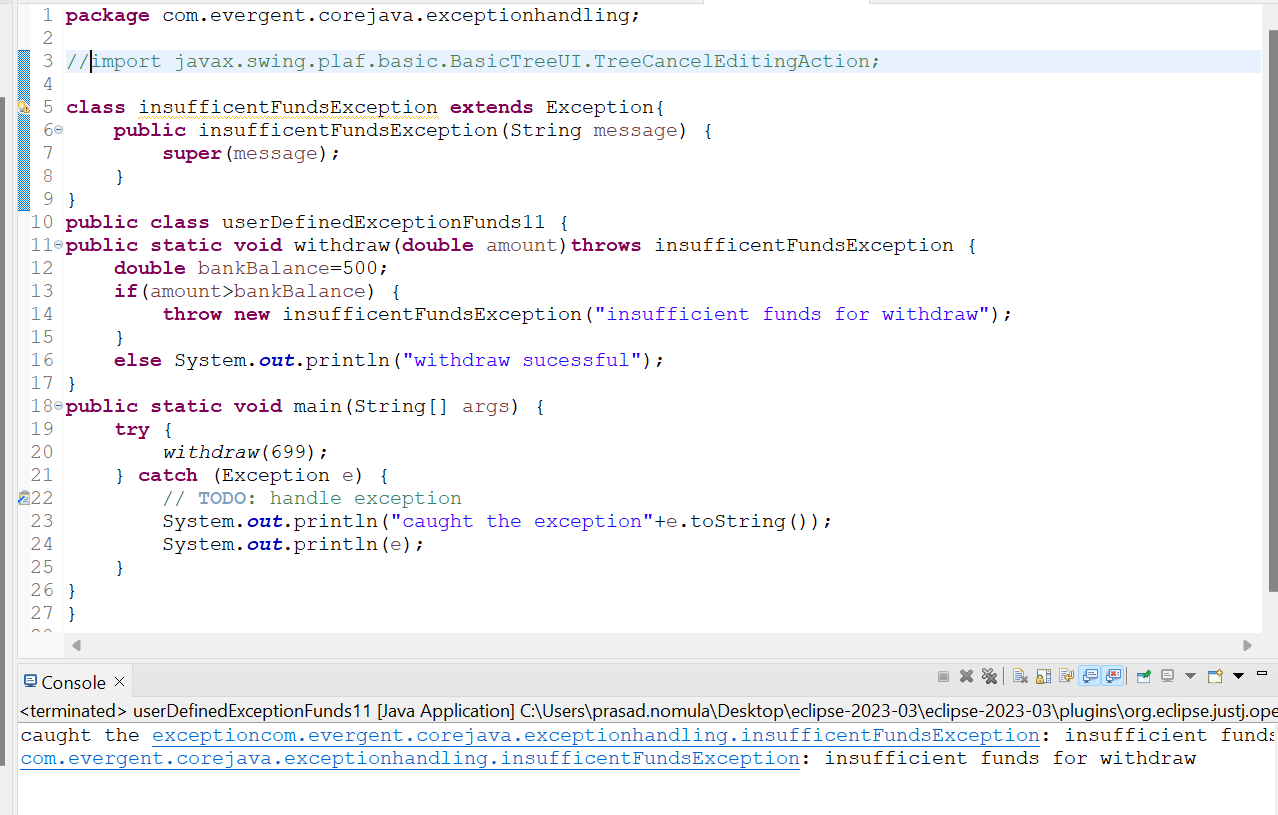
PROGRAM-1



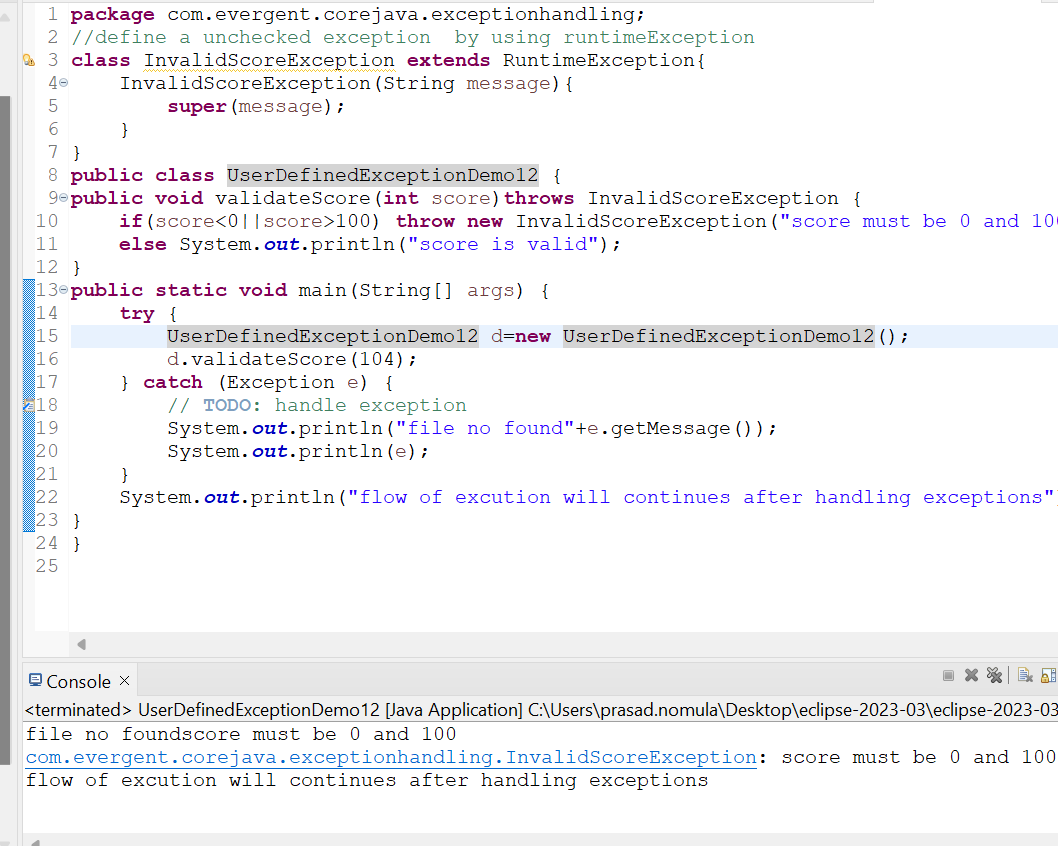
PROGRAM-2



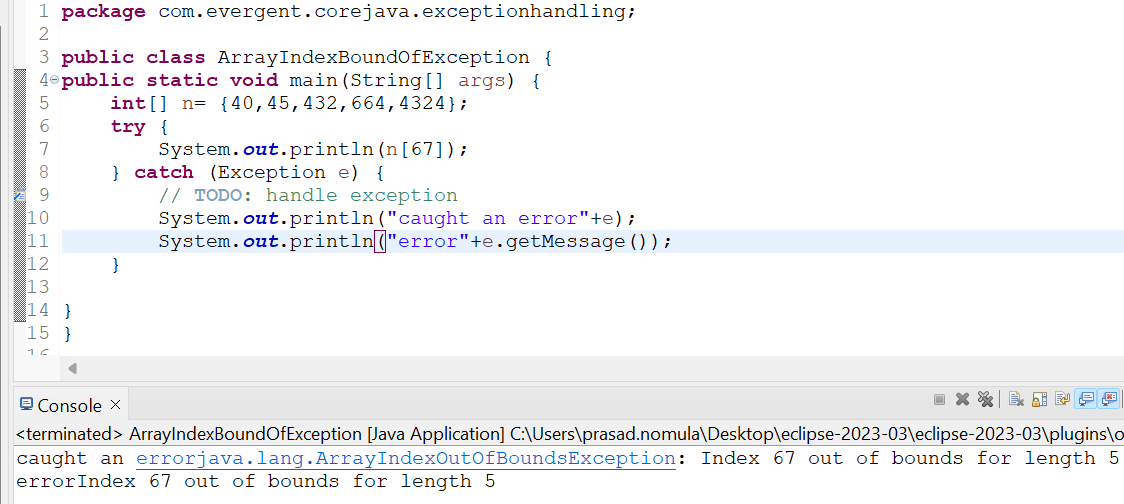
PROGRAM-3



PROGRAM-4



PROGRAM-5



PROGRAM-6

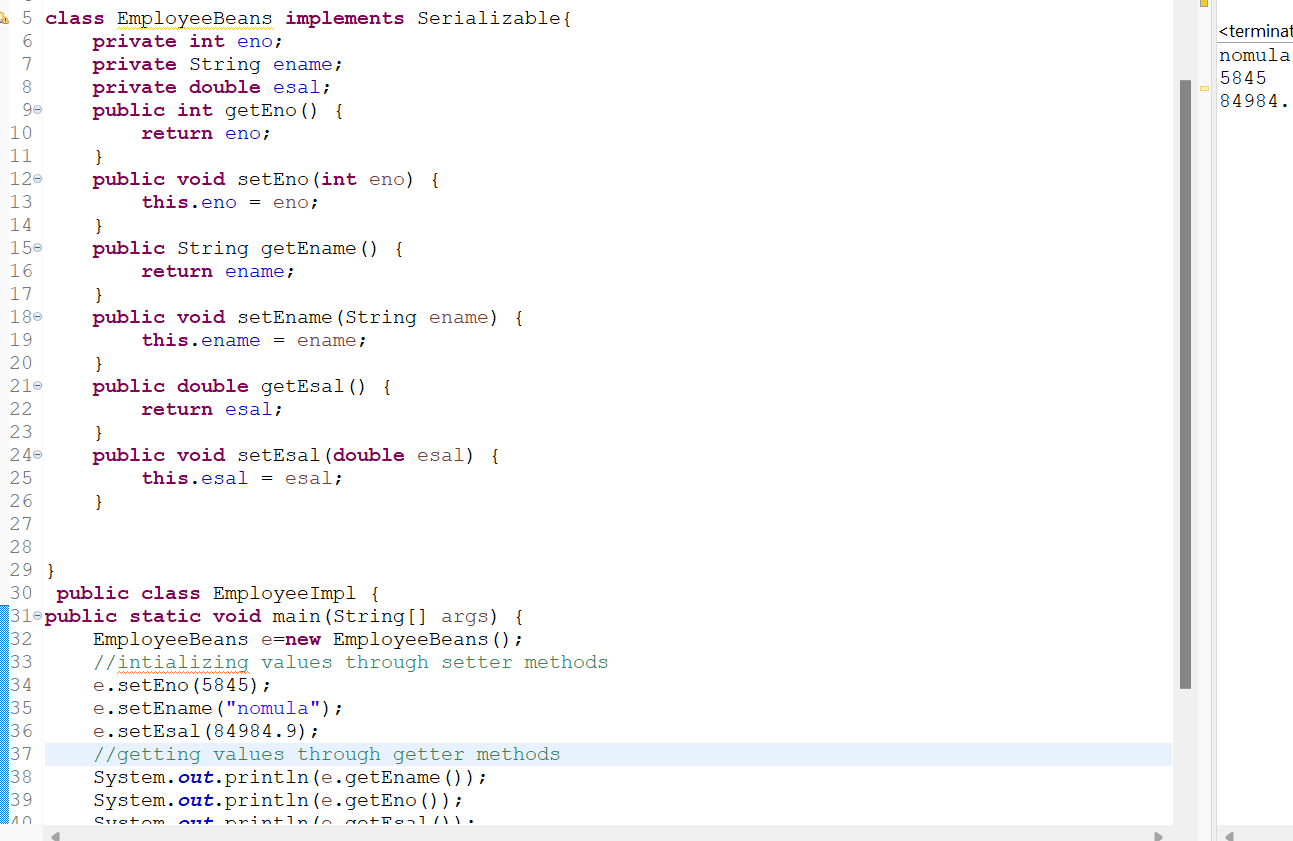


**22-08-2024 AFTERNOON**

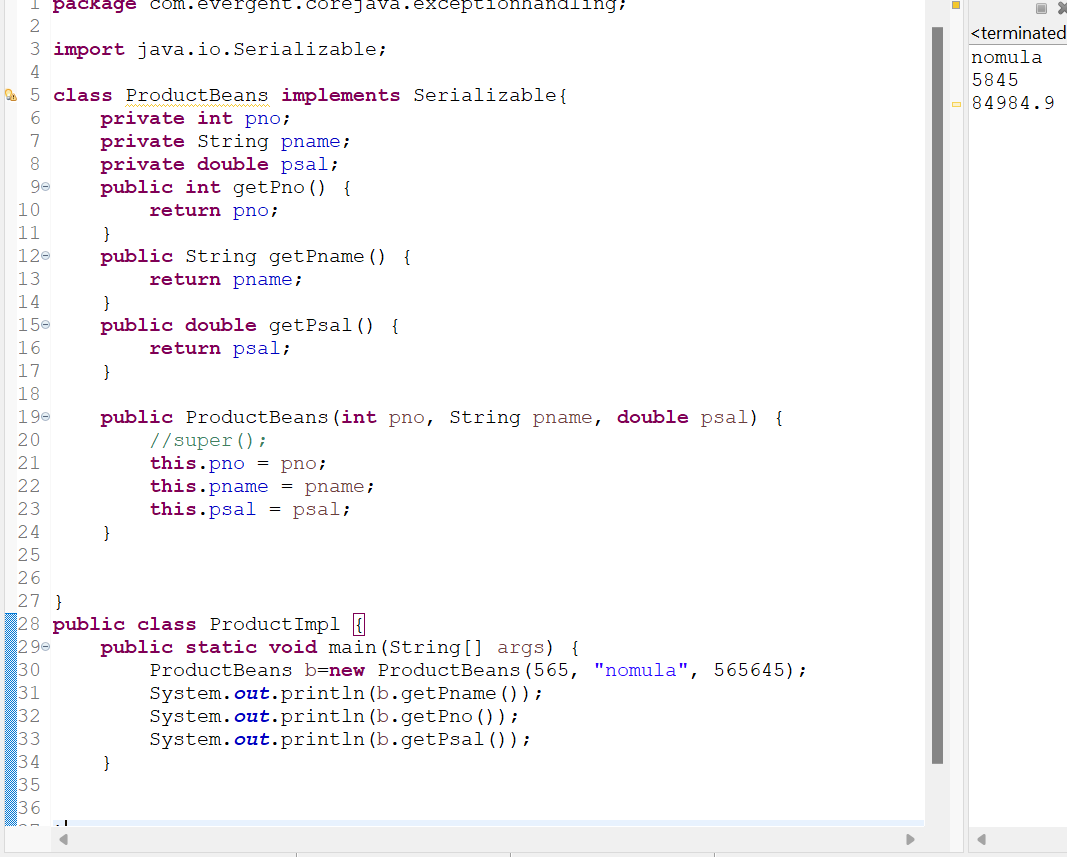
**JAVA BEANS**

* Java bean is a mechanism
* Java bean is a light weight
* All attributes are private and get/set methods are public implements java.io.Serializable interface
* We can achieve tightly encapsulation through java beans

PROGRAM-1



PROGRAM-2



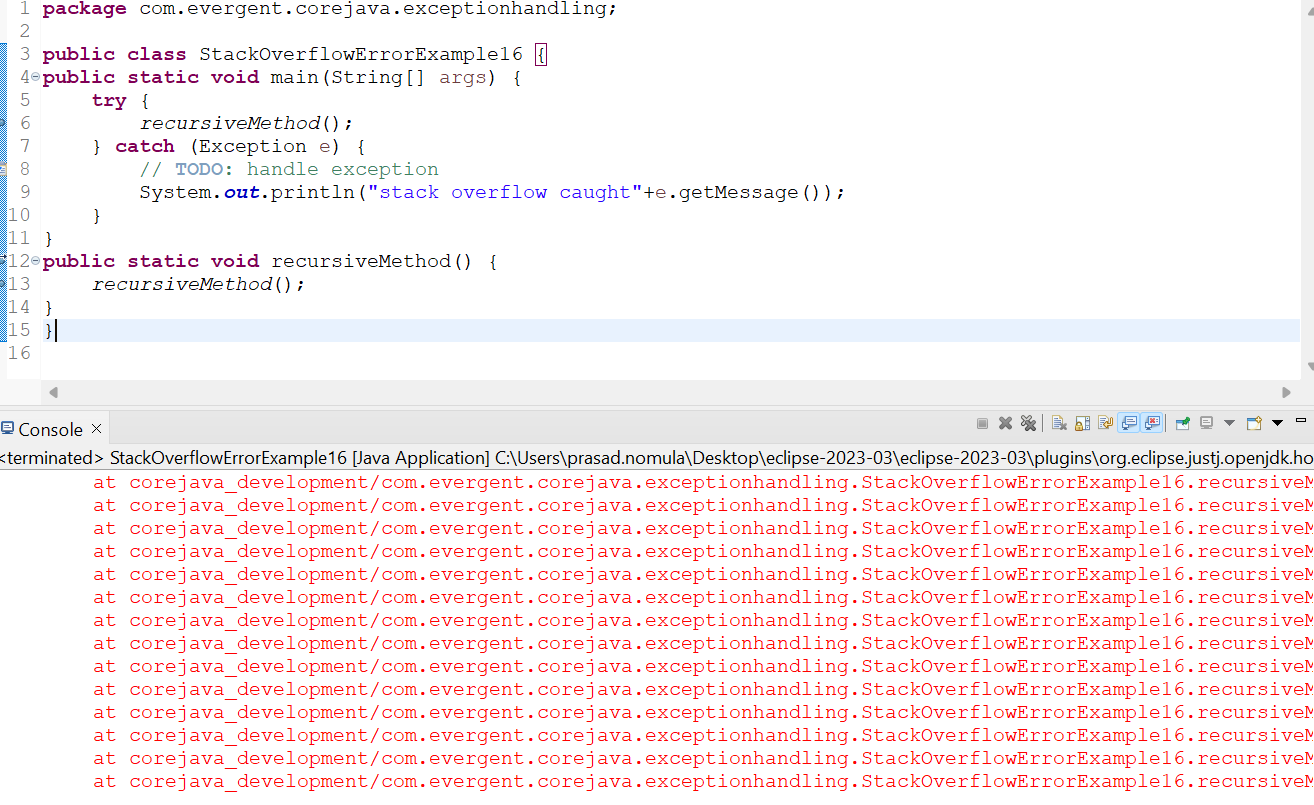
PROGRAM-3



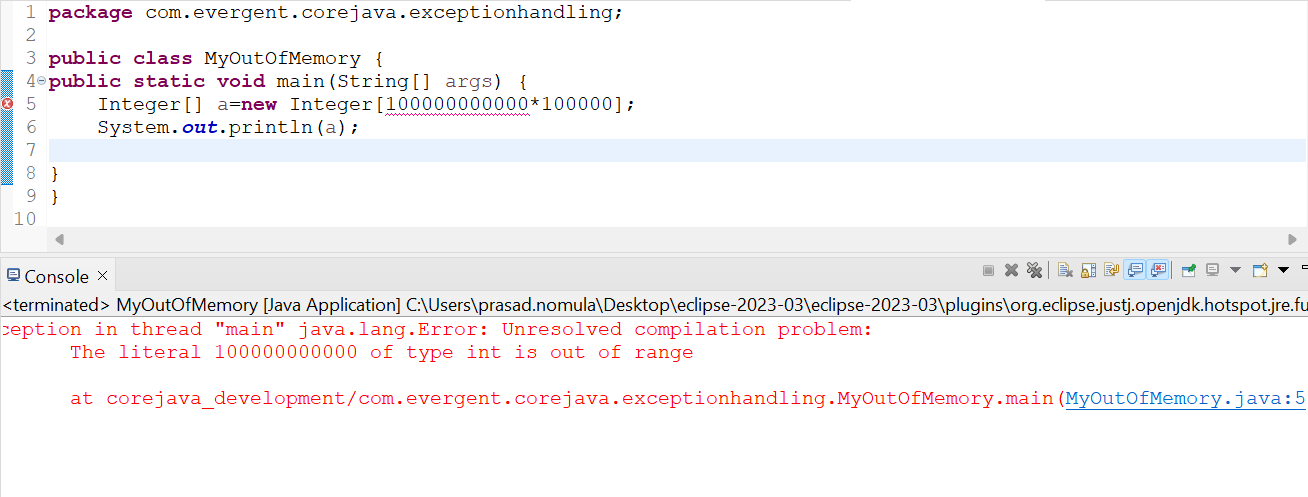
**22-08-2024 (DAY-13) THRUSDAY**

* Revised all concepts related to exception handling once again
* Discussed error concepts like stack overflow,out of memory
* Discussed about git ,github
  + Git init
  + Git status
  + Git add .
  + Git status
  + Git commit -m “abstract”
  + Git branch //origin or master
  + Git remote add origin https……………………….
  + Git push --force origin master

**PROGRAM-1**



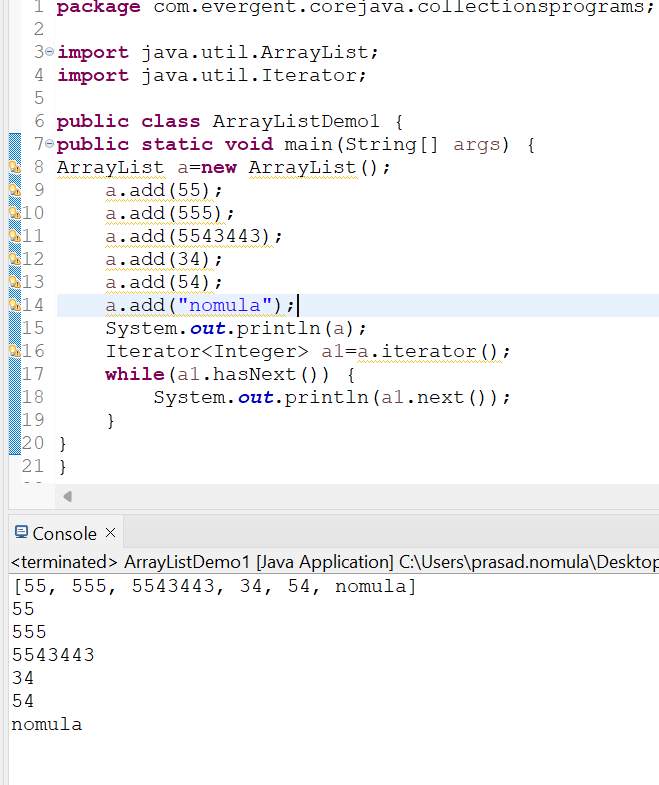
PROGRAM-2



**22/08/2024 AFTERNOON**

* Discussed about collections
* In collections we learned only 2 topics list and set
* In set it is 2 types they hashset and treeset
* In list they are 3 types
  + Arraylist
  + Linkedlist
  + Vector
* Vector is a legacy it is not recommonded to use
* Set items are stored randomly and duplicates are not allowed
* List items are stored one by one it will allow duplicates

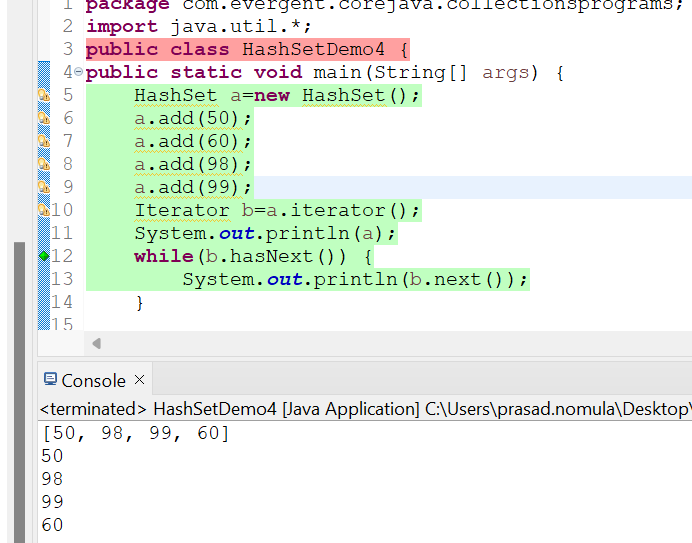
**Program1**

****

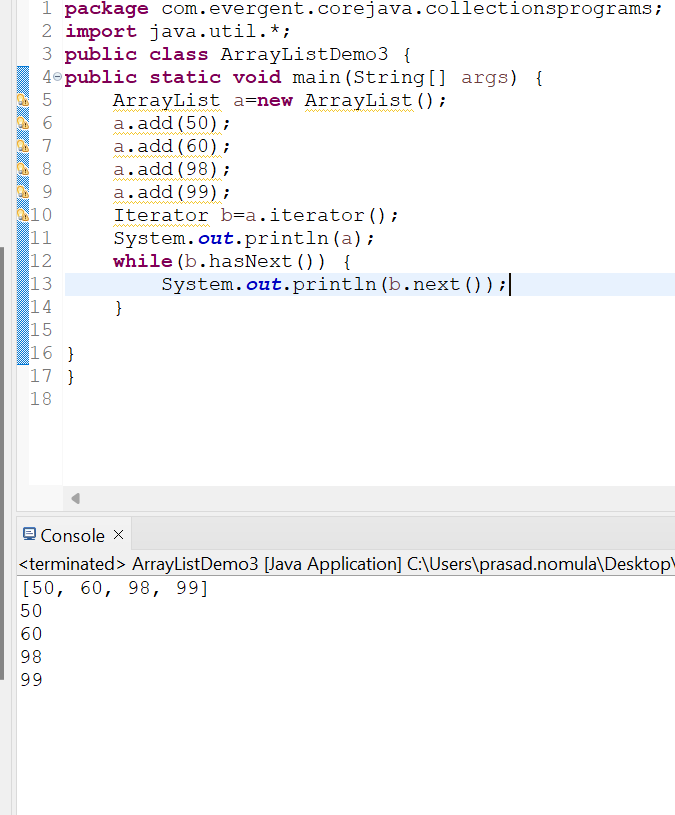
**PROGRAM-2**

****

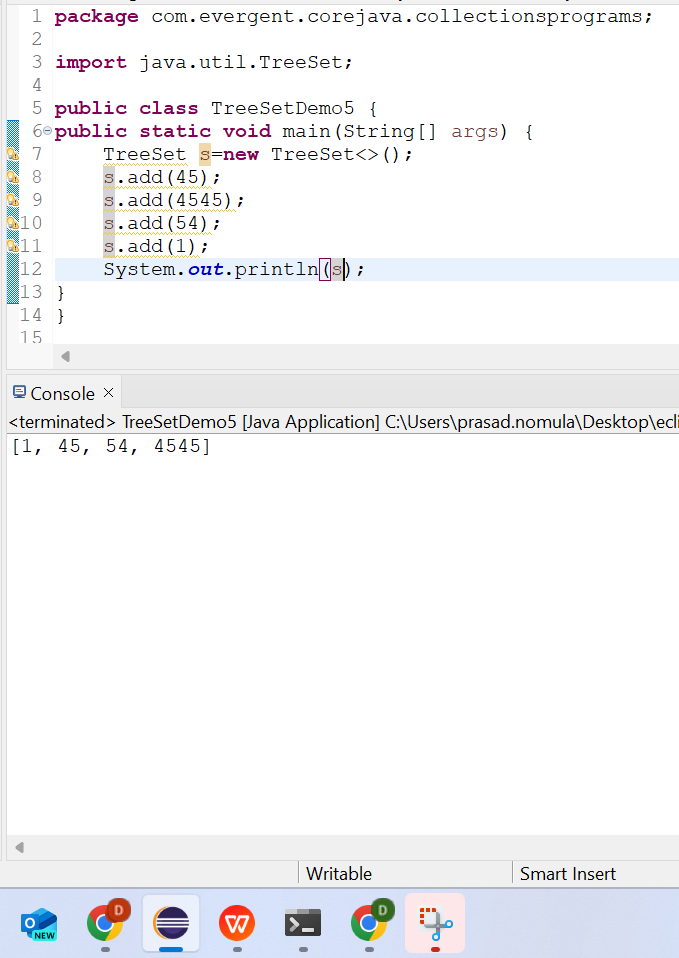
**PROGRAM-3**

****

**PROGRAM-4**

****

**PROGRAM-5**

****

**23/08/2024 DAY-14 FRIDAY**

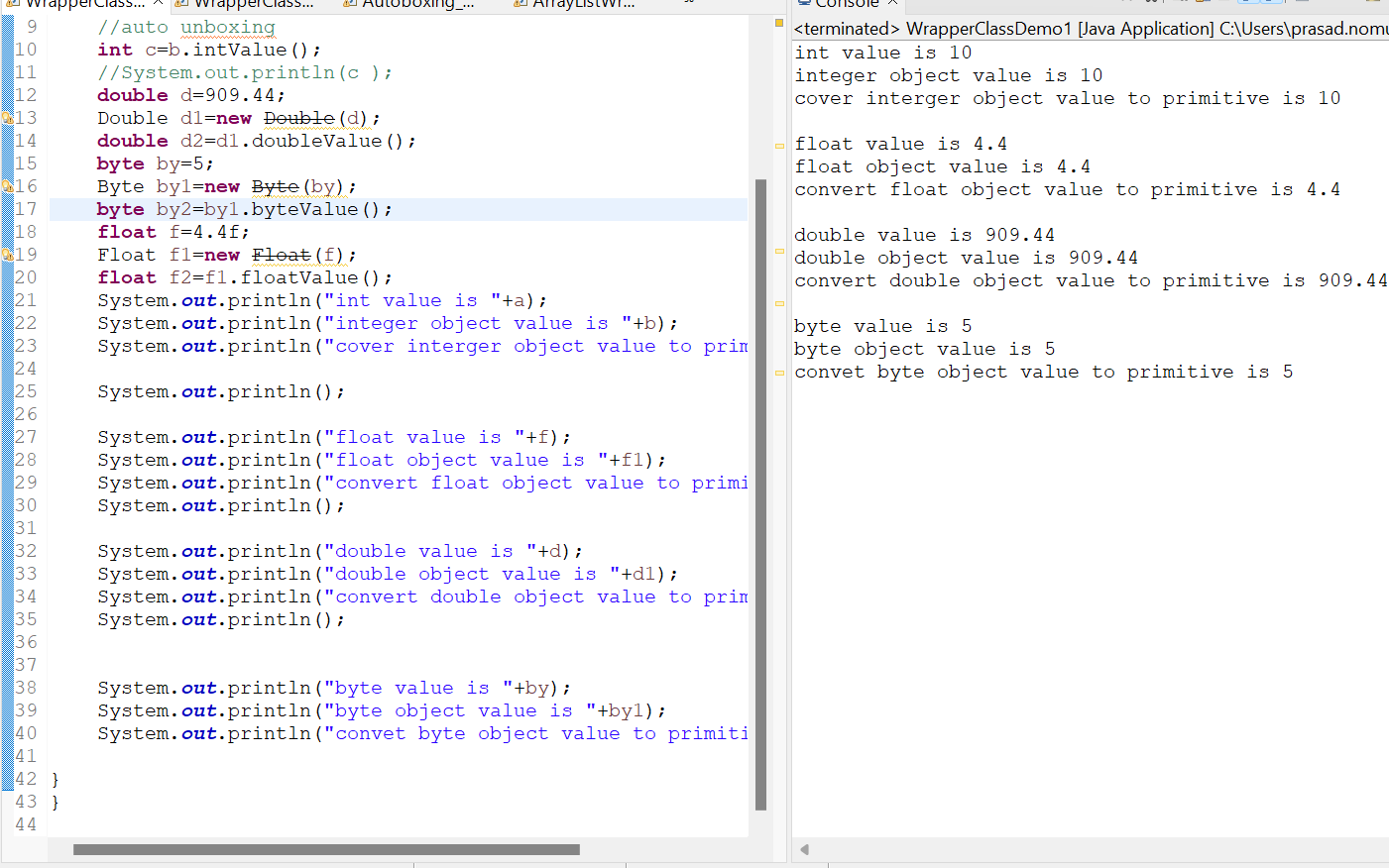
**MORNING**

**WRAPPER CLASS**

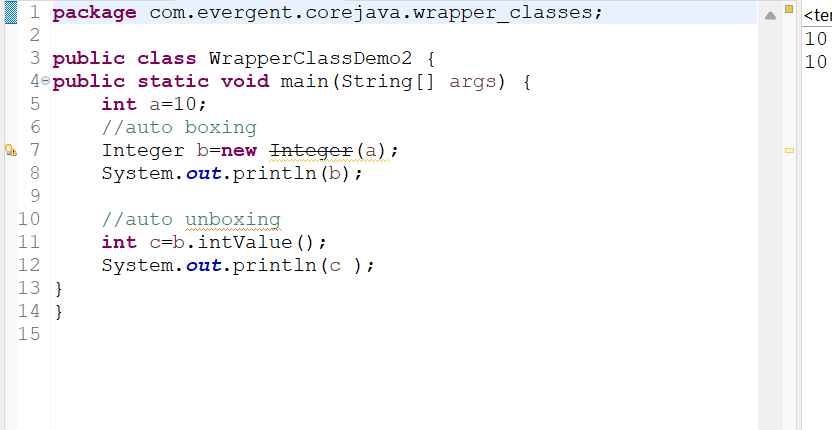
A Wrapper class in Java is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object. Let’s check on the wrapper classes.

In wrapper class auto boxing and auto unboxing will happens hiddenly this process we cant see

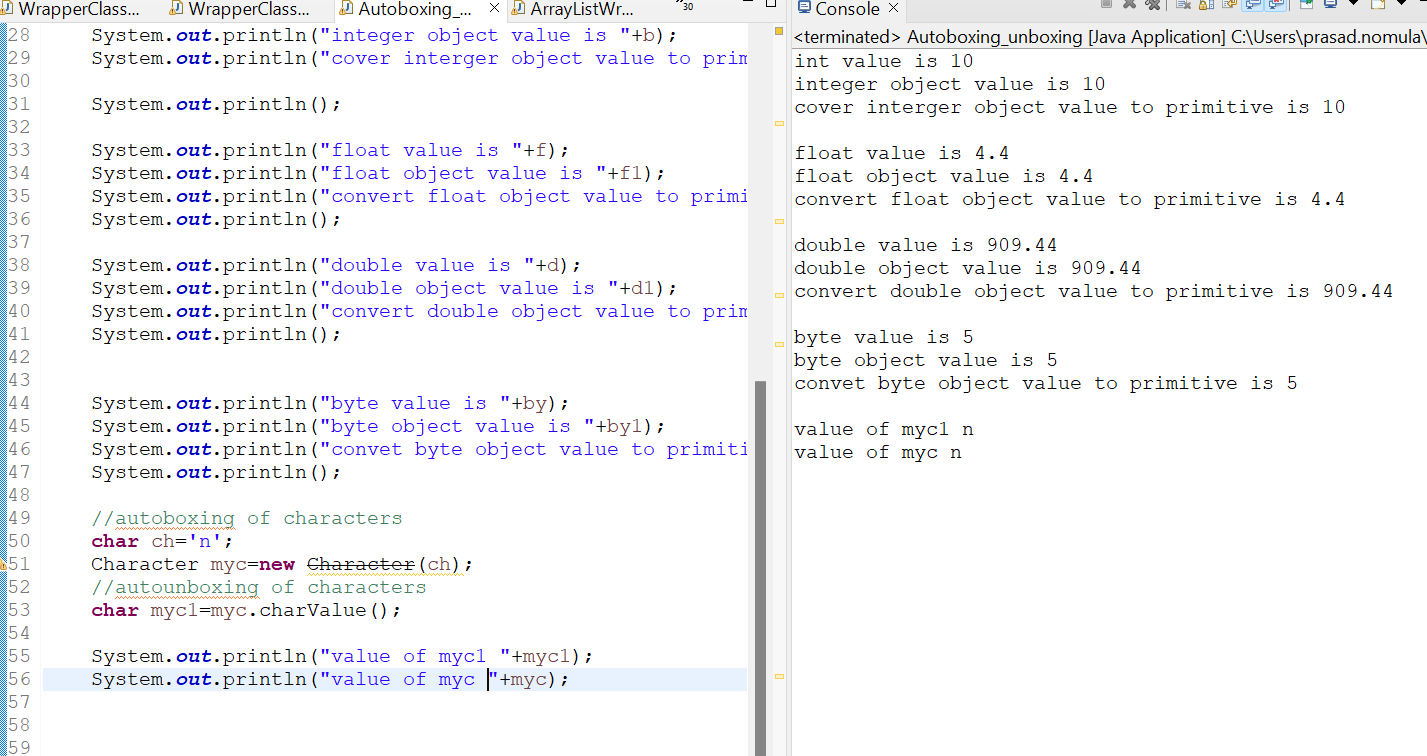
PROGRAM-1



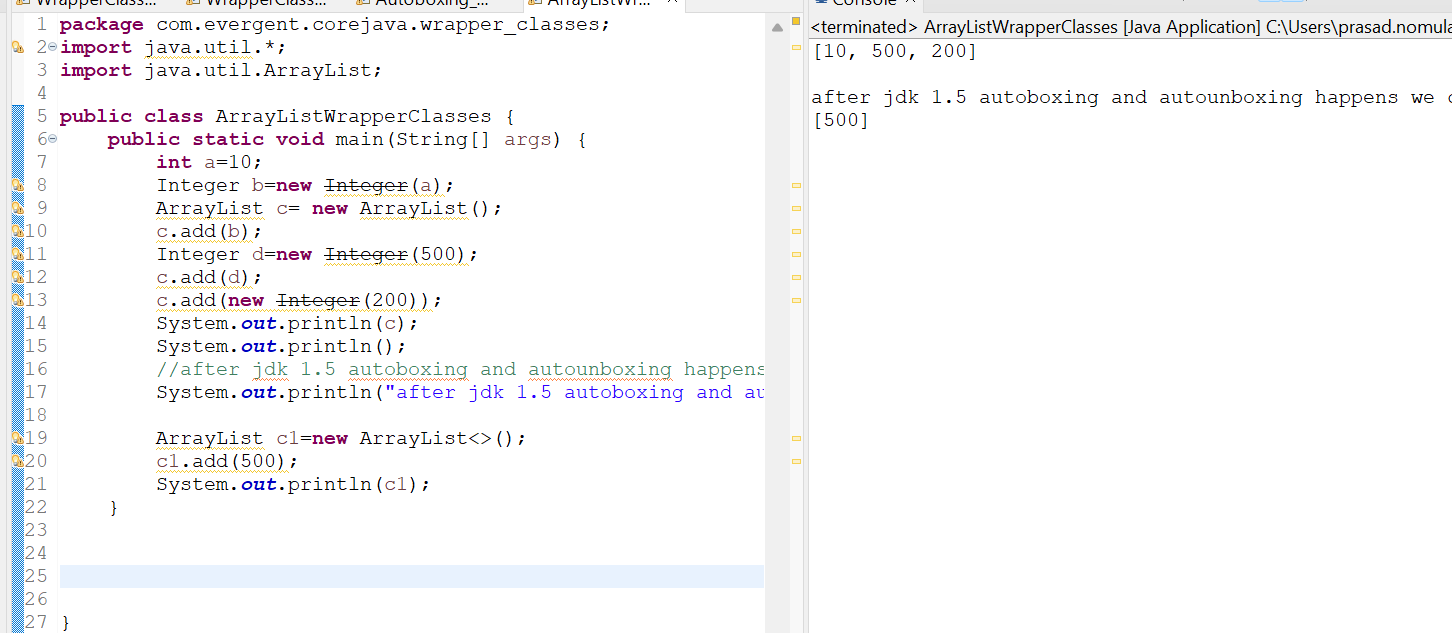
PROGRAM-2



PROGRAM-3



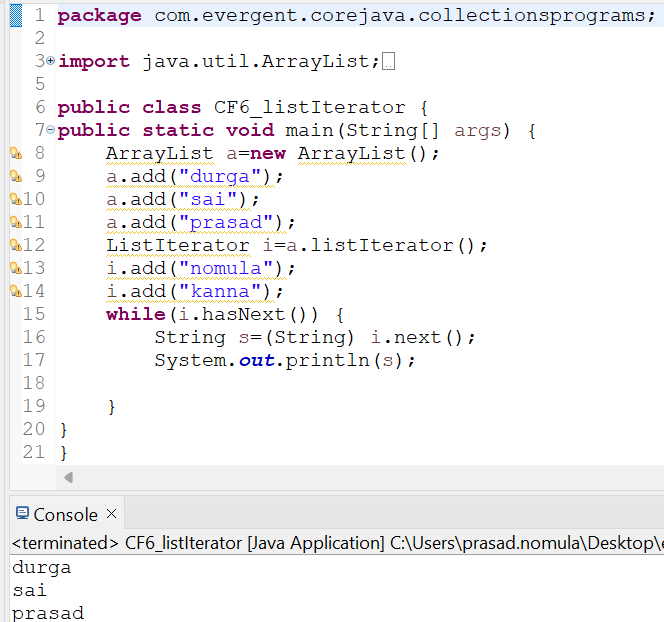
PROGRAM-4



**23/08/2024 AFTERNOON**

* For arraylist we use iterator and listiterator the main advantage of the listIterator is  
  we can add and print elements from boths sides
* In listiterator we can mainly 2 methods they are  
  1.hasNext()  
  2.hasPrevious()

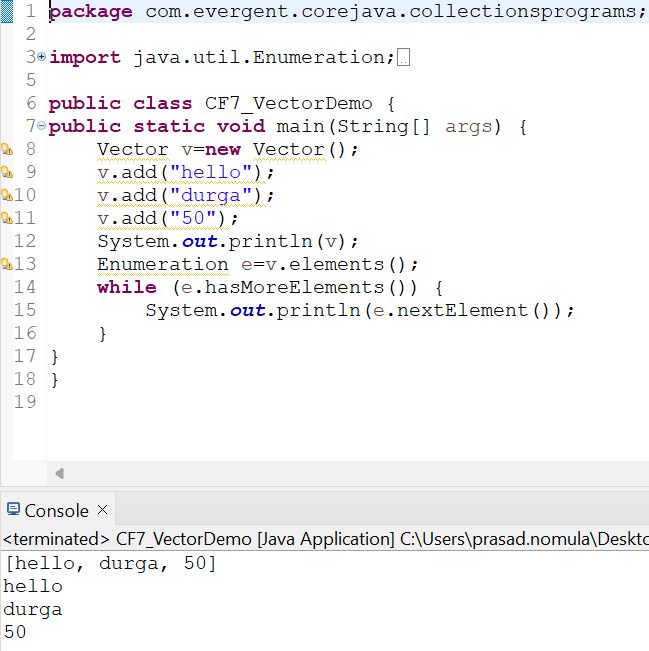
**PROGRAM-1**

****

**VECTOR**

* It constructs an empty vector with the default size as 10.
* It constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.
* ****Vector**** is like the dynamic array which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit. It is a part of Java Collection framework since Java 1.2. It is found in the java.util package and implements the List interface, so we can use all the methods of List interface here.
* t is recommended to use the Vector class in the **thread-safe** implementation only. If you don't need to use the thread-safe implementation, you should use the ArrayList, the ArrayList will perform better in such case.
* The Iterators returned by the Vector class are fail-fast. In case of concurrent modification, it fails and throws the ConcurrentModificationException.
* **It is similar to the ArrayList, but with two differences-**
* Vector is synchronized.
* Java Vector contains many legacy methods that are not the part of a collections framework.

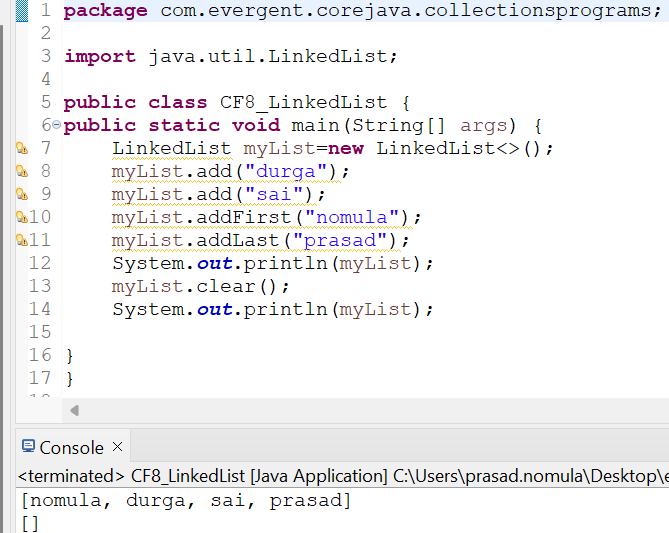
**PROGRAM**

****

**LINKED LIST**

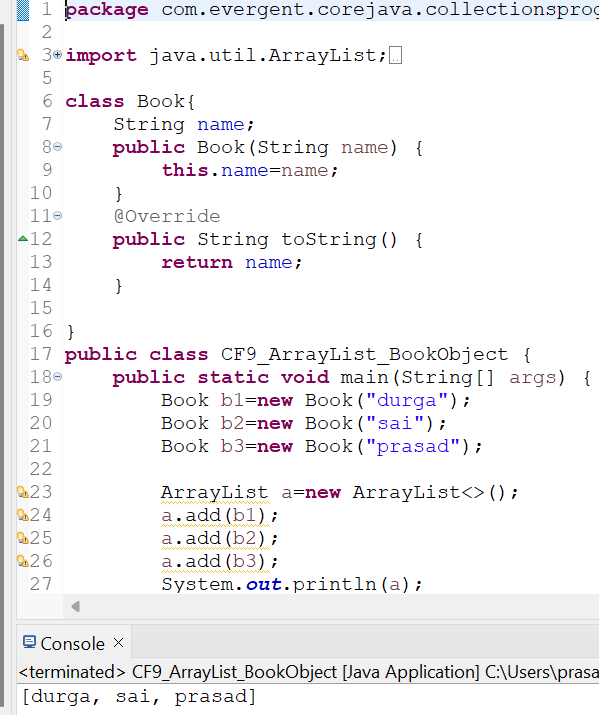
* In a linked contains a group of elements in a form of nods
* Each node will have 3 fields of data the fiels contains data and the link field contains references to previous and next node
* Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.
* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to occur.
* Java LinkedList class can be used as a list, stack or queue.

PROGRAM



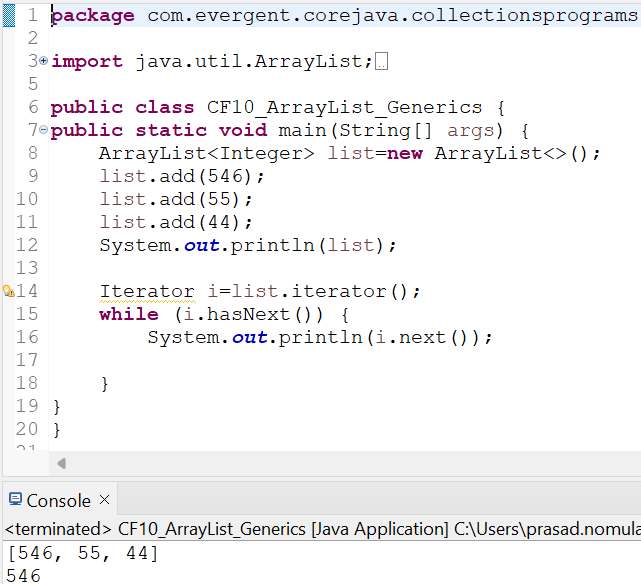
**ARRAYLIST WITH OBJECTS**

with the help of arraylist list we can access or we can store objects in the arraylist

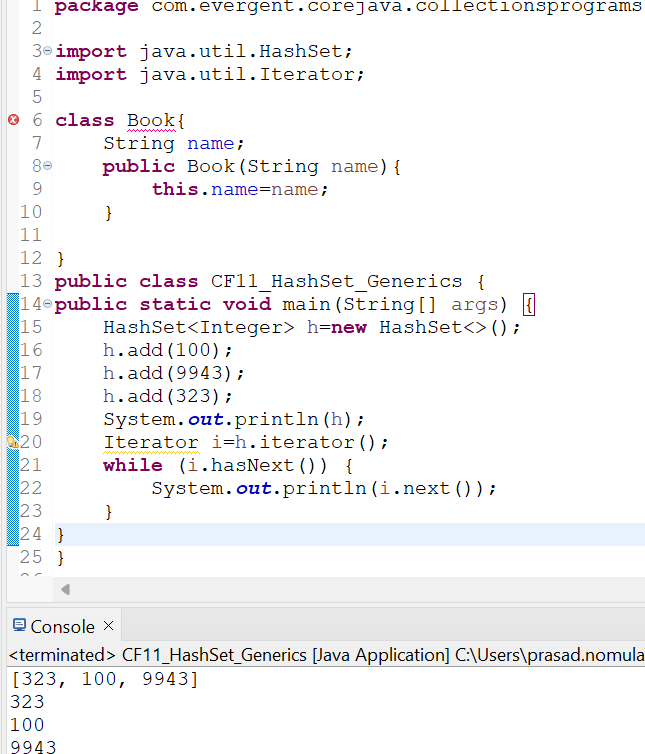
To acces objects in arraylist we want to use **toString() Method** in parent class

**GENERICS**

**Generics** means **parameterized types**. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.



**PROGRAM-2**

****

**26/08/2024**

**DAY-15(MONDAY)**

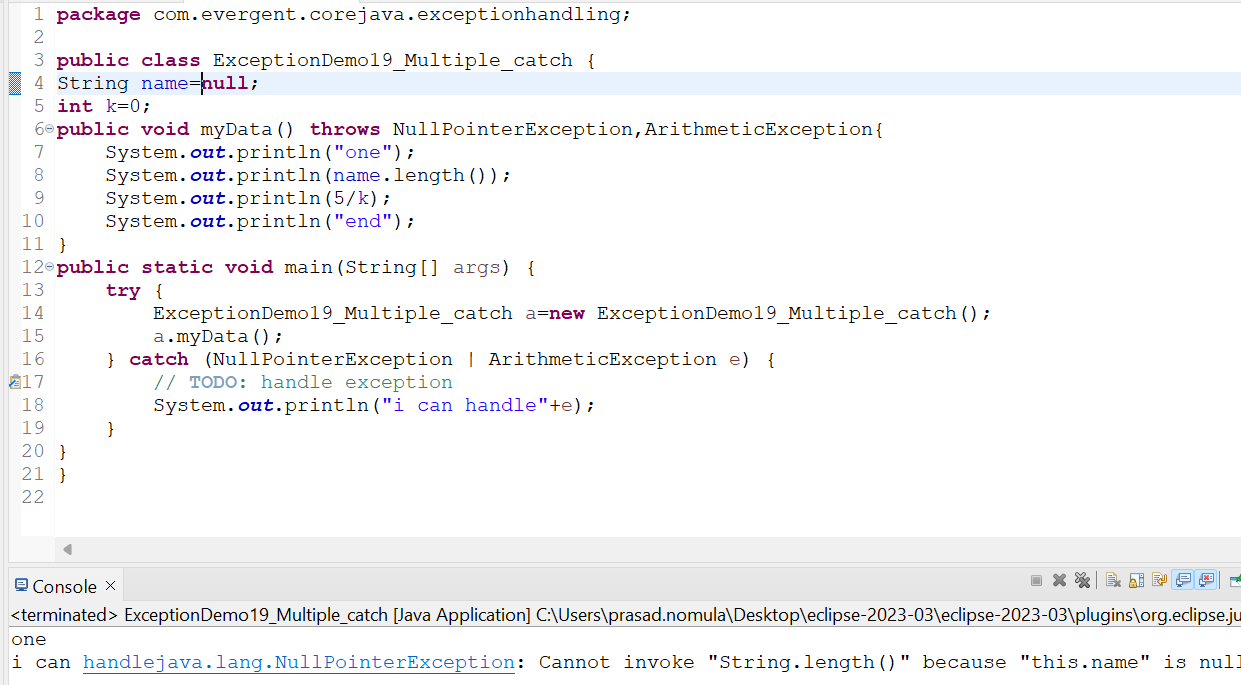
* We can throws multiple exceptions at a time we can separate with commas

PROGRAM-1



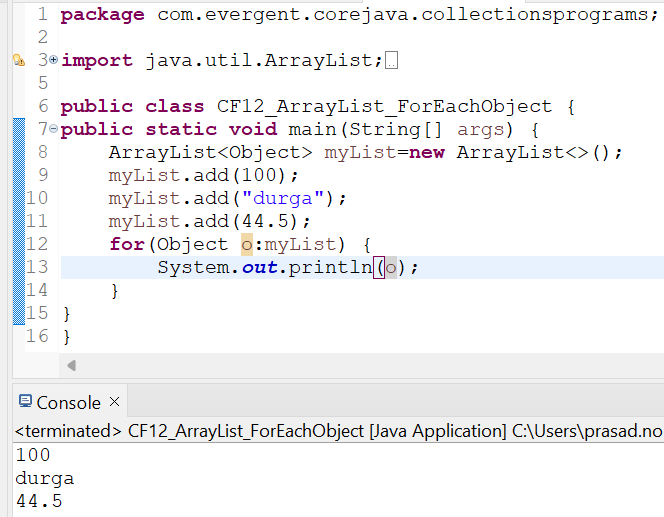
* We can handle multiple exceptions at a time in catch block we can seprate exception names with pipeline symbol( | )

PROGRAM



* Retriving values from collection objects using iterator,listiterator,enumeration
* Instead of printing data using while loop we use foreach loop

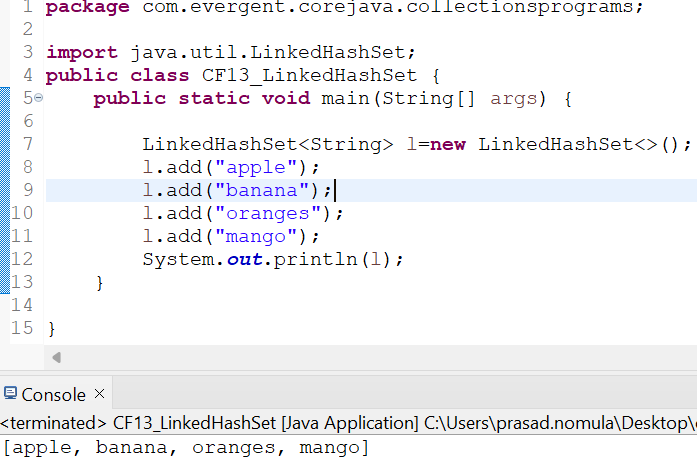
PR0GRAM



**LINKED HASH SET**

* Linkedlist is a subclass of hashset that maintains the insertion order of elements
* This means that elements are added and retrived in same order they were inserted
* Insertion order:elements are stored and retrived in the order they were adddressed

PROGRAM



**STACK**

· The Stack class in Java follows the Last-In-First-Out (LIFO) principle, where the last element added is the first to be removed.

· It is part of the java.util package and extends the Vector class.

· The primary operations include push() to add an element, pop() to remove the top element, peek() to view the top element without removing it, empty() to check if the stack is empty, and search() to find the position of an element in the stack.

· Being a subclass of Vector, Stack is synchronized, making it thread-safe for use in multi-threaded environments.

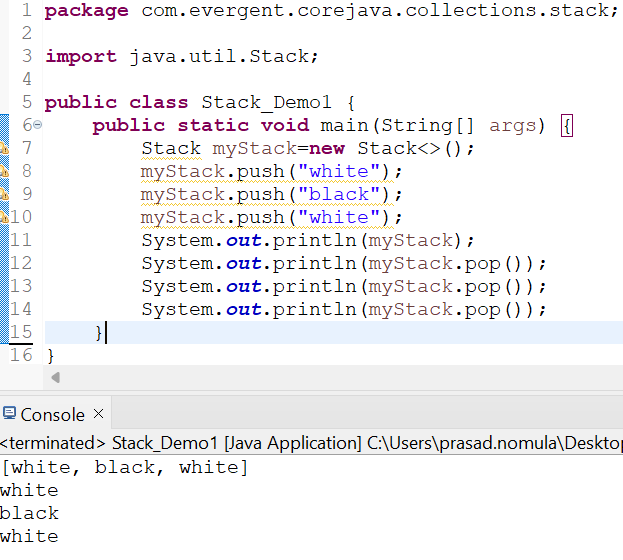
· Due to the overhead of synchronization, Stack may be slower than non-synchronized alternatives, such as ArrayDeque or LinkedList, which can be used for stack-like behavior.

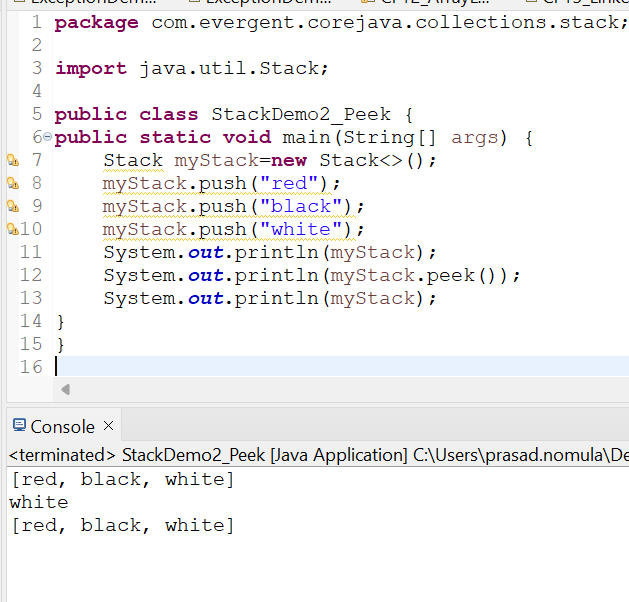
· Stack is considered a legacy class in Java, with newer alternatives like Deque often preferred for modern applications.

· The size of a Stack can grow dynamically as elements are added, thanks to its inheritance from Vector.

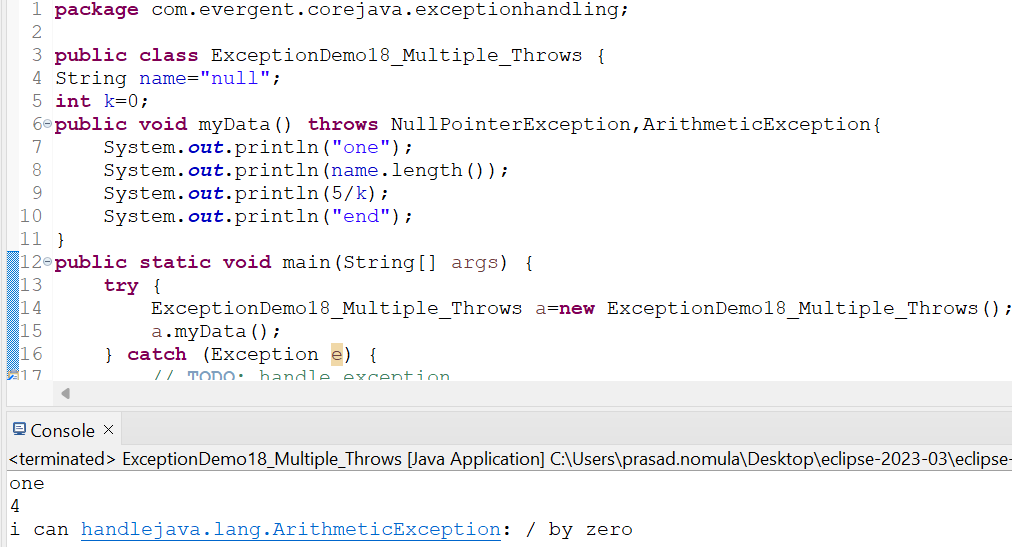
· The Stack class implements the Serializable interface, allowing it to be serialized and deserialized.

**PROGRAMS**

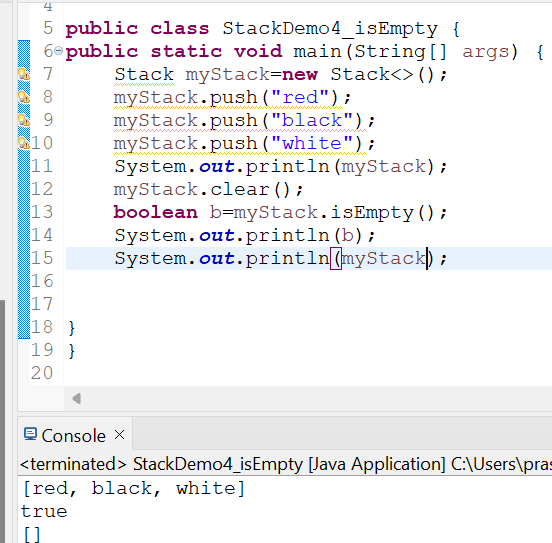
****

****

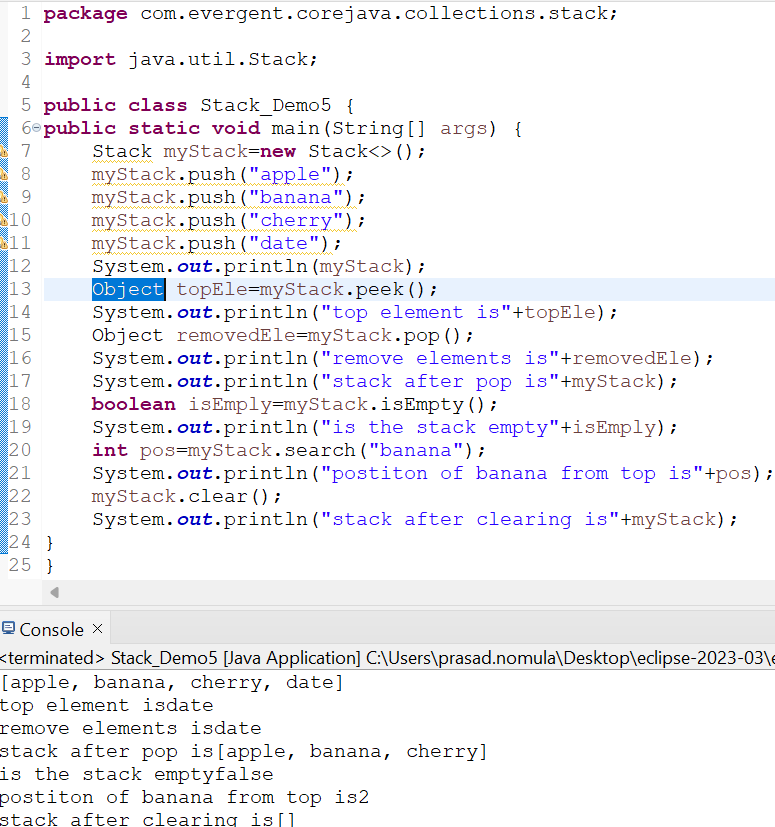
**PROGRAM -3**

****

PROGRAM-4



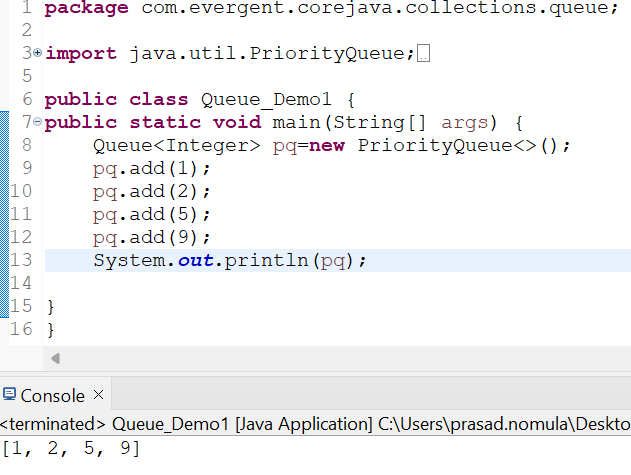
PROGRAM -5

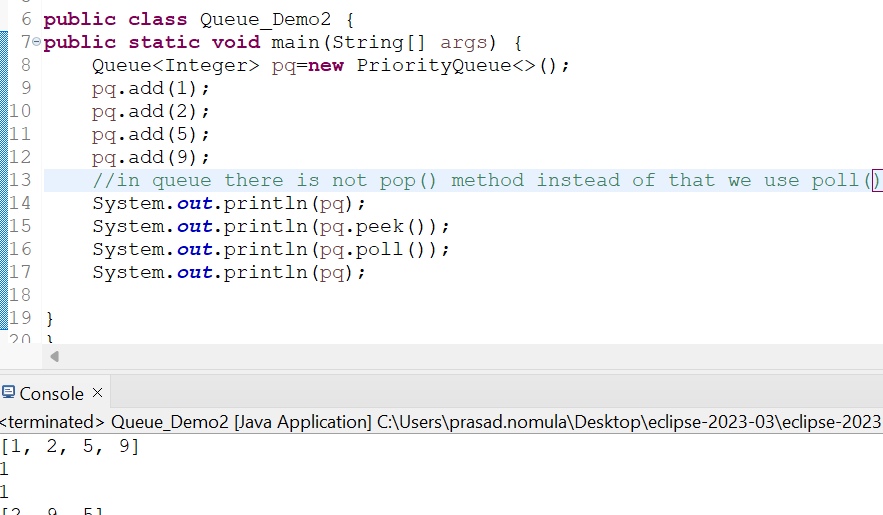


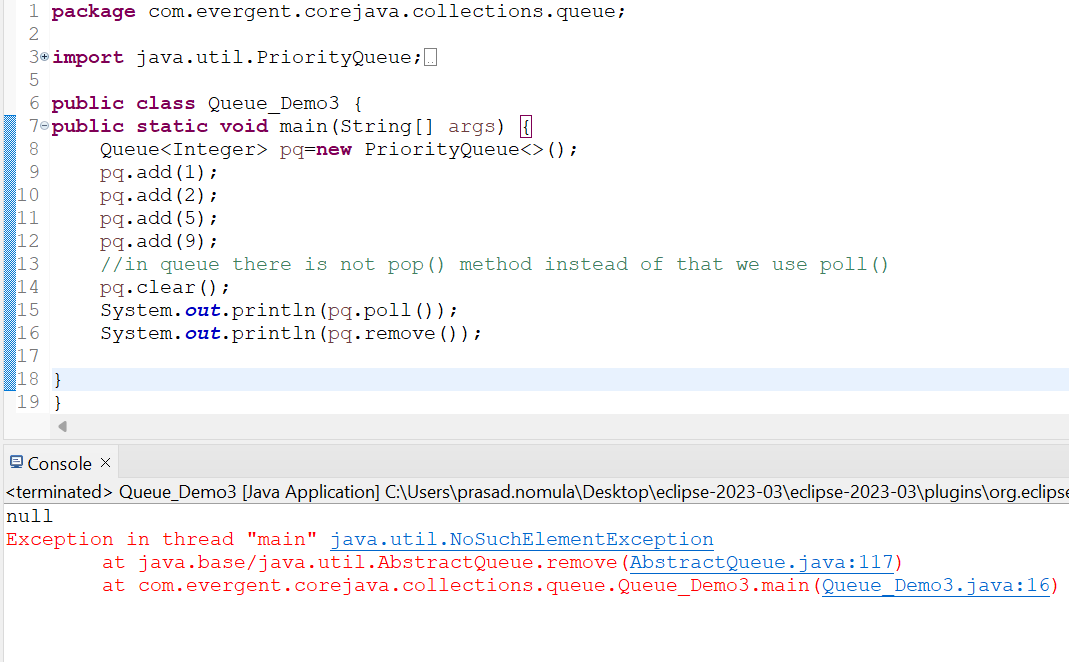
**QUEUE**

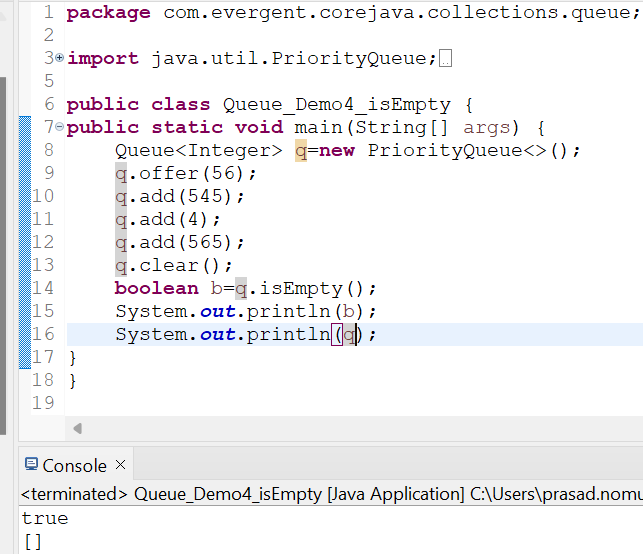
* The Queue interface in Java is part of the java.util package and represents a collection designed for holding elements prior to processing, typically in a First-In-First-Out (FIFO) order.
* Queue does not allow direct instantiation; instead, it is implemented by classes such as LinkedList, PriorityQueue, and ArrayDeque.
* The primary operations include offer() to add an element to the queue, poll() to remove and return the head of the queue, peek() to retrieve the head of the queue without removing it, and remove() to remove a specific element.
* Unlike add(), the offer() method returns false if the queue is full or unable to accommodate the element, making it preferable in capacity-restricted queues.
* poll() and remove() methods differ in behavior when the queue is empty: poll() returns null, while remove() throws an exception.
* Queue implementations can be categorized into different types, such as PriorityQueue for priority-based ordering and LinkedList or ArrayDeque for FIFO ordering.
* Deque (Double-Ended Queue), which extends Queue, allows elements to be added or removed from both ends, providing more flexibility.
* The BlockingQueue interface, which extends Queue, is used in concurrent programming, offering thread-safe operations and blocking methods like take() and put().
* Queue is widely used in scenarios like task scheduling, buffering, and managing resources in a controlled manner.

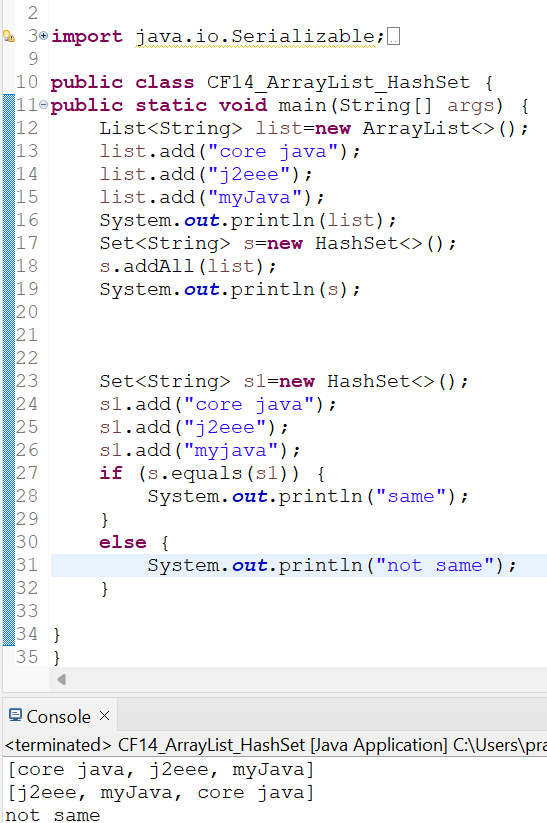
PROGRAMS-1





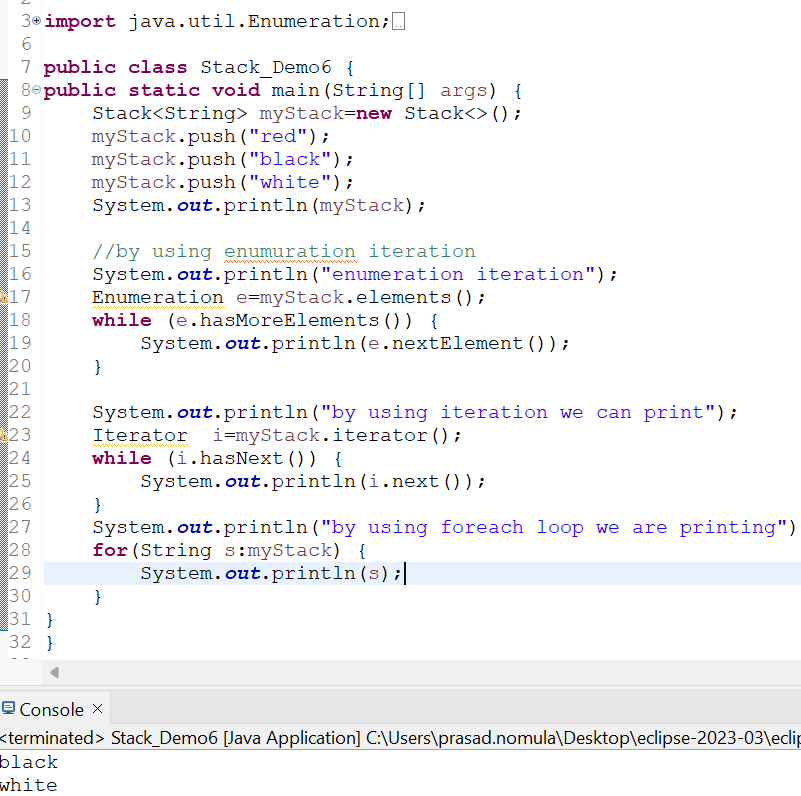




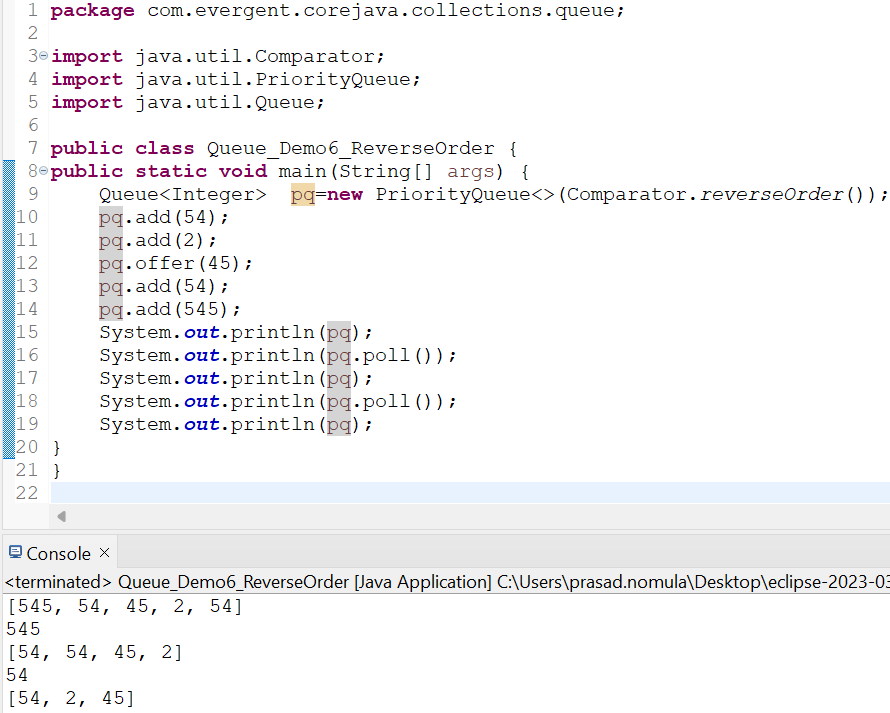


**27/08/2024**

**DAY-16(TUESDAY)**

****

**HIGHEST VALUE --> HIGHEST PRIORITY**

****

**ARRAY DEQUE**

· **ArrayDeque** (short for "Array Double Ended Queue") is a resizable array implementation of the Deque interface in Java, providing a more flexible and efficient alternative to traditional stacks and queues.

· It is part of the java.util package and can be used as both a queue (FIFO - First-In-First-Out) and a stack (LIFO - Last-In-First-Out).

· Unlike LinkedList, which also implements Deque, ArrayDeque does not have the overhead of node-based pointers, making it faster for adding and removing elements from both ends.

· The capacity of an ArrayDeque can grow dynamically as elements are added, but unlike ArrayList, it does not allow null elements.

· **Primary Operations** include addFirst() and addLast() to insert elements at the beginning or end, removeFirst() and removeLast() to remove elements from the beginning or end, and peekFirst() and peekLast() to view elements at the ends without removing them.

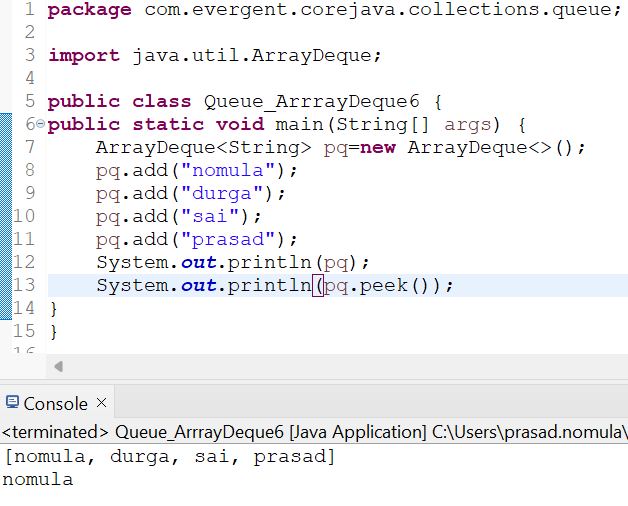
· For stack-like operations, push() is equivalent to addFirst(), and pop() is equivalent to removeFirst().

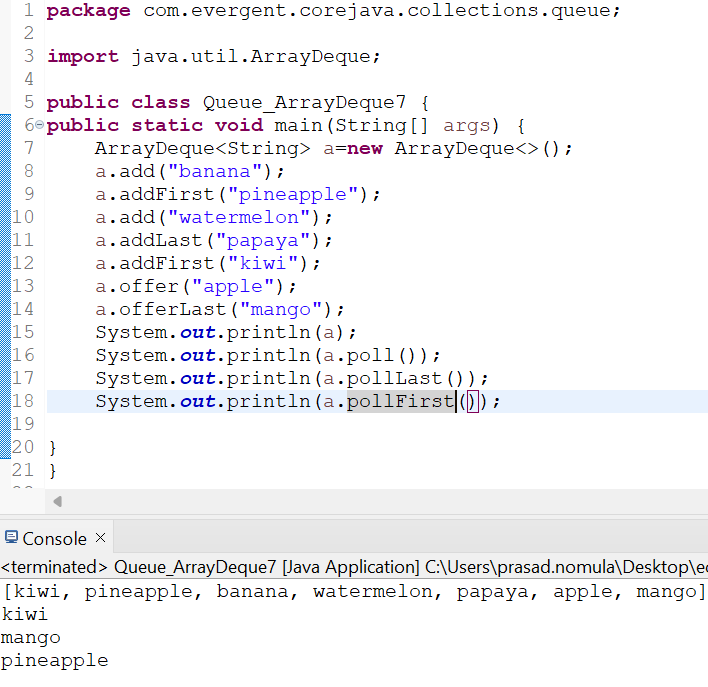
· **Performance** is generally superior to that of LinkedList for use cases involving frequent additions and removals from both ends, as ArrayDeque is implemented with a circular array, minimizing the cost of these operations.

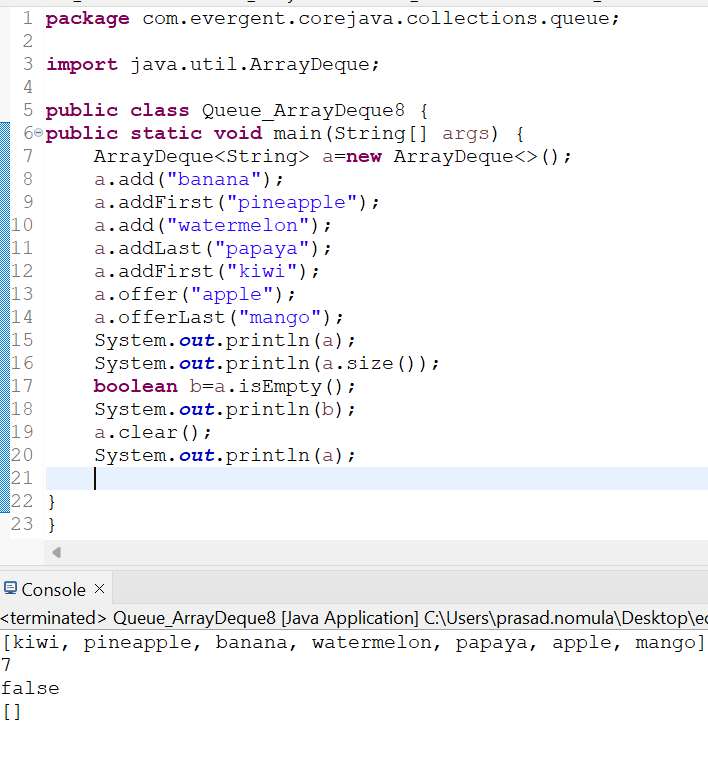
· **Thread Safety** is not provided by ArrayDeque; if used in a multi-threaded environment, external synchronization is necessary.

· ArrayDeque is a good choice when a resizable array-based double-ended queue is needed, offering constant time performance for most operations, including insertions, deletions, and element retrievals from both ends.

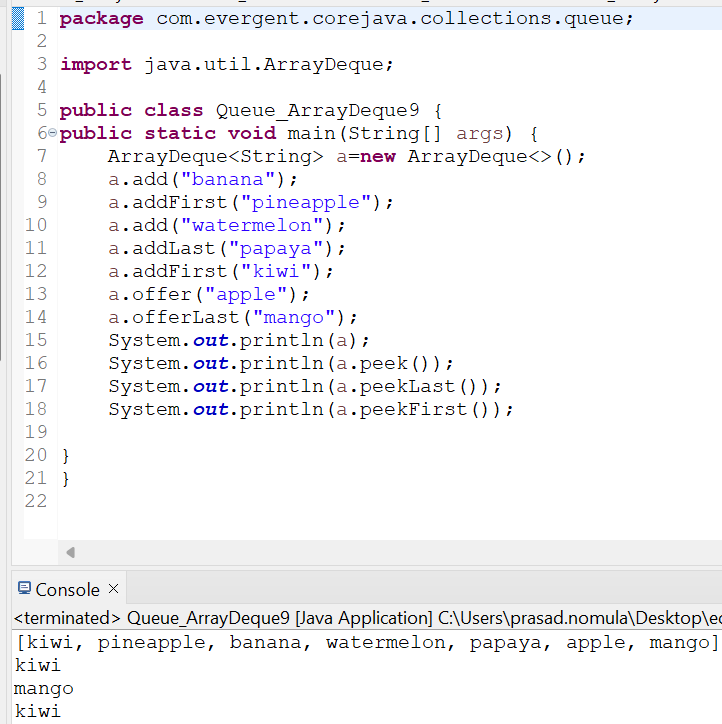
· It is an ideal choice for implementing non-blocking, thread-local stacks and queues in applications that require high performance.

****

****

****

**PROGRAM**

****

**LINKED LIST IN QUEUE**

· **LinkedList** in Java is a class that implements both the List and Deque interfaces, making it a versatile data structure that can function as both a list and a queue.

· When used as a queue, LinkedList follows the First-In-First-Out (FIFO) principle, where elements are added to the end of the queue and removed from the front.

· The LinkedList is based on a doubly linked list structure, meaning each element (node) contains references to both the previous and next elements in the sequence.

· **Primary Queue Operations** include offer() to add an element to the end of the queue, poll() to remove and return the element at the front of the queue, peek() to retrieve the element at the front without removing it, and remove() to remove a specific element.

· Unlike array-based implementations, LinkedList does not have a fixed size; it can grow or shrink dynamically with the addition or removal of elements.

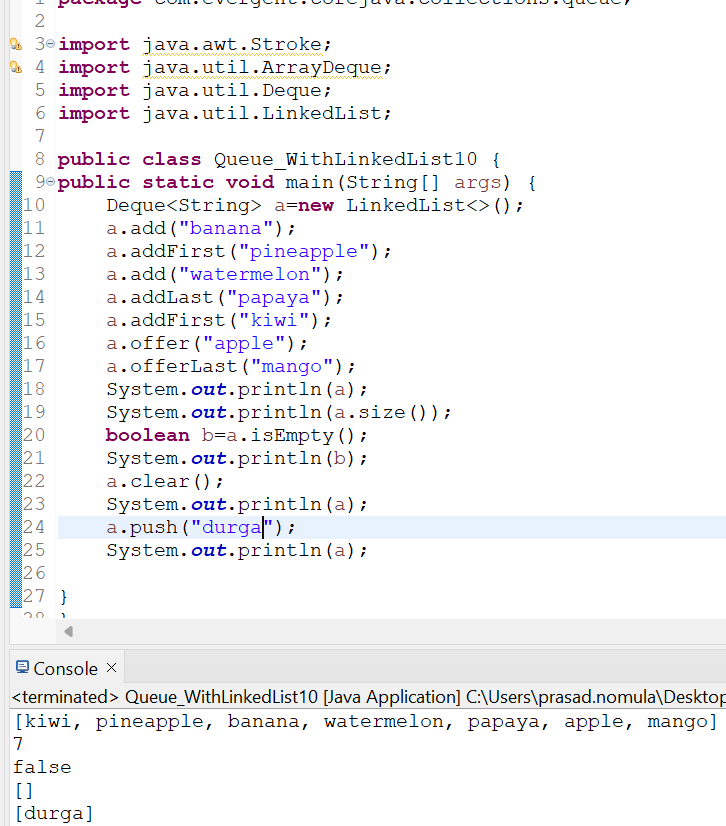
· **Performance Characteristics** of LinkedList are different from array-based structures. It provides O(1) time complexity for adding or removing elements from the front or end, but accessing elements by index is slower (O(n)) compared to array-based lists due to the need to traverse the nodes.

· **Thread Safety** is not provided by LinkedList; if it is used in a multi-threaded environment, external synchronization must be applied.

· **Use Cases** for using LinkedList as a queue include scenarios where frequent insertions and deletions from the front or end of the queue are needed, such as implementing queues, deques, and other sequential data structures.

· LinkedList also allows null elements, unlike some other queue implementations such as ArrayDeque.

· **Advantages** of using LinkedList in a queue context include its dynamic size and efficient insertion and deletion operations, especially when dealing with large datasets or when the size of the queue is unpredictable.

****

**28/08/2024**

**DAY-17(WEDNESDAY)**