**Core JAVA training - index**

**Date:05/08/2024**

1. Language And Applications
2. JAVA Features

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

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

**Date:06/08/2024**

1. Nested Loops
2. One Dimensional Array
3. Two Dimensional Array
4. Logical Programming
5. SwitchCase
6. Scanner Class
7. Java.lang
8. Object Class Methods
9. Enum
10. Event Management Application

**Date:07/08/2024**

1. oops
2. Encapsulation
   1. Programs
   2. Calculation
   3. Person
   4. MethodFlow
3. Inheritance
4. Polymorphism
   * 1. Method overloading
     2. Method Overriding
5. Abstraction
6. IS-A (Inheritance)
7. HAS-A (Object Creation).

**Date:08/08/24**

1. **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

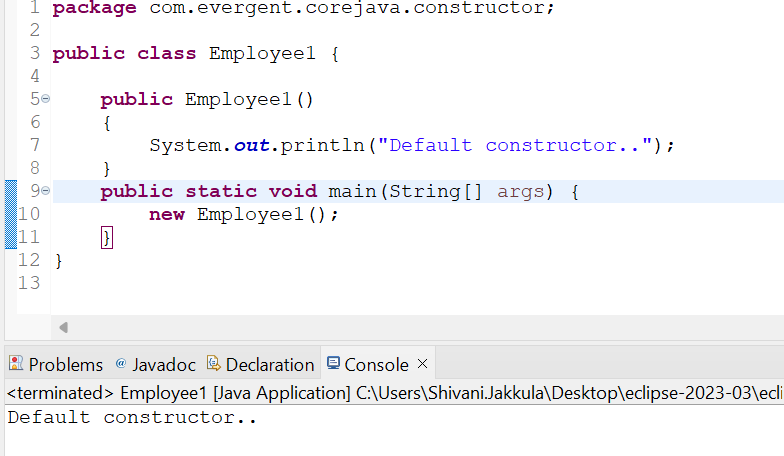
vi. Every class needs atleast 1 deafult constructor

vii. this, super This

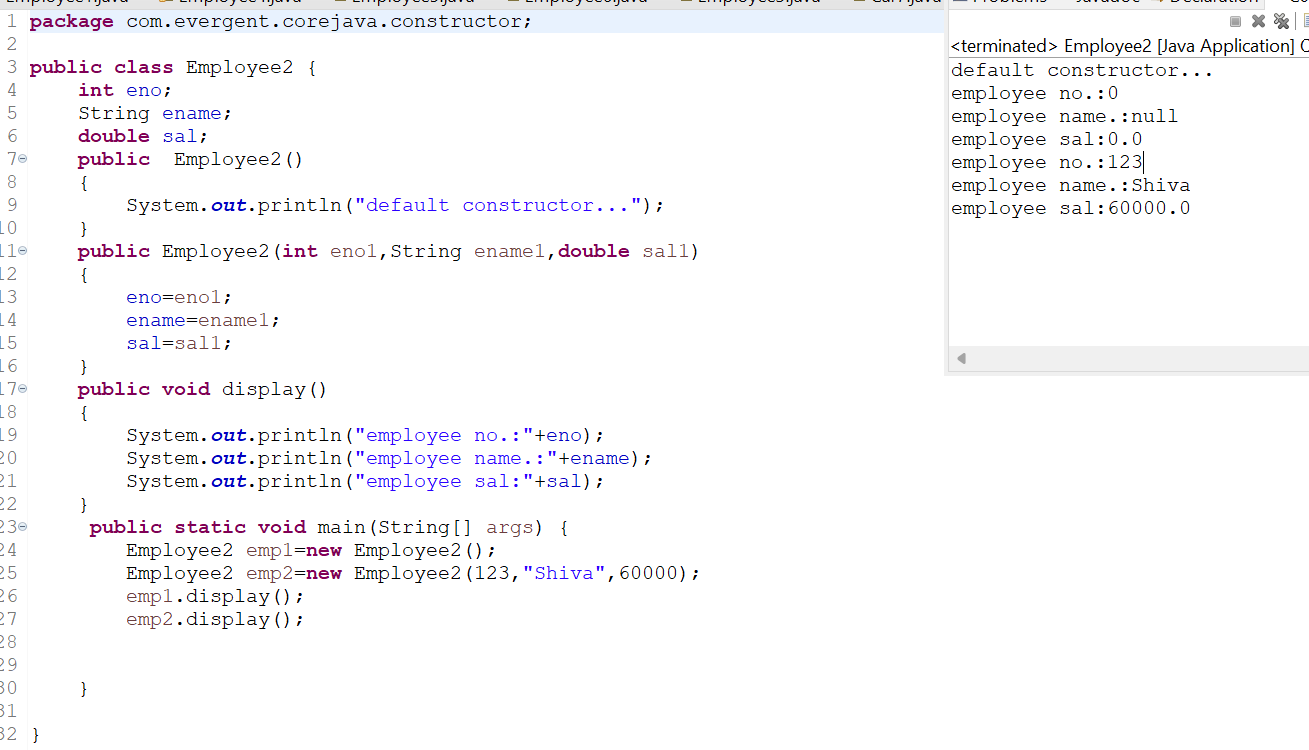
--> this is a keyword always refers to instance variables

viii. Always constructor are overloaded

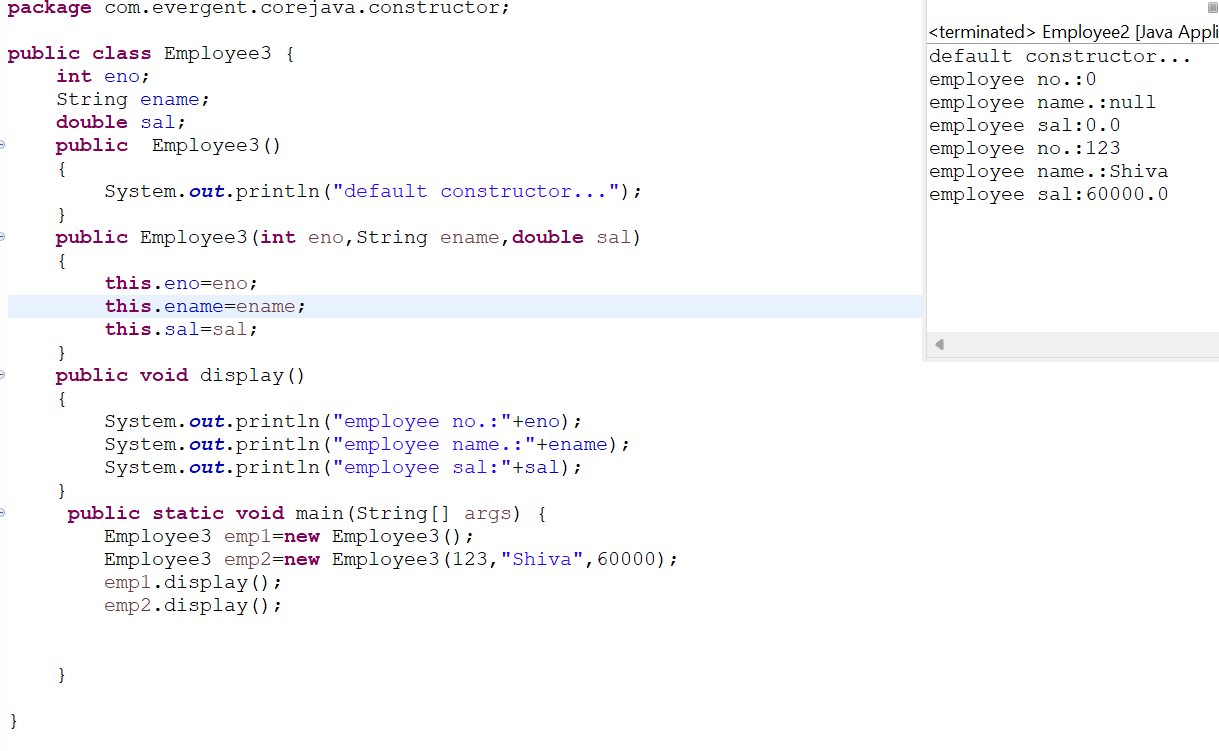
**Program1:**

****

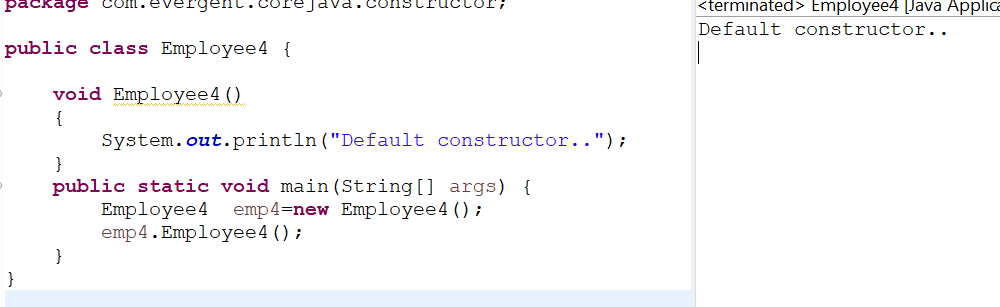
**Program2:**

****

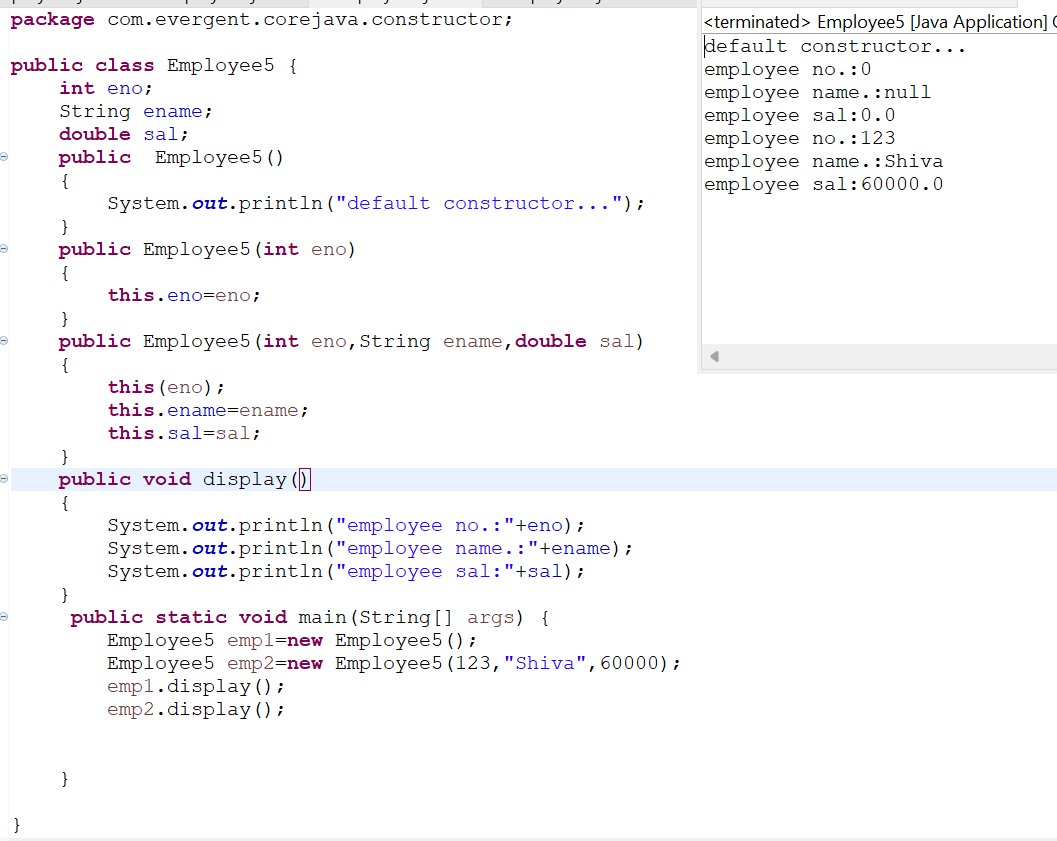
**Program3:**

****

**Program4:**

****

**Program5:**

****

**Program 6:**

****

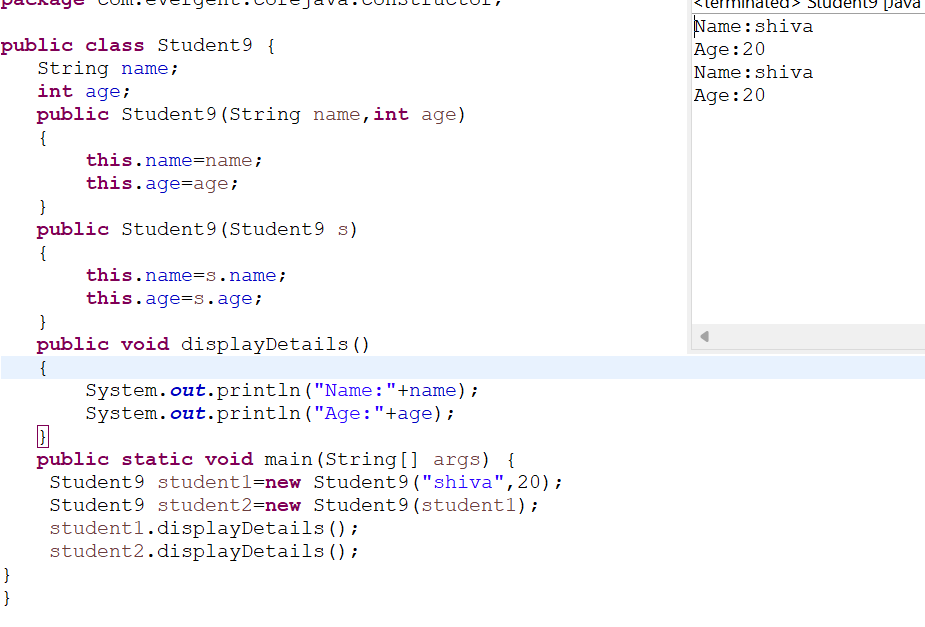
**Program 7:**

****

**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.

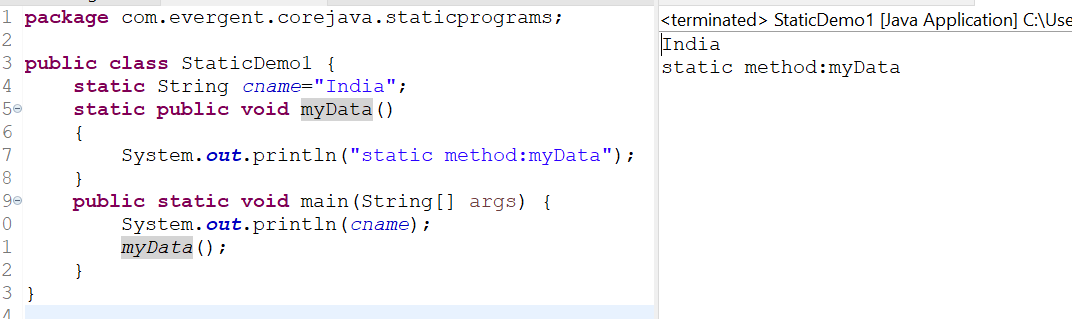
d. Static methods can access static methods and static variables only.

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

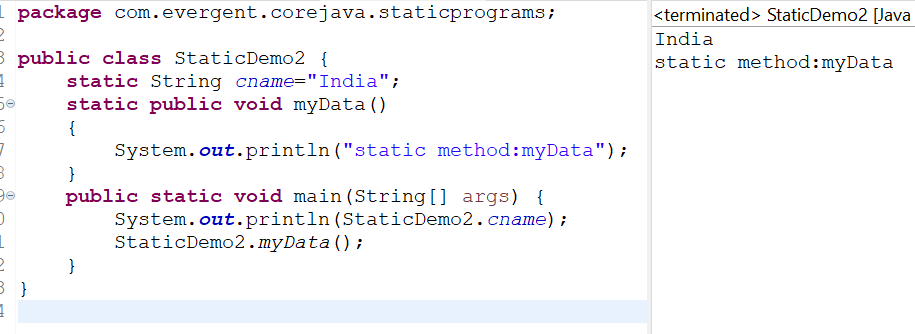
f. Non static methods can access static methods and static variables.

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

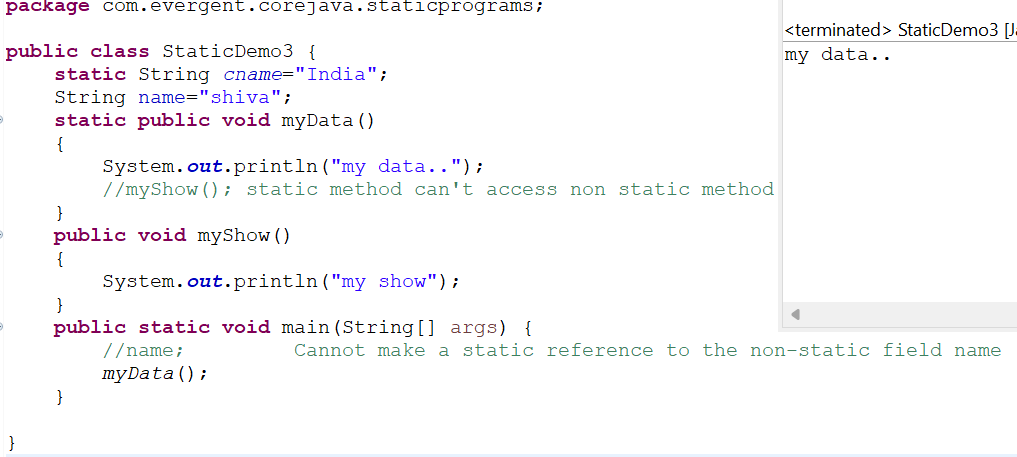
**Program1:**

****

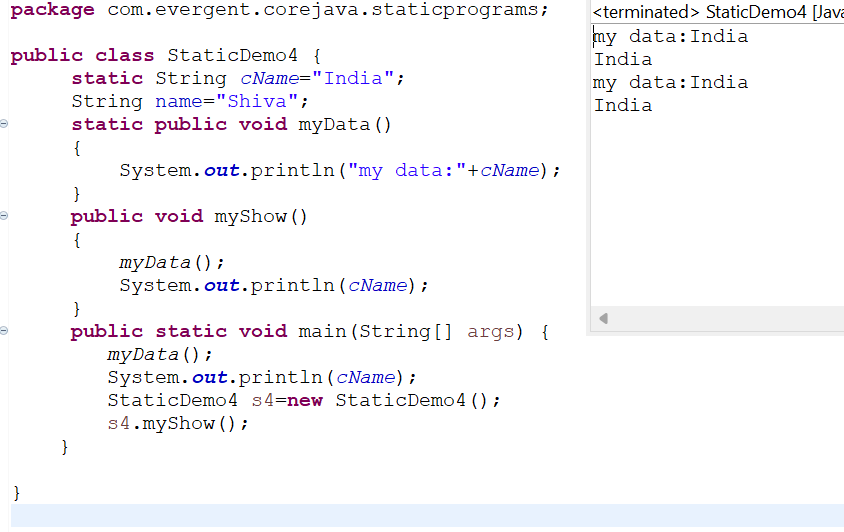
**Program2:**

****

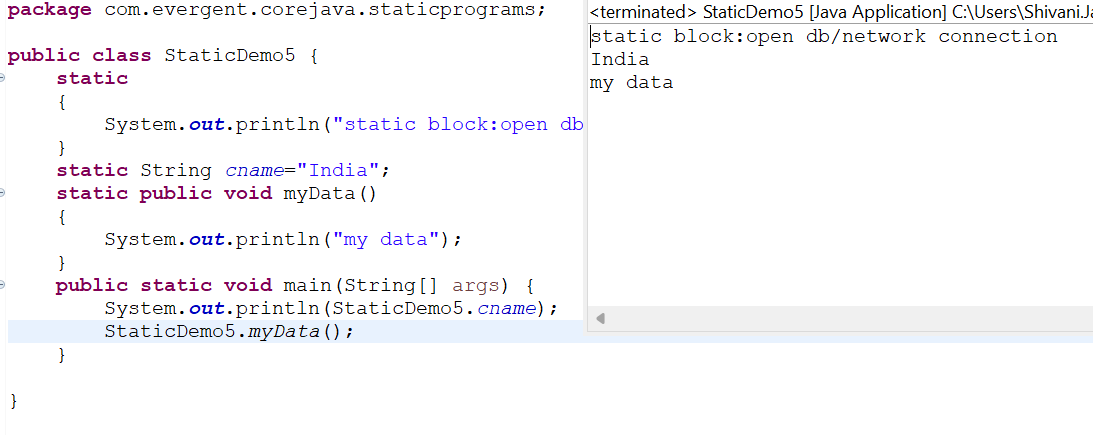
**Program3:**

****

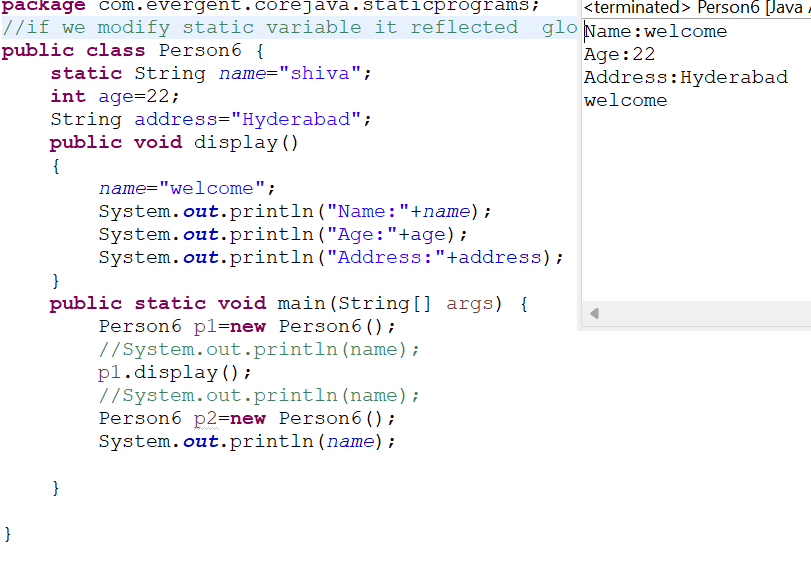
**Program4:**

****

**Program5:**

****

**Program6:**

****

1. **Final**

a. Final is a Keyword.

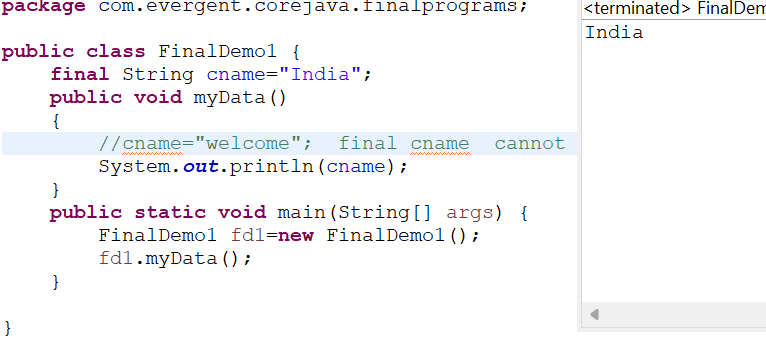
b. We can declare a variable, method, or a class as final.

c. Final variable cannot be modified.

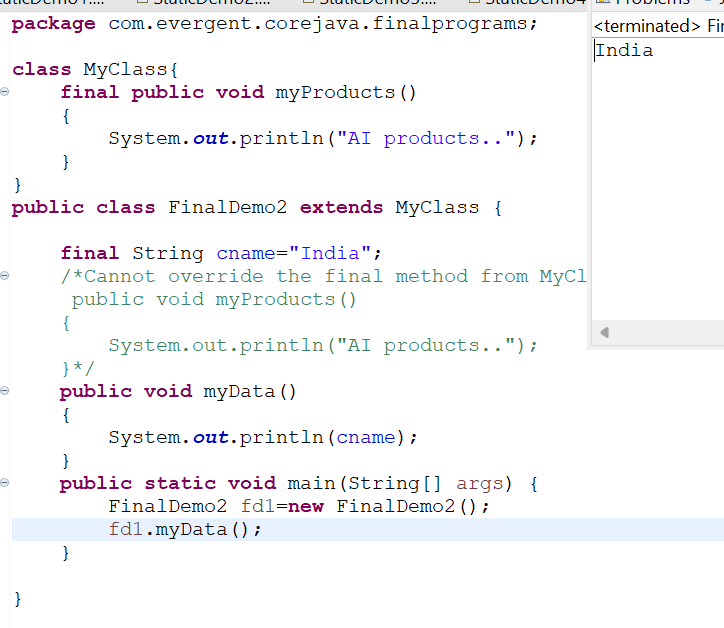
d. Final Method cannot be overrided.

e. Final class cannot be inherited.

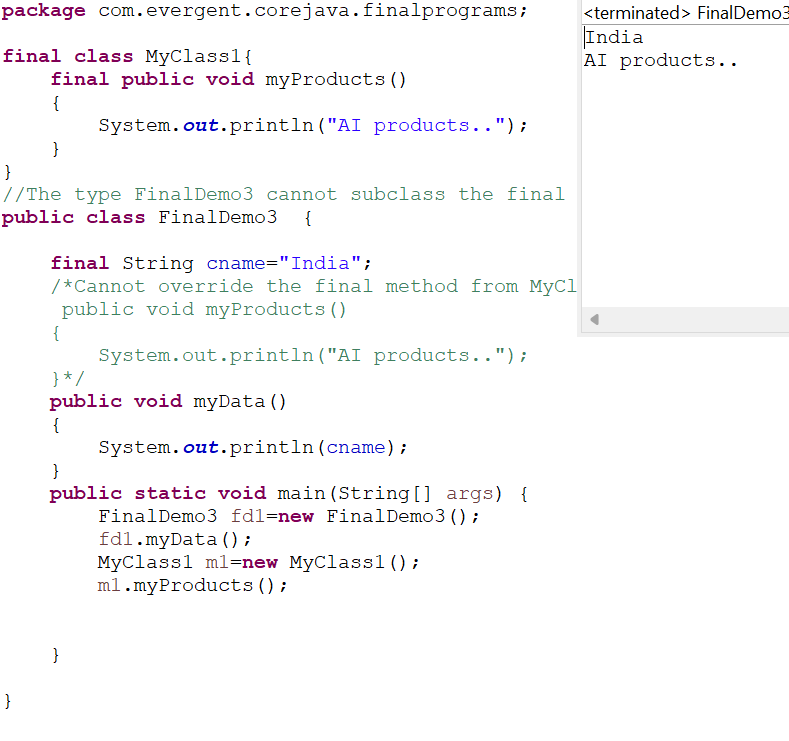
**Program1:**

****

**Program2:**

****

**Program3:**

****

**12/08/24- Day 6**

**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

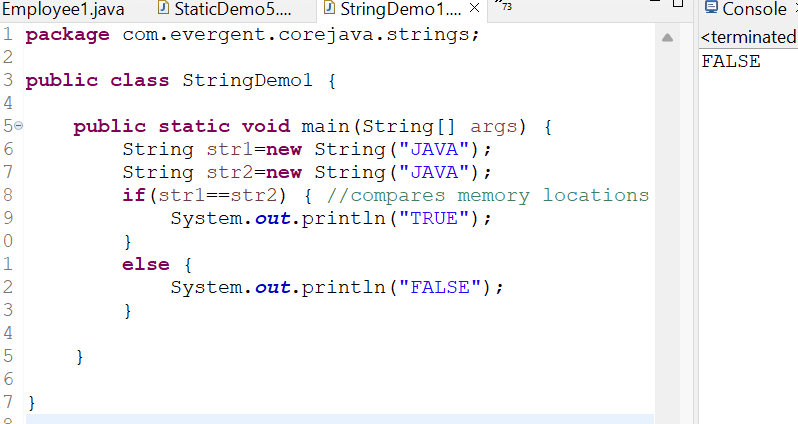
**String Buffer:**

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

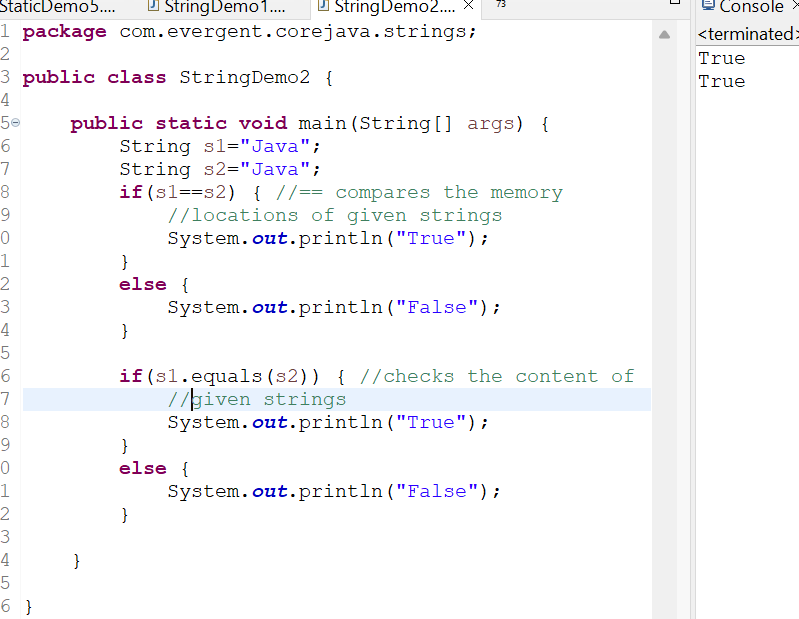
**String Builder:**

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

**Program 1:** using equals() and == operator:



**Program 2:** String initialization without using new keyword.



**Program 3:**

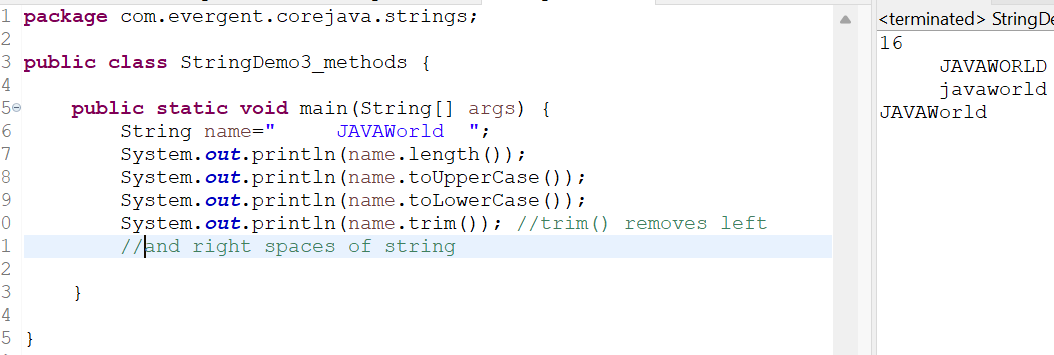
String methods:

-length()

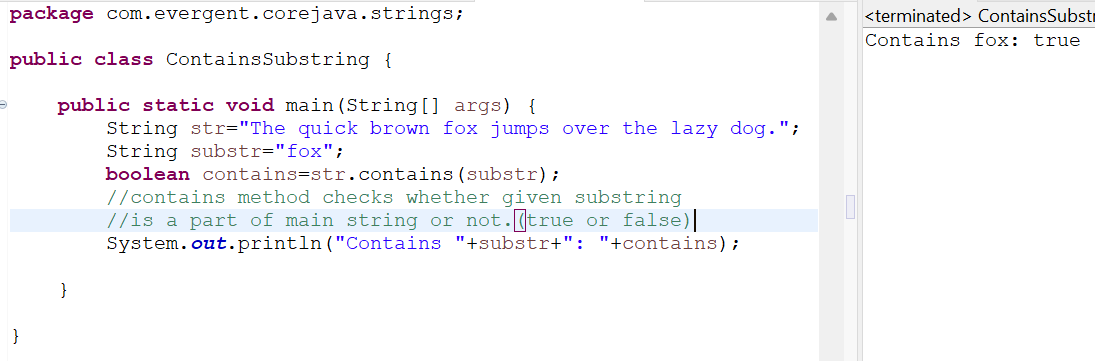
-toLowercase()

-toUpperCase()

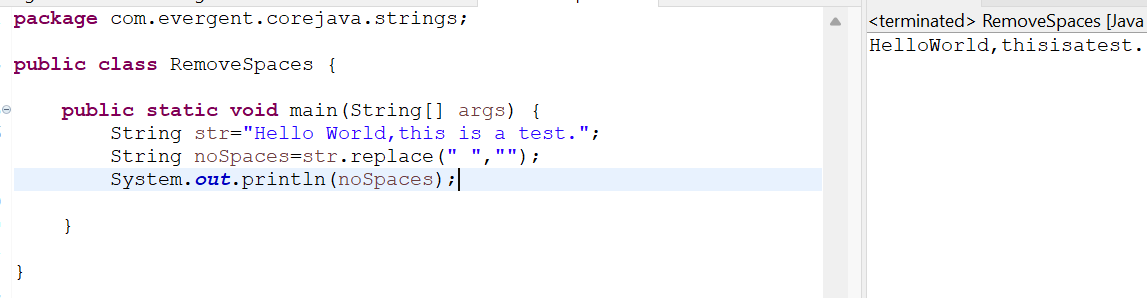
-trim()



**Program 4:**Create a java program that creates a string and checks if it contains specific substring and then prints out the result.

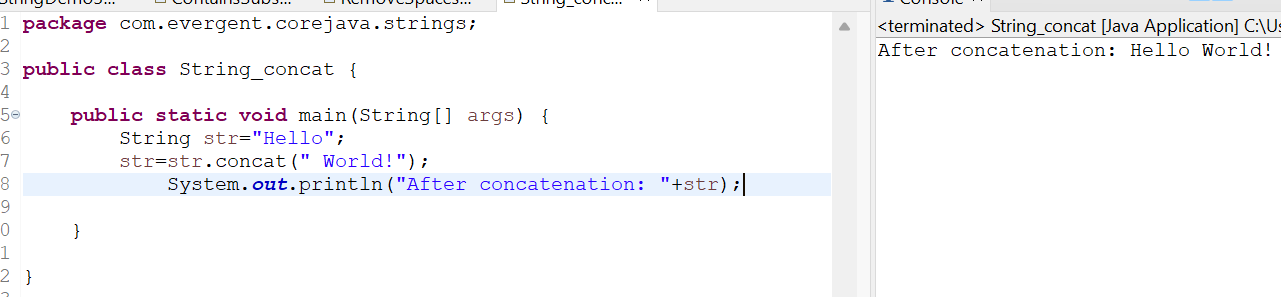


**Program 5:**Write a java program to create a String ,remove all spaces from the string and then print out the result.

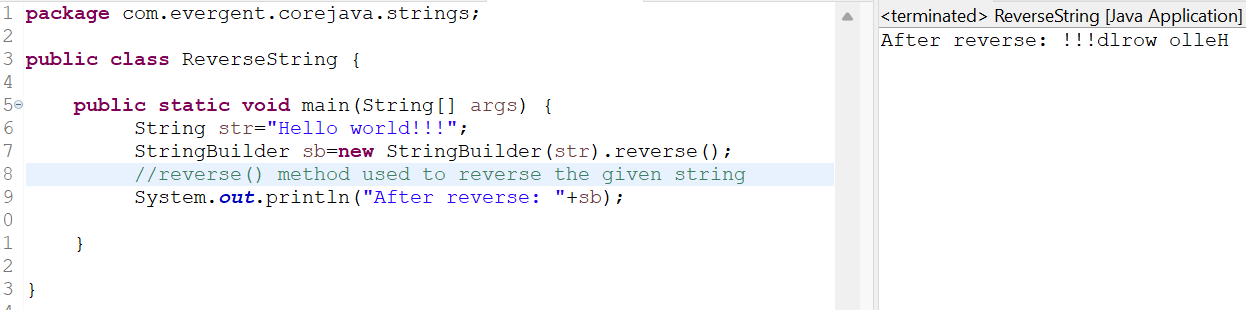


**String concatenation:**

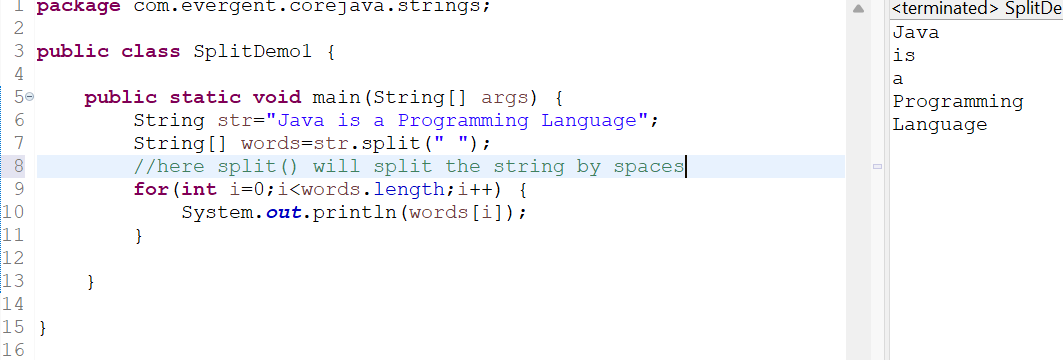
**Program 6:-**Strings can be concatenated using + operator (or) concat() method.



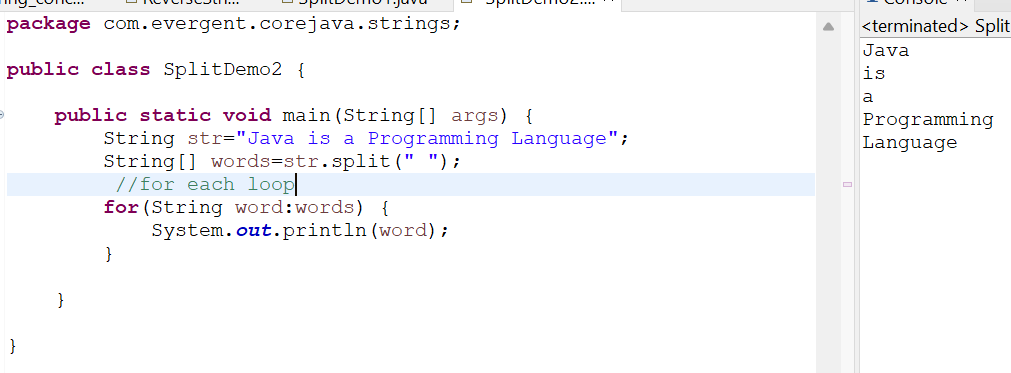
**Program 7:**Reverse of a String:



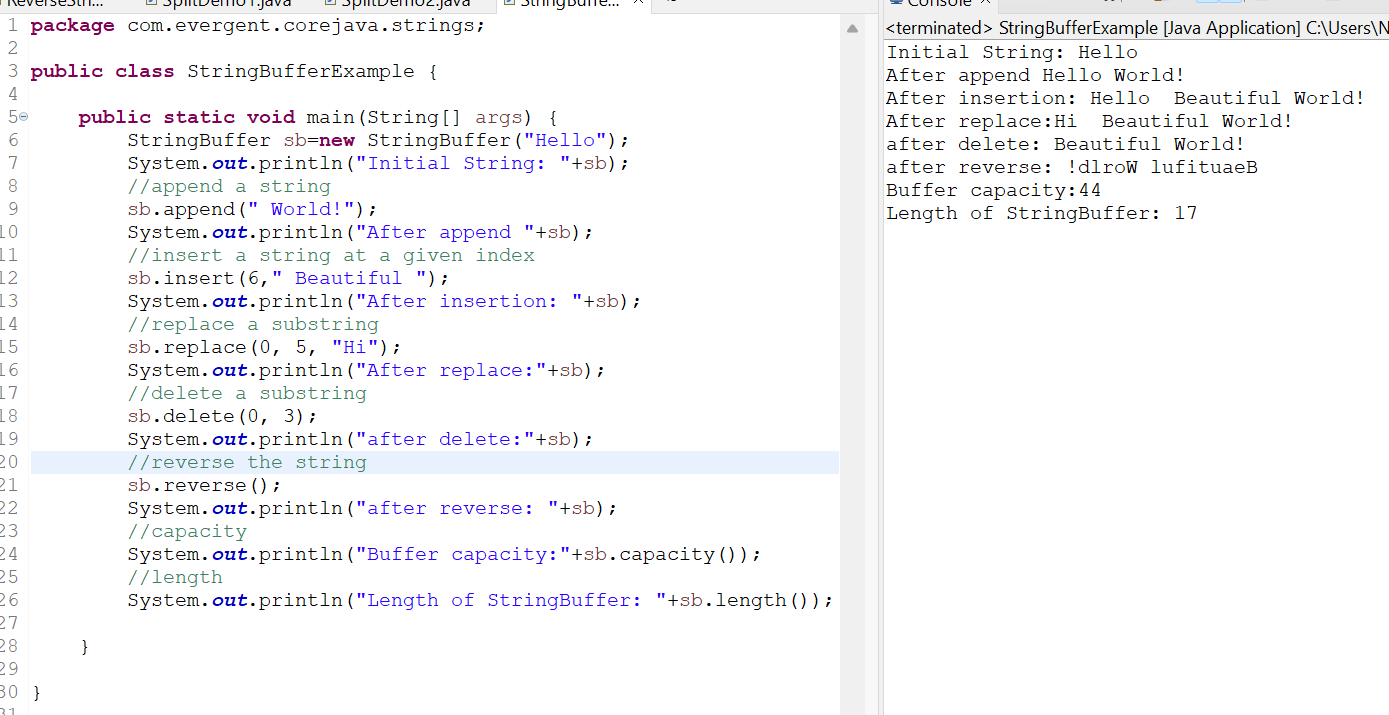
**Program 8:** split given string by white spaces.



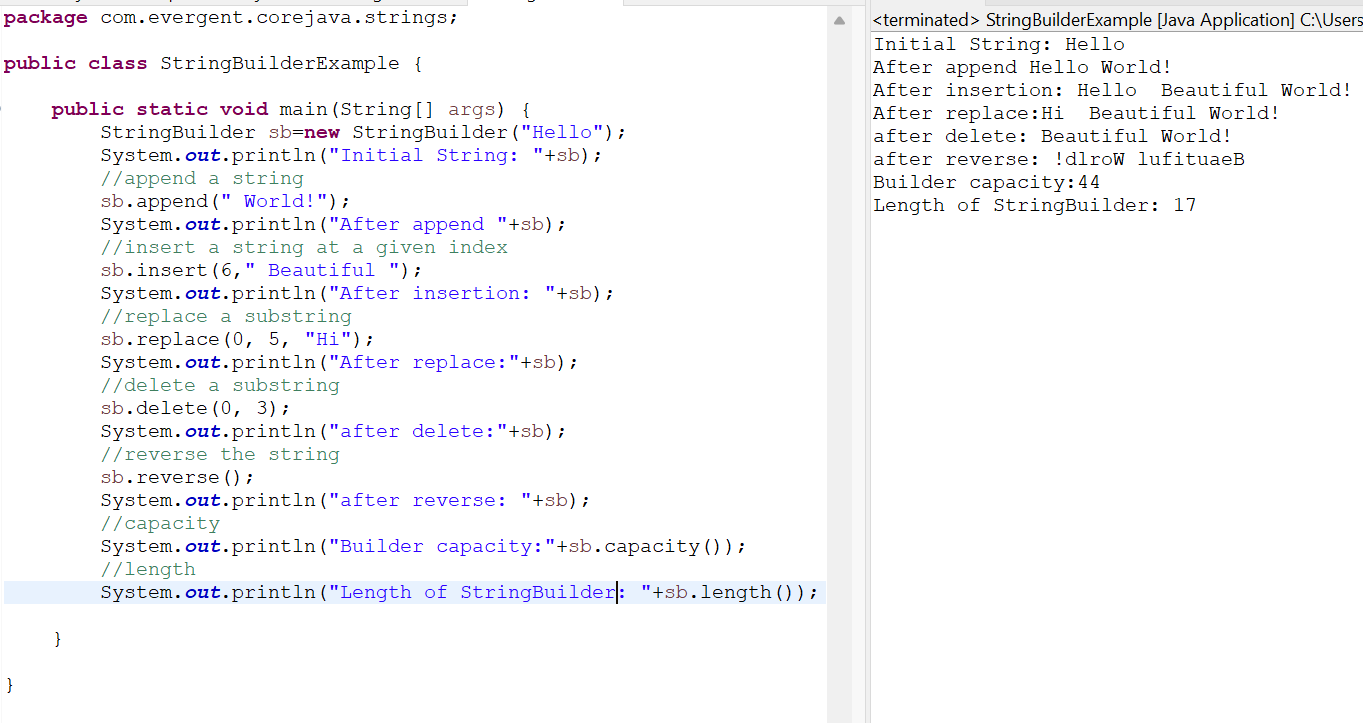
**Program 9:**Using for each loop:



**Program 10:** String Buffer methods:



**Program 11:** String Builder Methods:



**String class important points:**

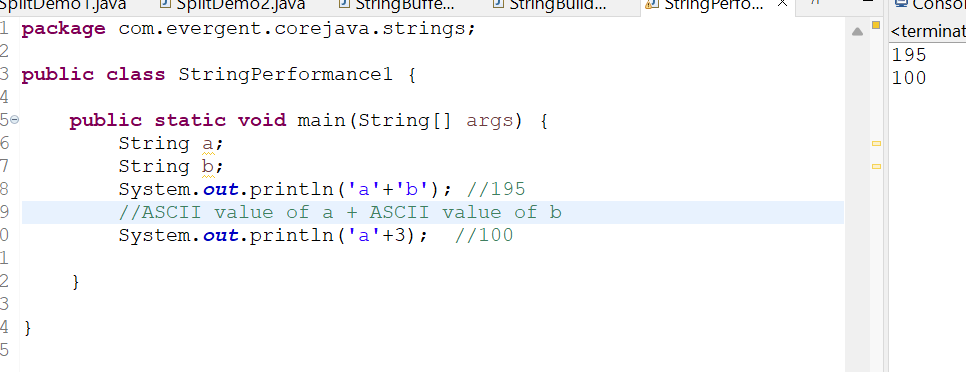
1. In java a string is a sequence of characters ,often used to represent text.
2. Strings are objects in java and are instances of the string class,which is part of the java java.lang package
3. Key features of strings in java:

**Immutable:**once a string object is created ,it cannot be changed .

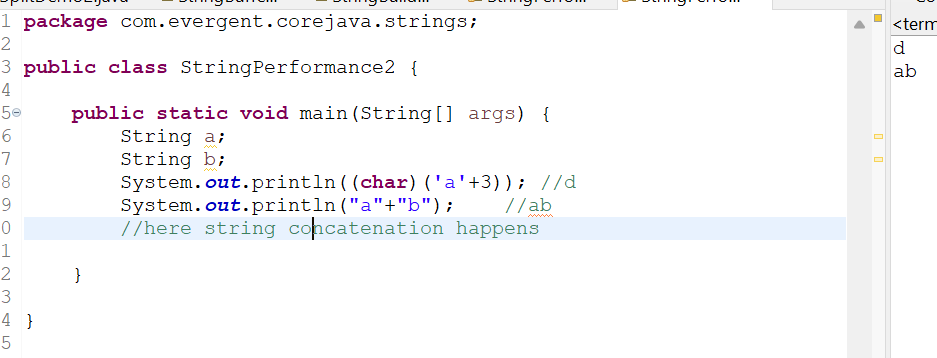
1. Java optimizes memory usage by storing strings in special area of memory as string pool
2. If two strings have the same value and are created without using new keyword they will refer to same object in the string pool.
3. We can create a string in java in multiple ways:
4. Using string literals :str=”hello world”;
5. Using the new keyword

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

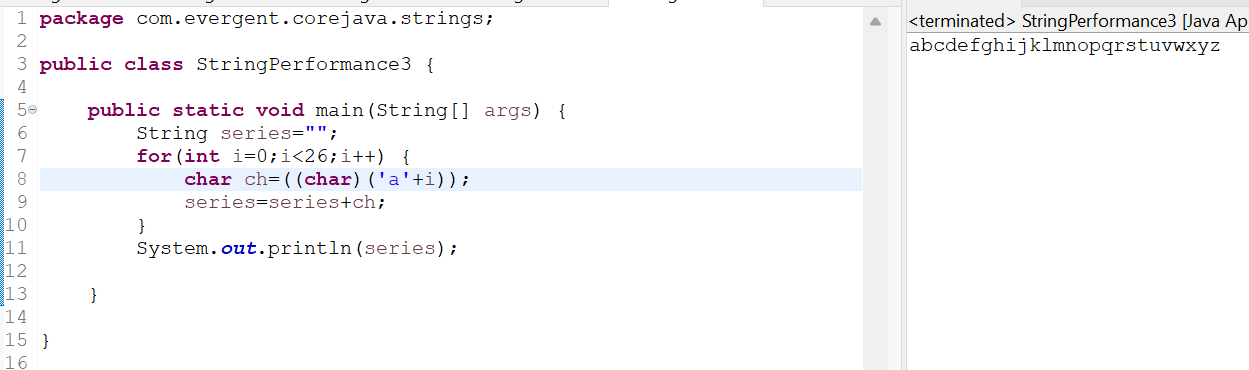
**String Performance 1:**



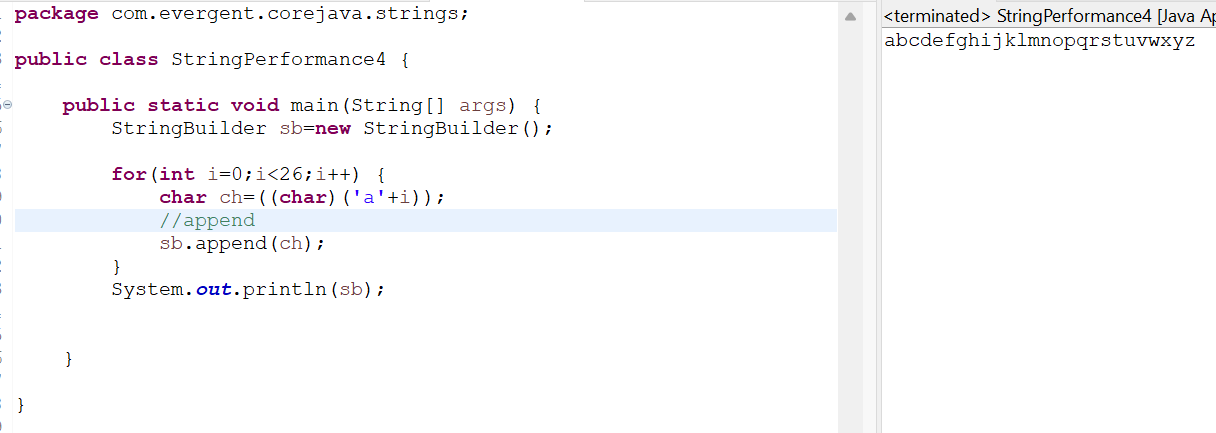
**String Perfomance 2:**



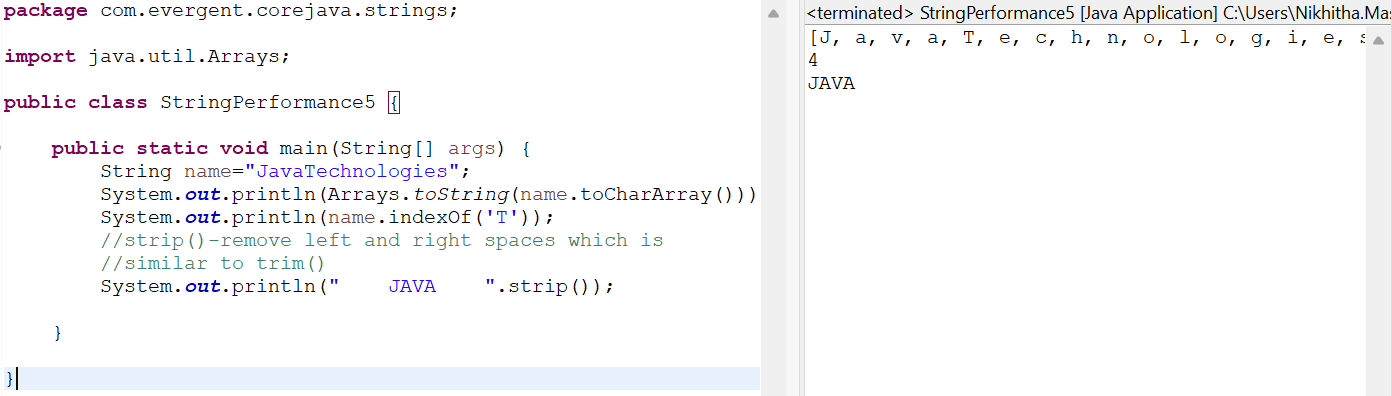
**String Performance 3:**



**String performance 4:**



**String Performance 5:**



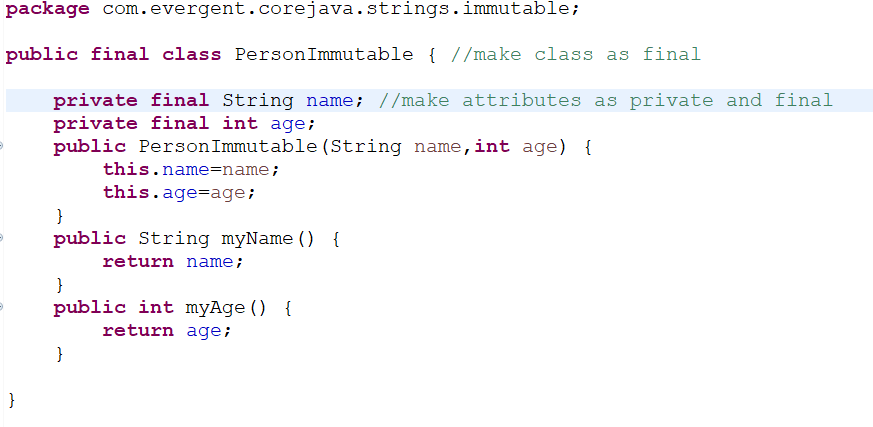
**Date : 13/08/2024 (DAY-7)**

* final variables also initializes while constructor calling.

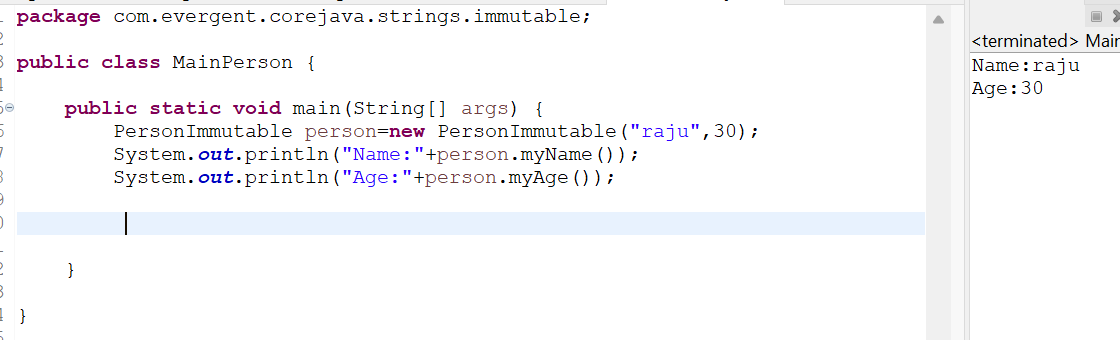
1. Can you make a class as Immutable?

Yes, we can declare our class as a final class and attributes as private and final.

**Program 1:**



**Program 2:**  MainPerson class:

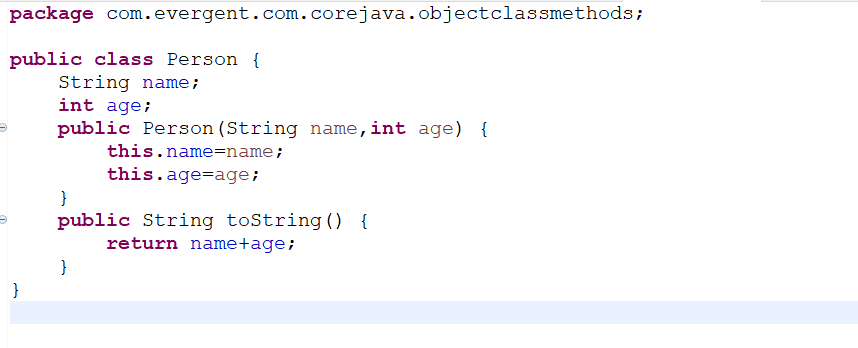


**Program 3:**

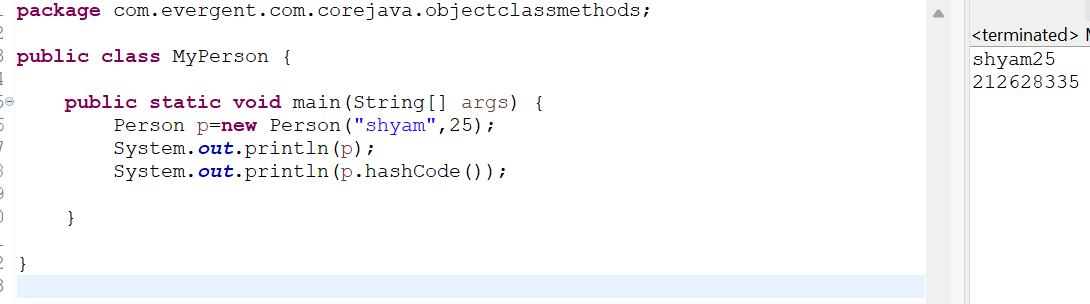
Object class methods:

-toString()

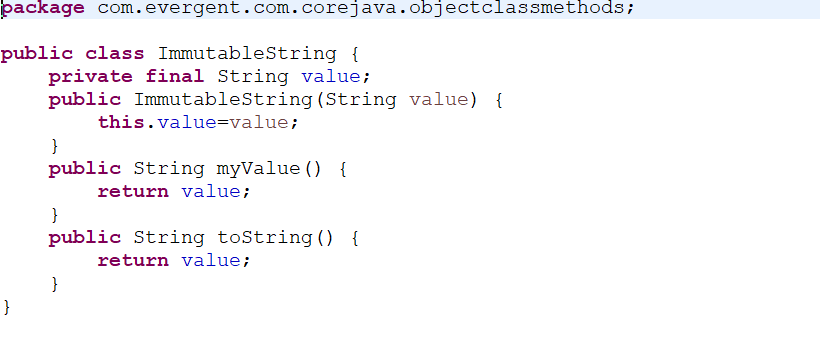
-hashCode();



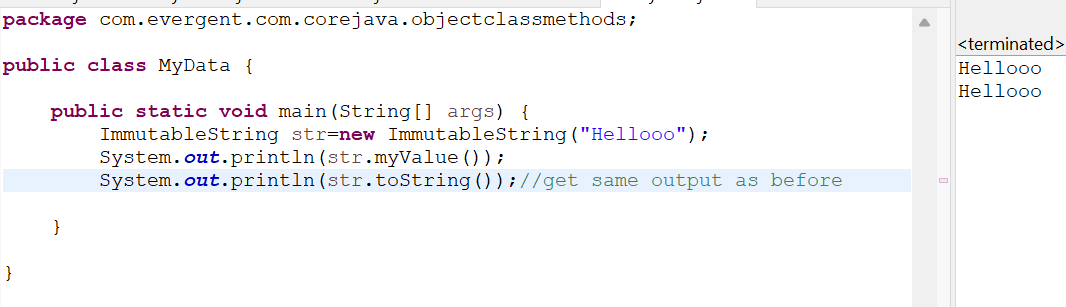
**Program 4:** MyPerson class:



**Program 5:** ImmutableString:



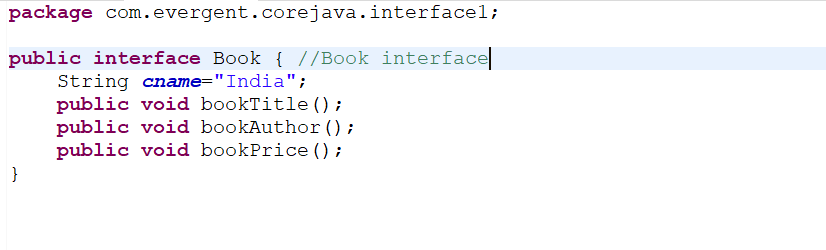
**Program 6:** MyData class:



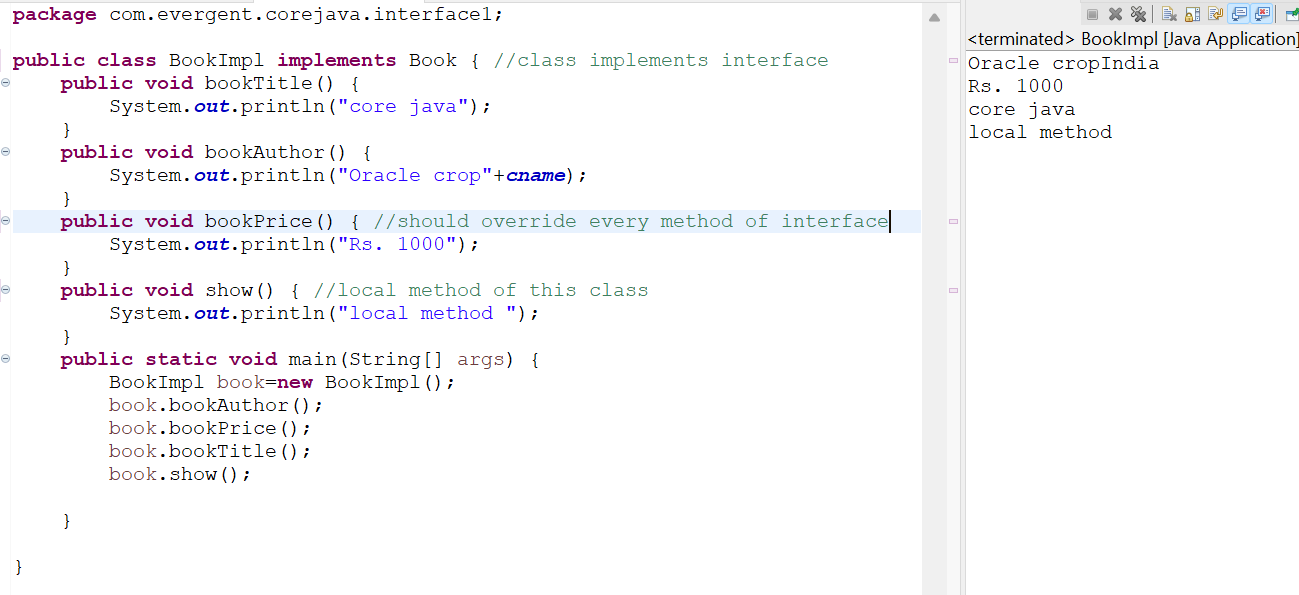
**Interfaces:**

1. Interface is a keyword.
2. We can declare methods signature only but not implementation.
3. By default all interface methods are abstract.
4. If any class implements interface, that class should be override all interface methods, otherwise the class will be showing compile time error.
5. We can’t create object to interface. But we can create reference to interface.
6. We can declare variables inside interface, all are public static final (default).
7. Java will support Multiple Inheritance through interfaces.
8. One class can implements Interface.
9. One interface can extends other interface.

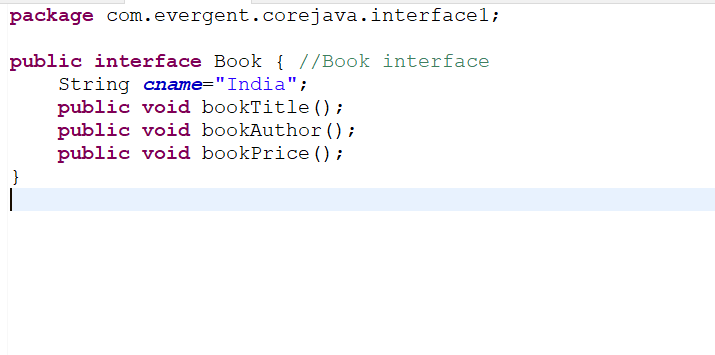
**Program 1:** Book interface

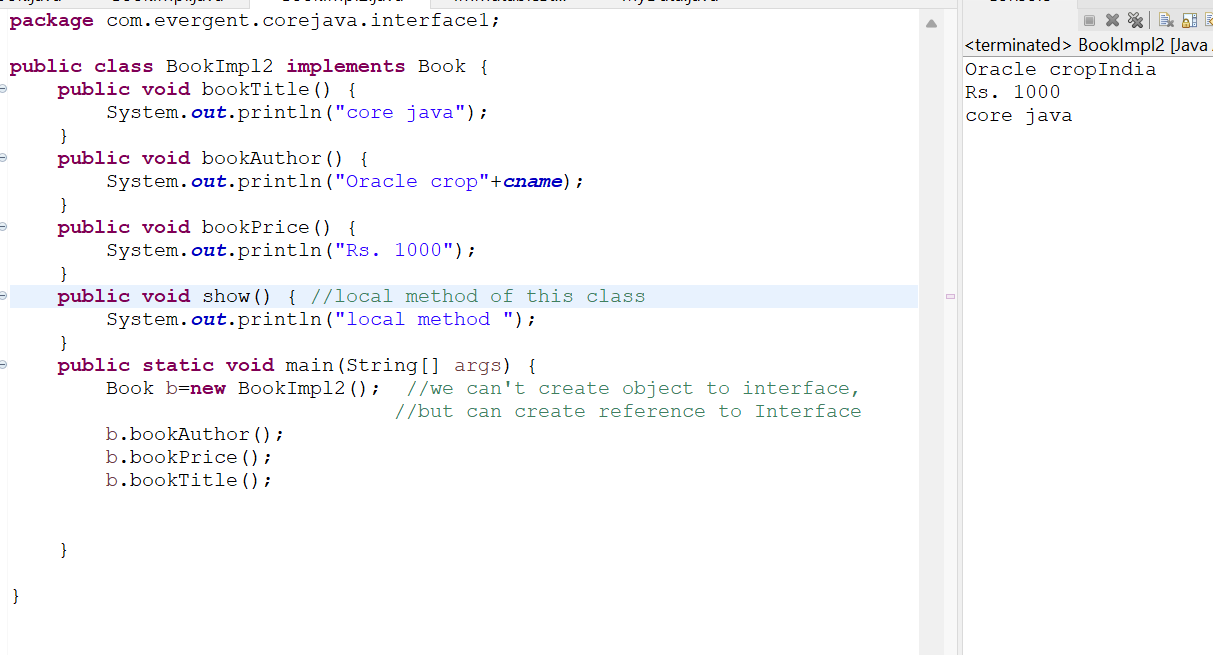


**Program 2:** BookImpl implements Book:



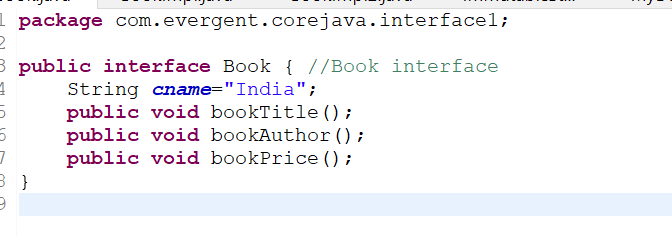
**Program 3:**  Book Interface



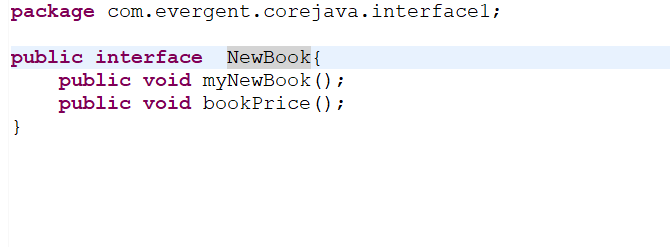


**Program 4:**

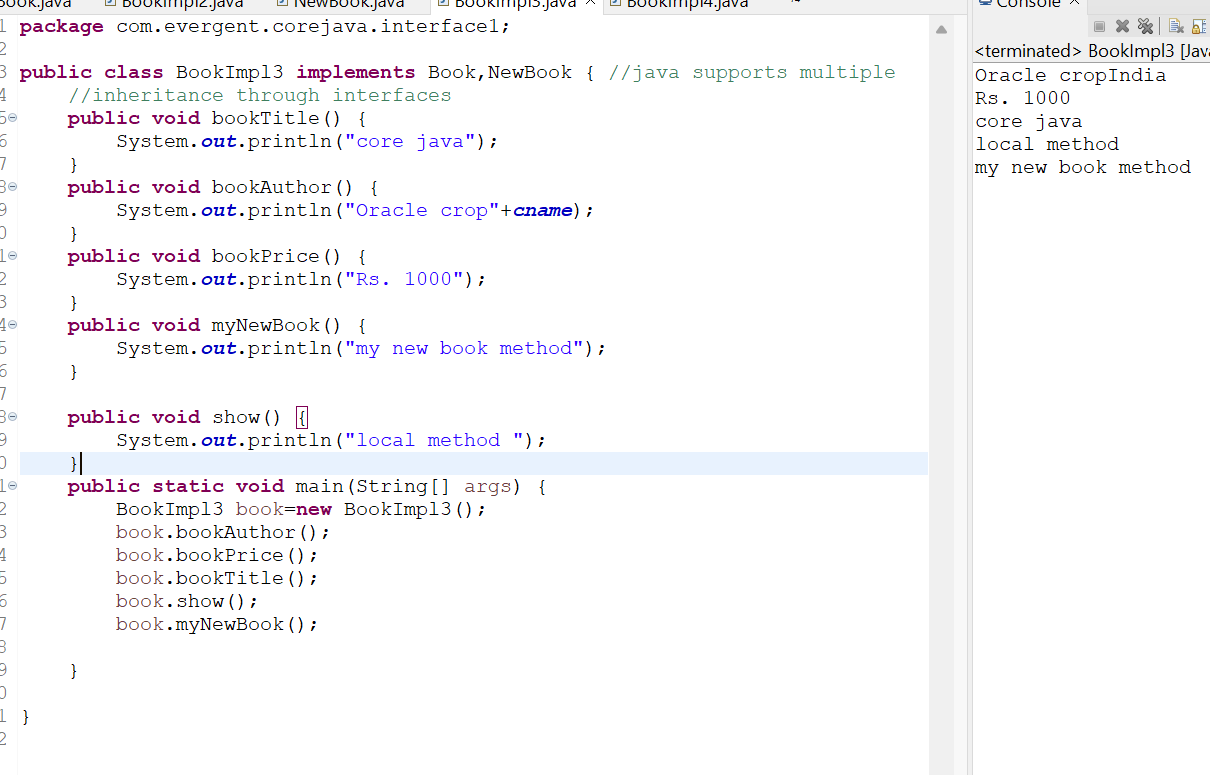
Book Interface:



NewBook Interface:

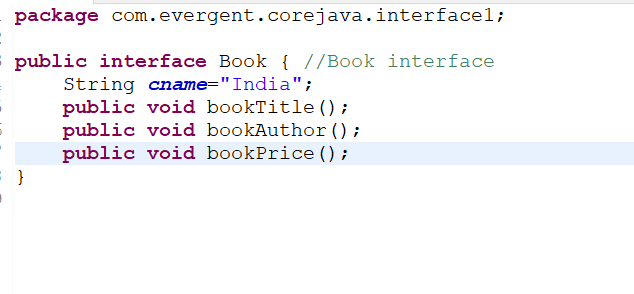


BookImpl3 implements Book, NewBook:

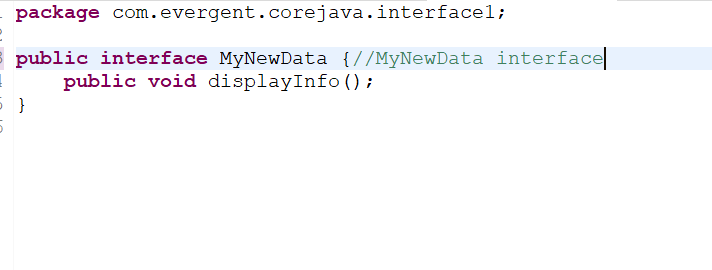


**Program 5:**

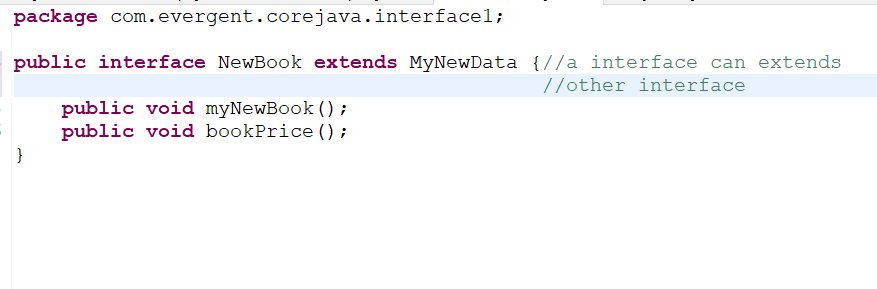
Book Interface:



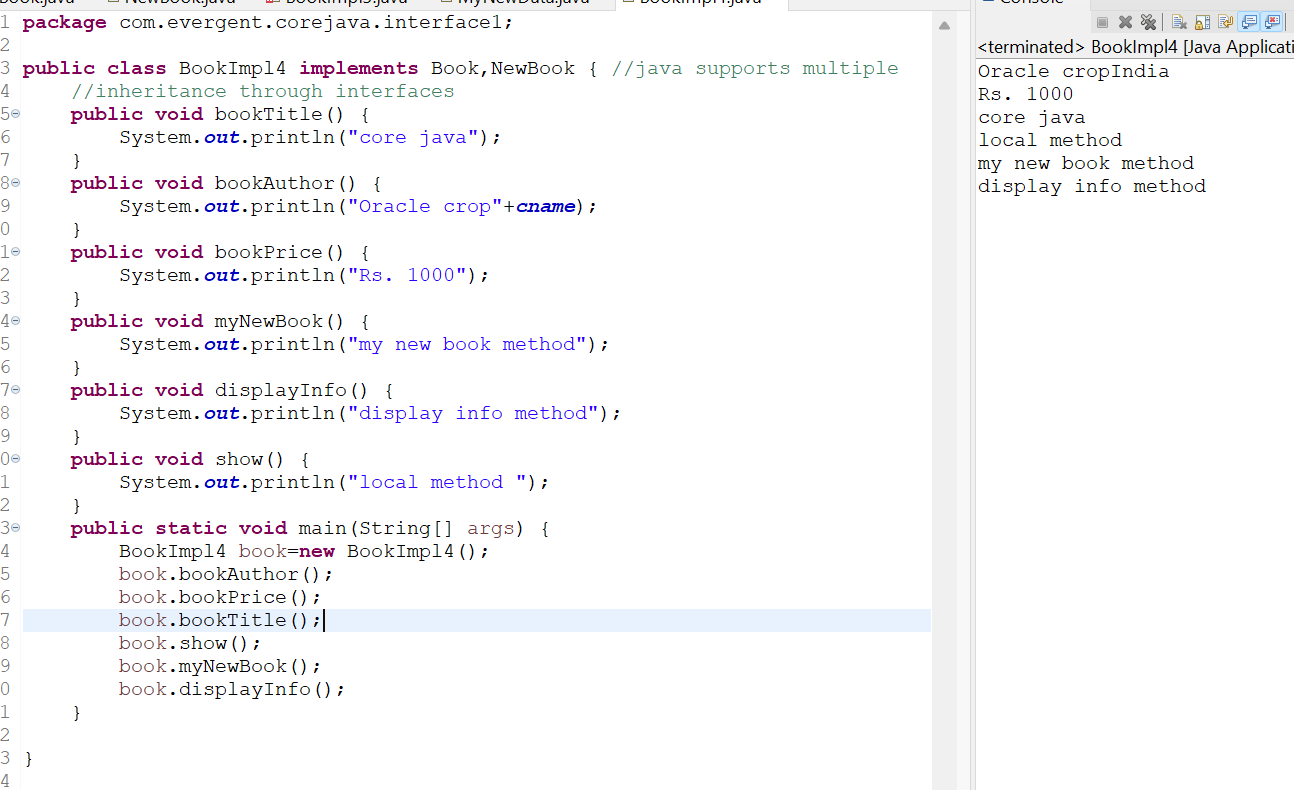
MyNewData interface:



NewBook interface extends MyNewData interface:



BookImpl4 implements Book,NewBook:



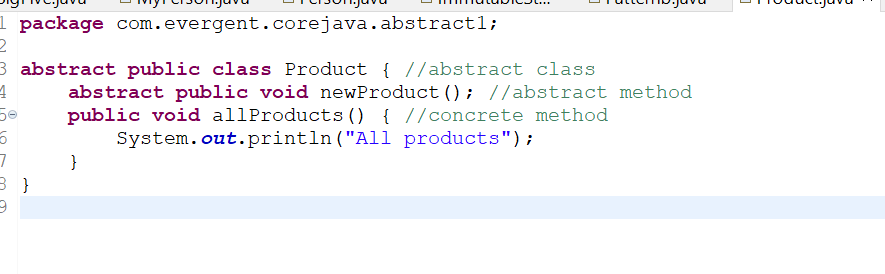
**Date: 14/08/2024 (DAY-8)**

**Abstract Class:**

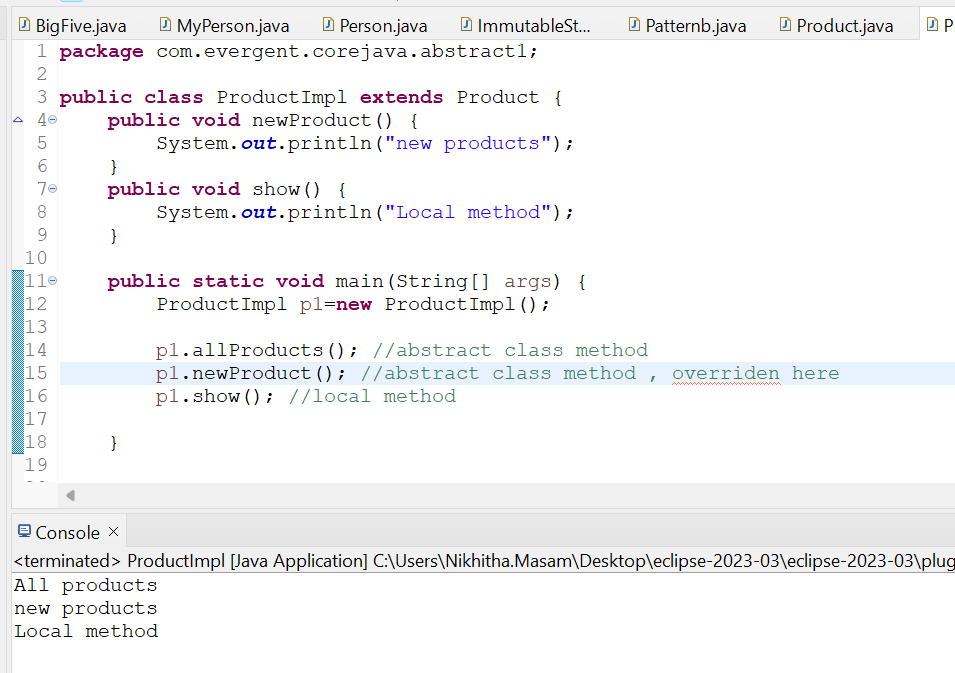
1. Abstract is a keyword.
2. Abstract class having abstract methods and concrete (implemented) methods.
3. If any class having one abstract method, that class should be declared as a abstract keyword, otherwise the class will be showing compile time error.
4. If any class extends abstract class, that class should be override all abstract methods. Otherwise the class will be showing compile time error.
5. We can’t create object to abstract class. But we can create reference to abstract class.
6. We can declare a class as abstract, with zero abstract methods.

**Program 1:**

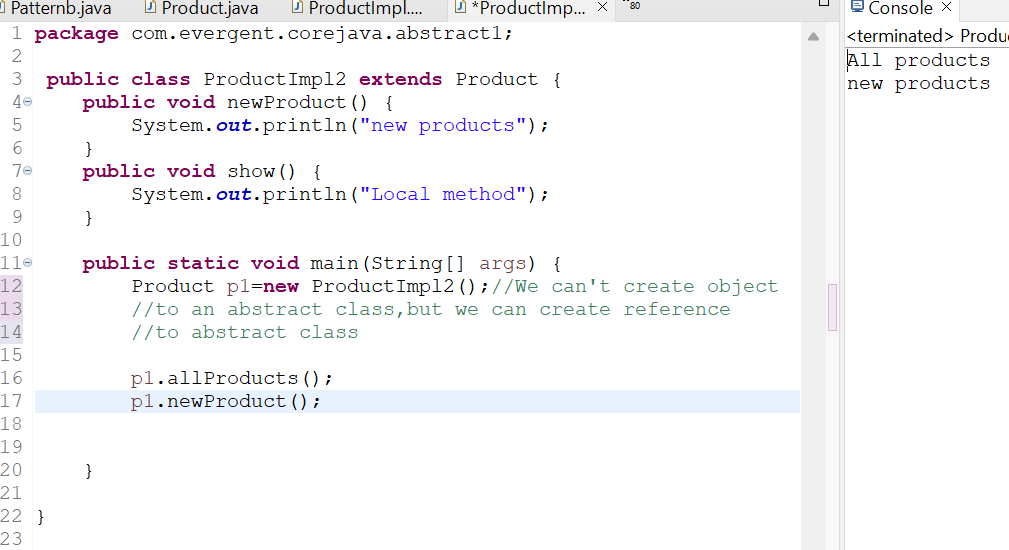
Product Interface:



Product Implementation :

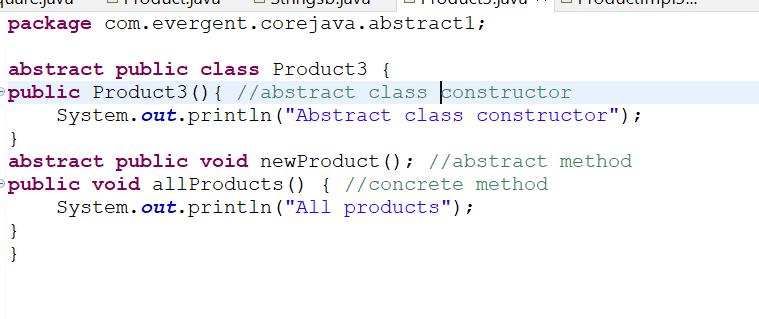


**Program 2:**

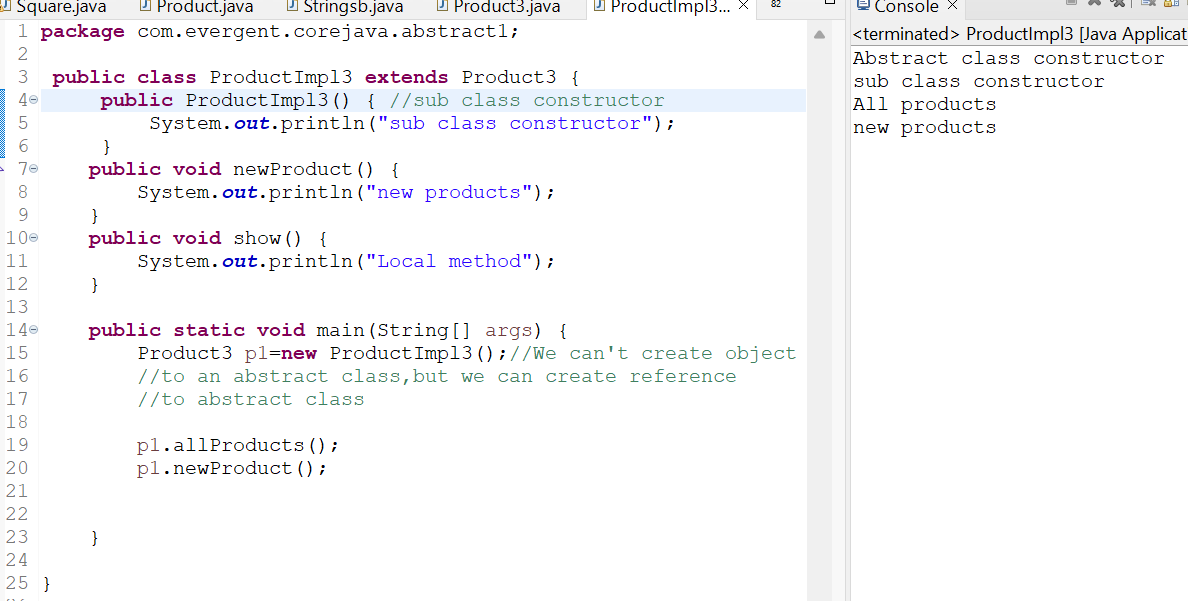


**Constructor within Abstract class:**

**Program 3:**



ProductIml3:



**Date : 16/08/2024 (Day -9)**

* Coding Test on Core Java
* Application3(Bank Application) Explanation

**Date : 19/08/2024 (Day -10)**

**Exception Handling**

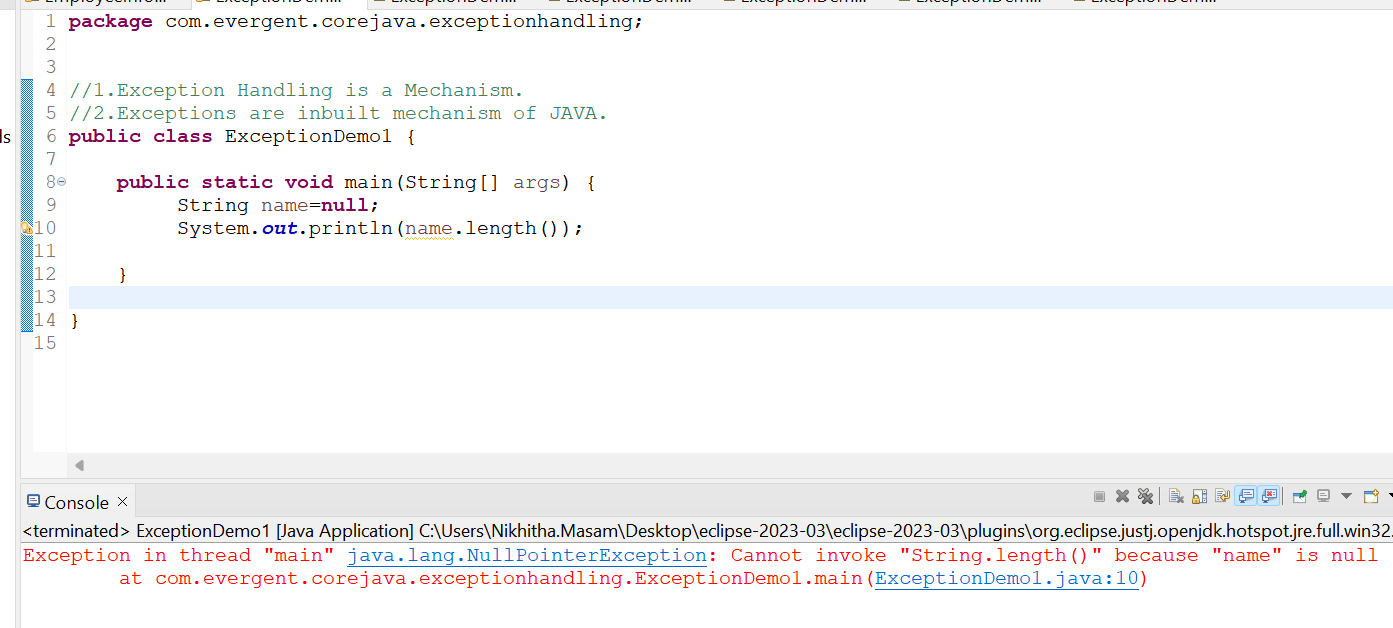
1. Exception Handling is a Mechanism.
2. Exceptions are inbuilt mechanism of JAVA.
3. All Exception are executed while abnormal conditions only.
4. Normal flow it won’t execute any exceptions.
5. Once any exceptions are occurring in java, then remaining lines of code is unreachable.
6. Java.lang.Throwable is super class for Exception and Error.
7. There are two types of Exceptions in JAVA.
8. Checked Exceptions
9. Unchecked Exceptions
10. All Checked Exceptions are compile time exceptions.
11. All Unchecked Exceptions are Run time exceptions.
12. There are 5 keywords in Exception Handling:

a.Try b. catch() c.finally d. throws e.throw

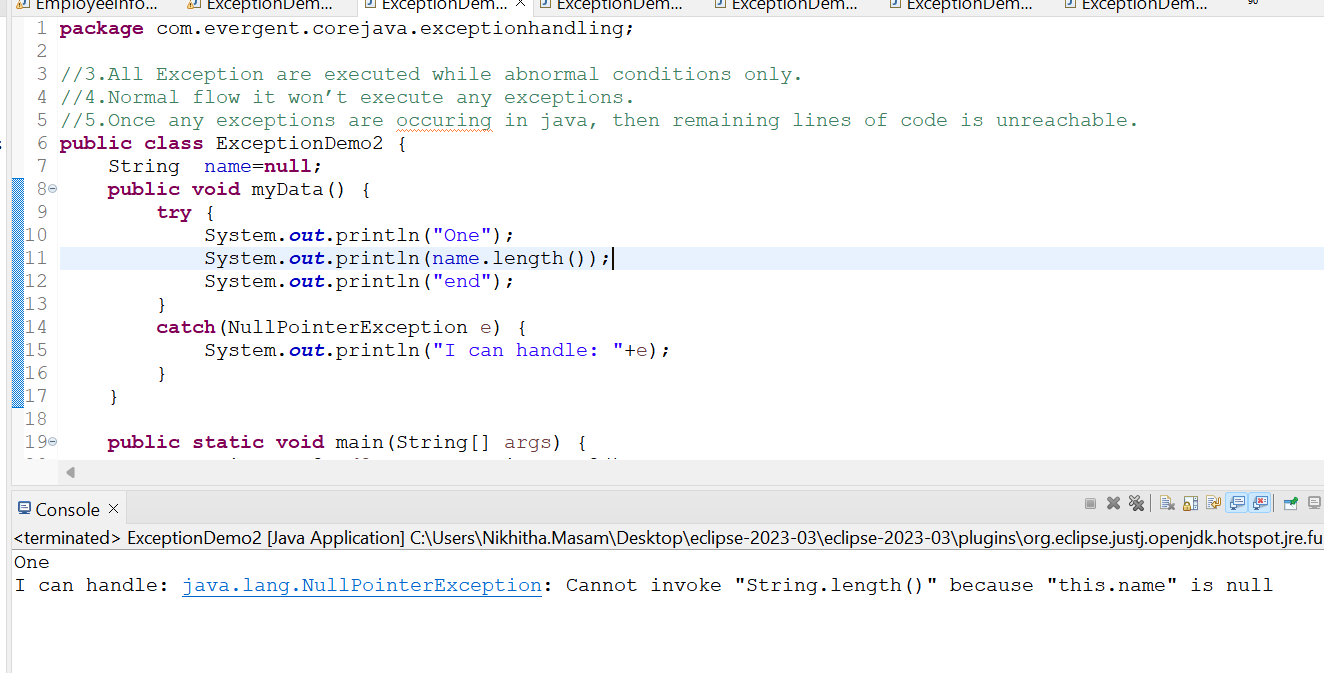
1. Try is for business logic.
2. Catch is for handling exceptions.
3. Finally is block, if exception is occur or not finally block will be executed.
4. Throws an exception will be executed method by method.
5. Throw is for runtime exceptions and will call predefined exceptions or user defined exceptions.
6. Try followed by either catch block or finally block.
7. We should follow exception hierarchy.
8. We can create our own (user defined) exceptions.
9. Our own exception extends Exception or Runtime Exception.
10. All Exception classes are in to java.lang package.
11. There is two exceptions in class, Developer should be handle one after one.
12. Developer can’t handle Errors.

**Programs :**

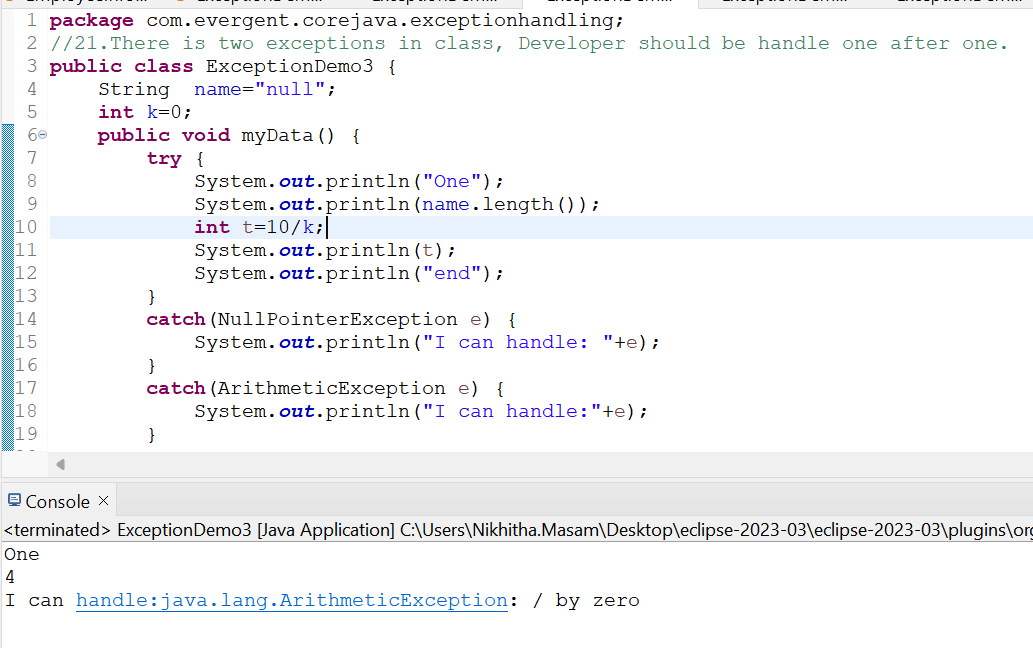
**ExceptionDemo1:**



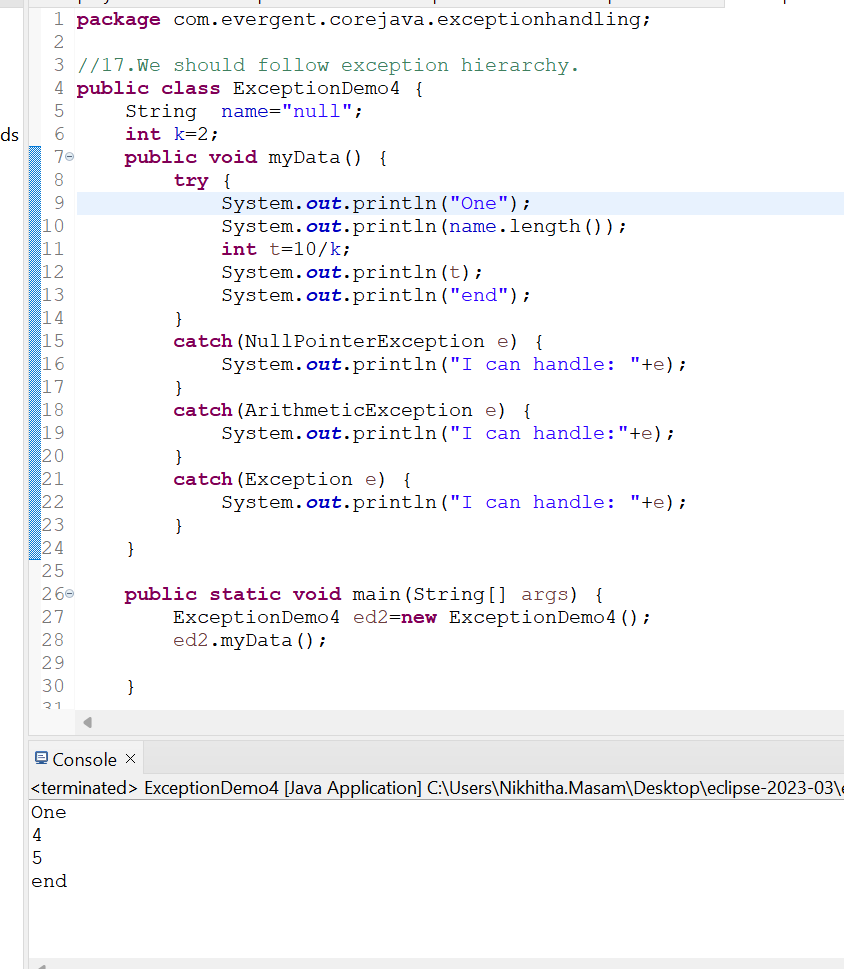
**ExceptionDemo2:**



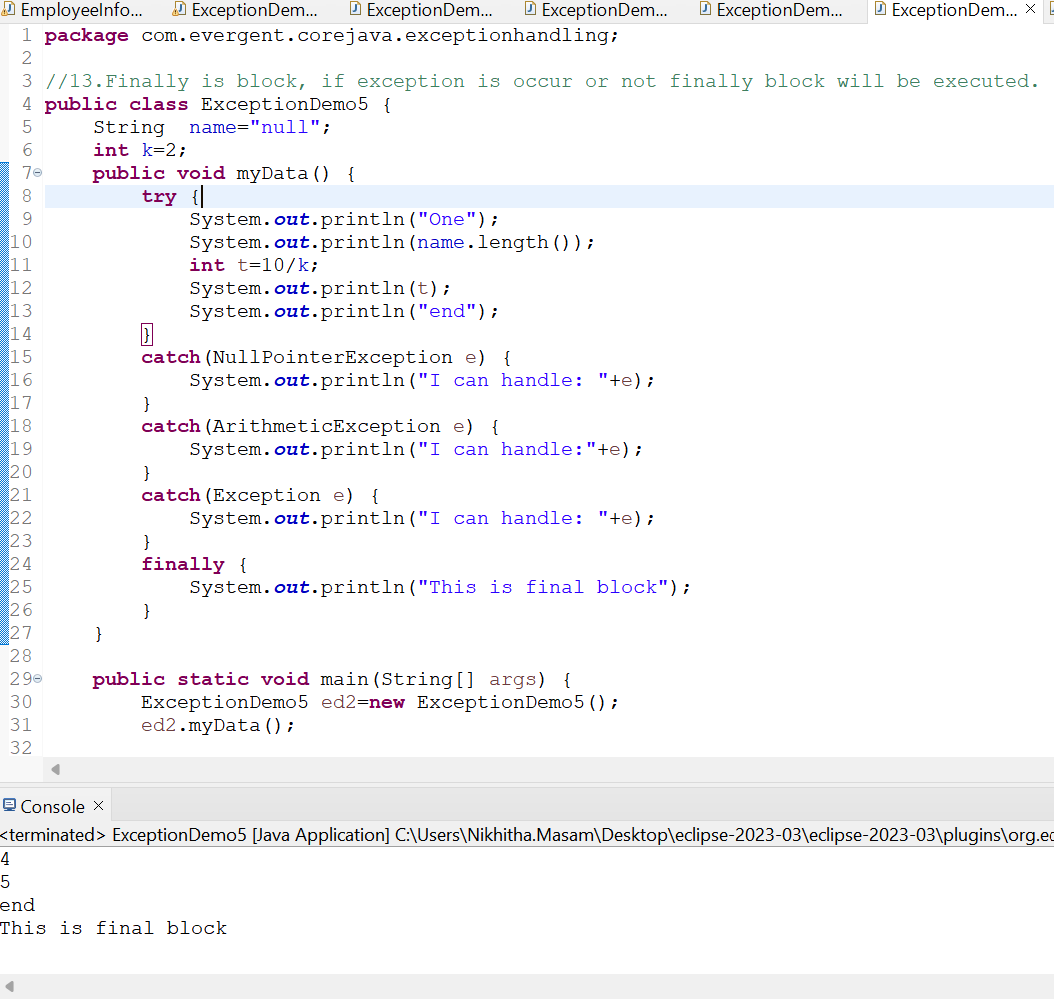
**ExceptionDemo3:**



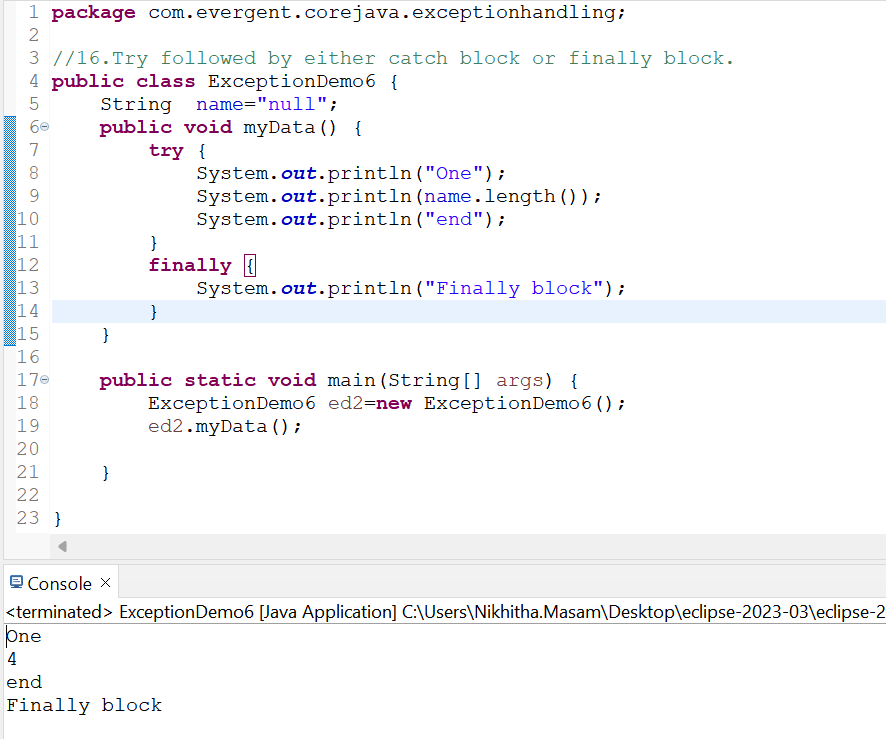
**ExceptionDemo4:**



**ExceptionDemo5:**

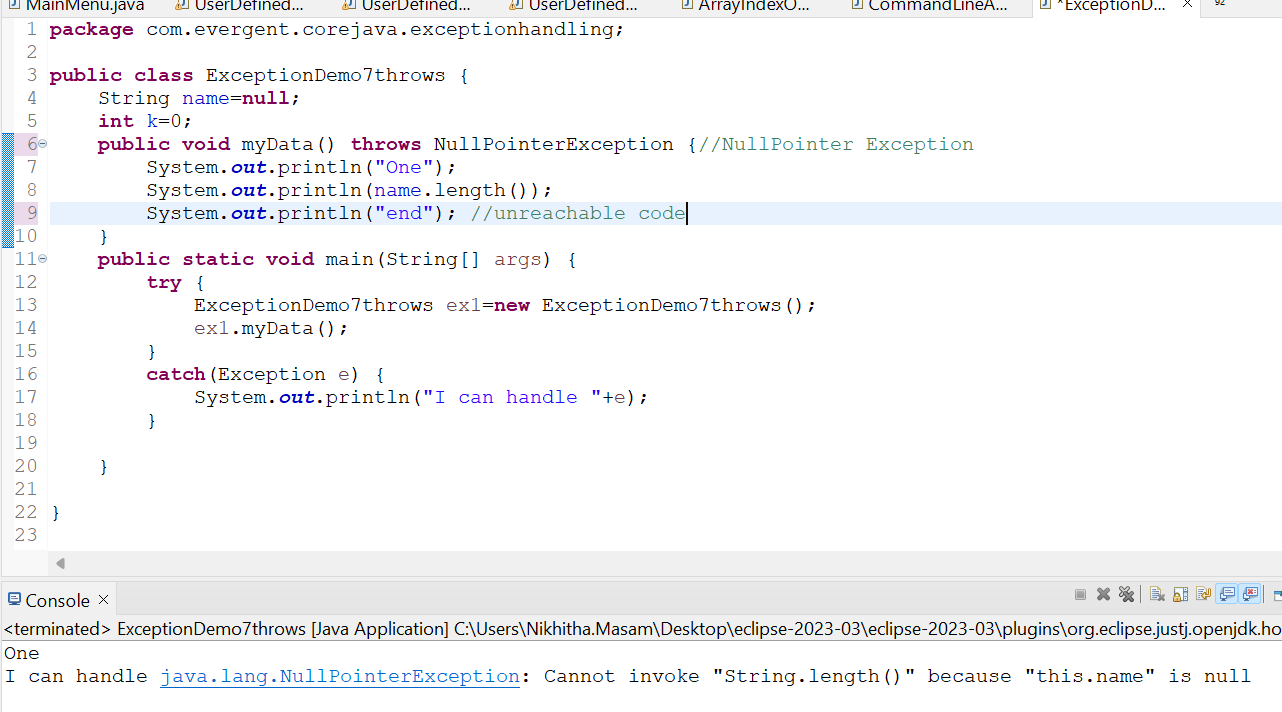


**ExceptionDemo6:**

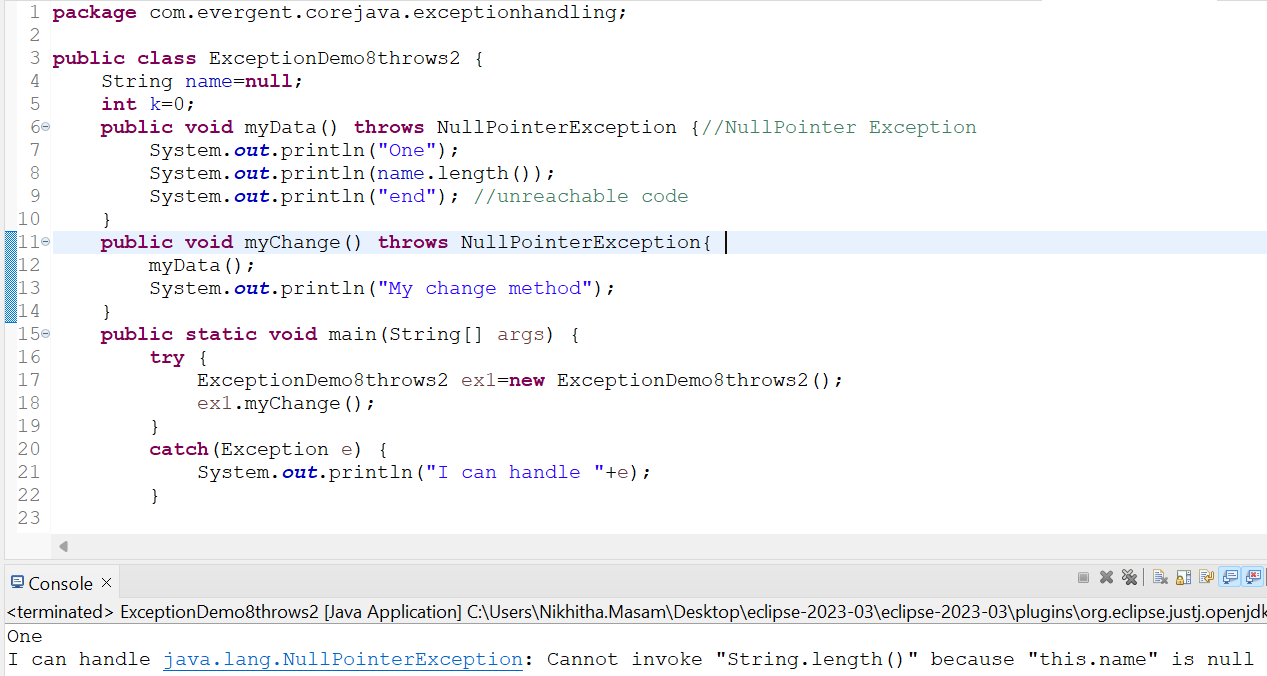


**Date:20/08/2024 (DAY-11)**

**Program 1:**



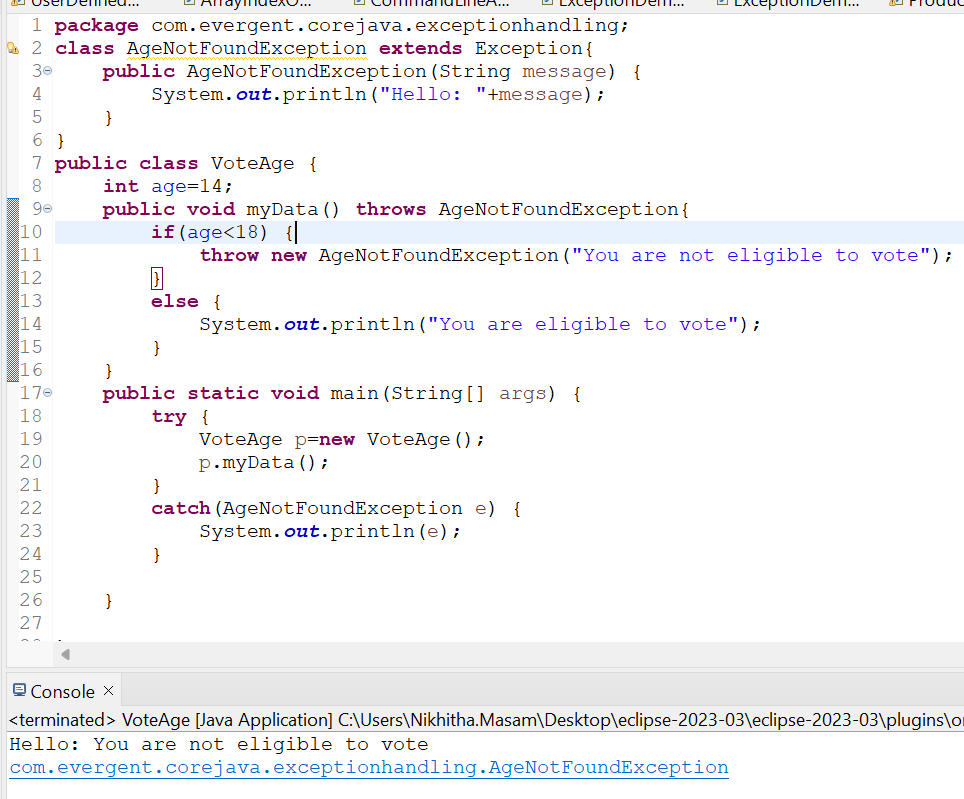
**Program 2:**



**Program 3:**

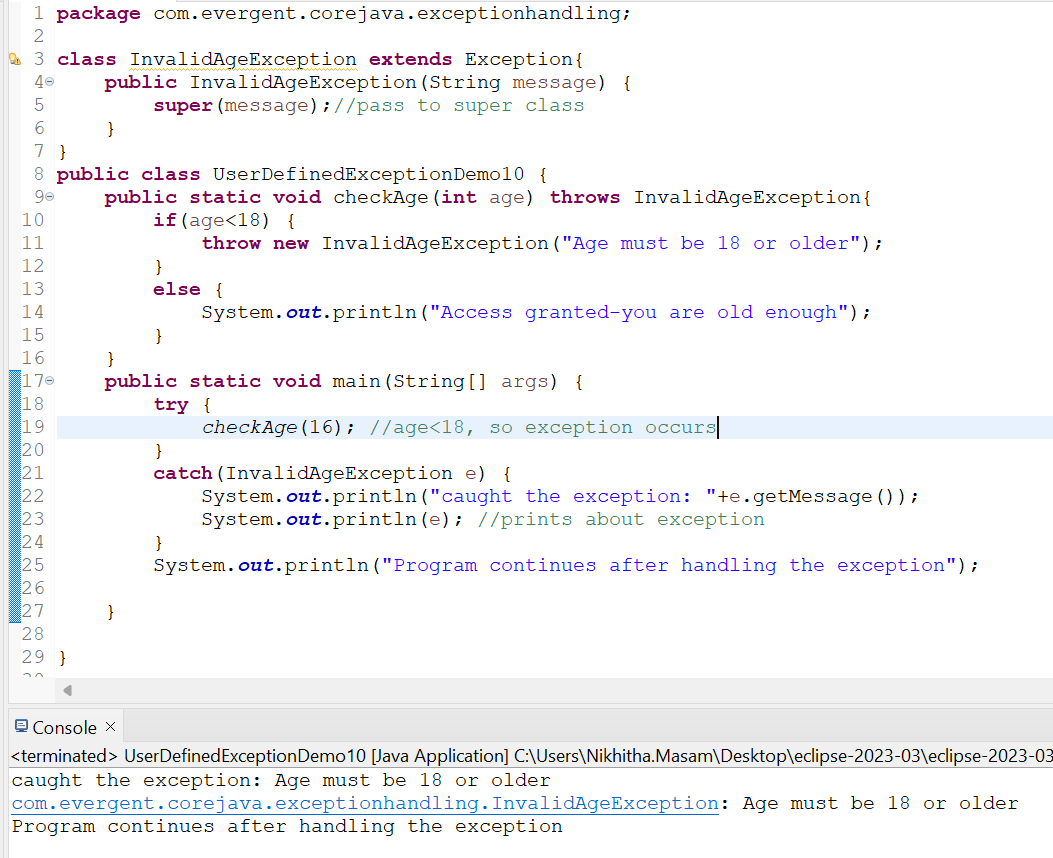


**AgeNotFoundException:**



**Date: 21/08/2024 - (Day12)**

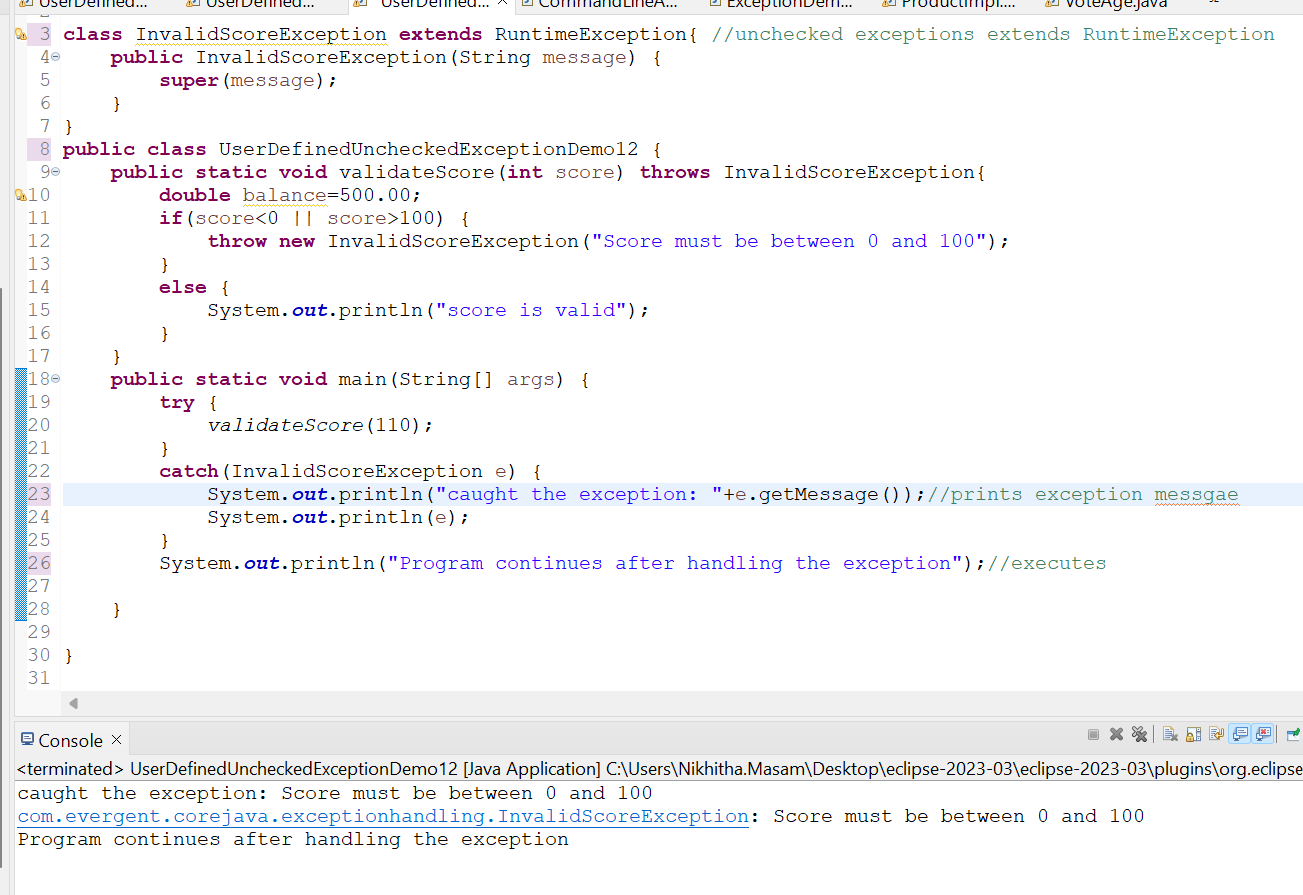
**InvalidAgeException:**



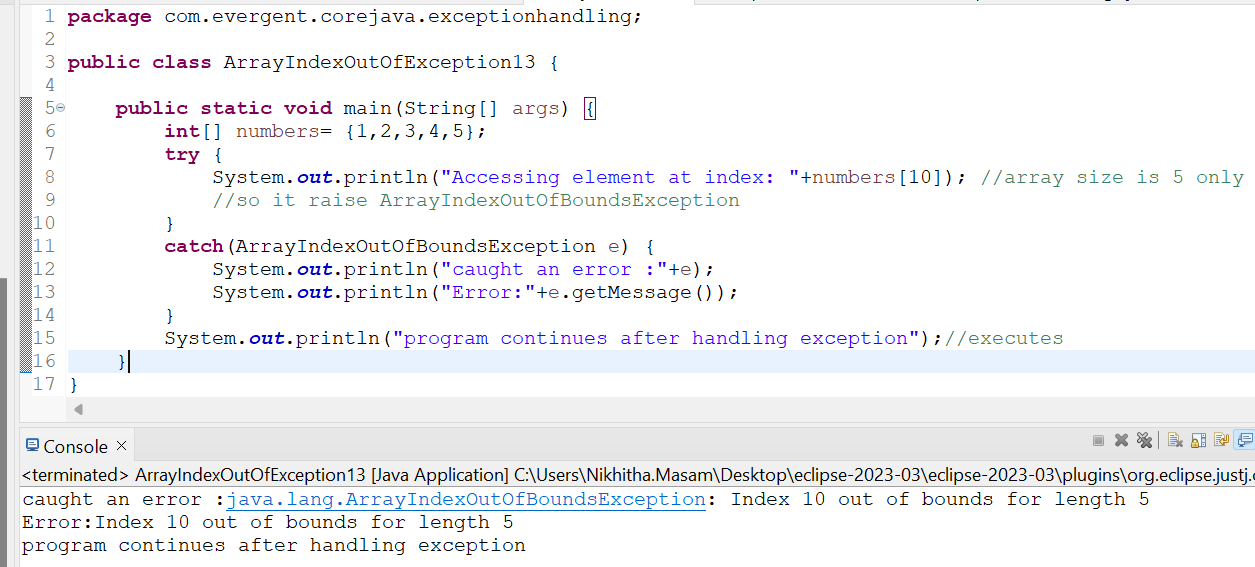
**InsufficientFundsException:**



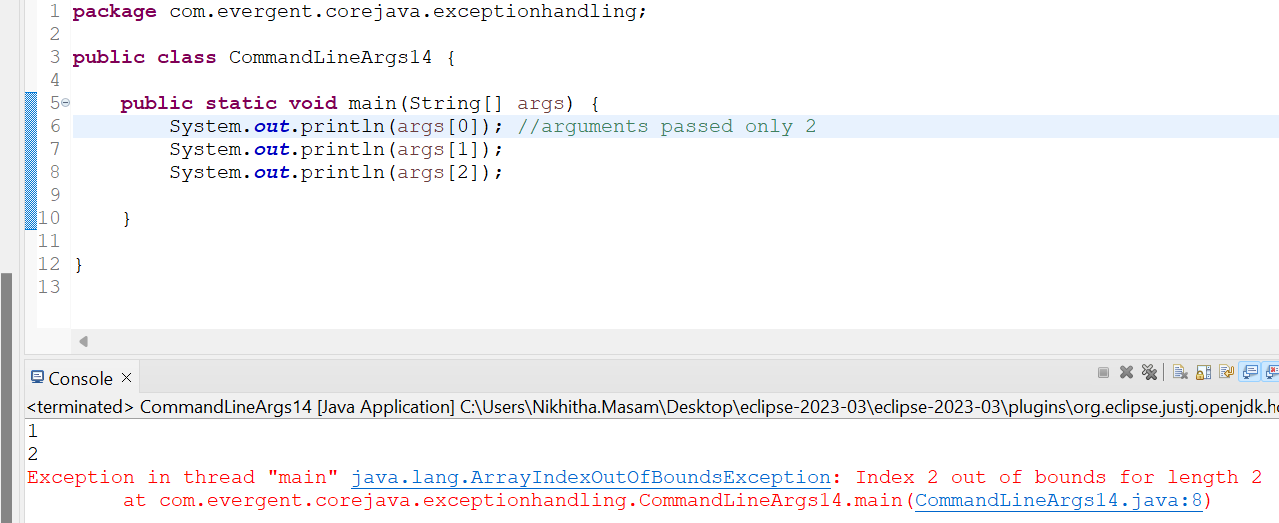
**InvalidScoreException**:



**ArrayIndexOutOfBoundsException**:



**CommandLineArgs**:



**FileNotFoundException:**



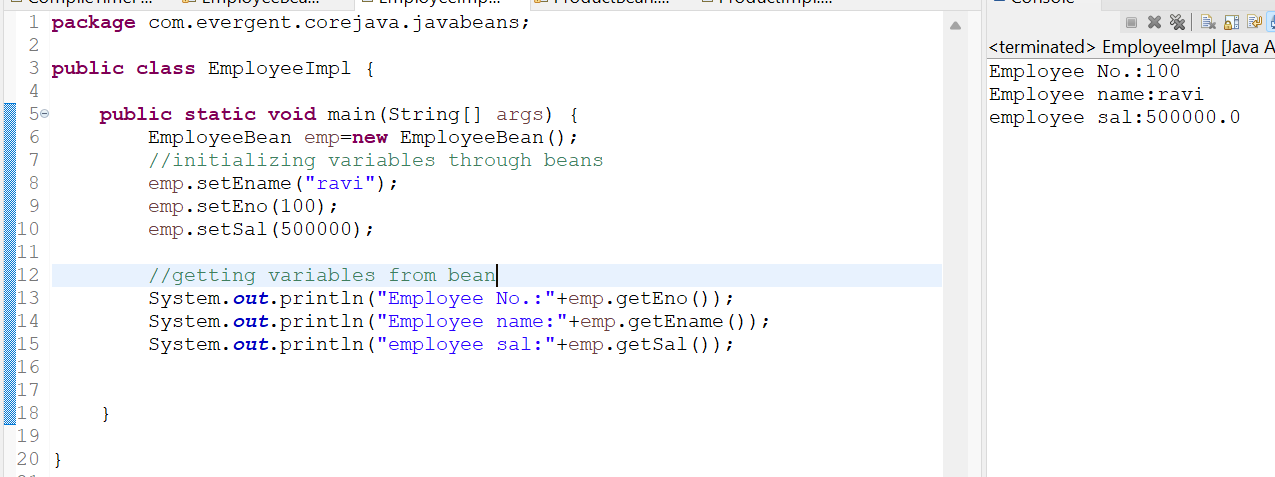
**Java BEANS:**

* Java Bean is a mechanism.
* Java Bean is lightweight.
* All attributes are private, and setter and getter are public.
* implements java.io.serializable interface.
* we can achieve tightly encapsulation through java beans.

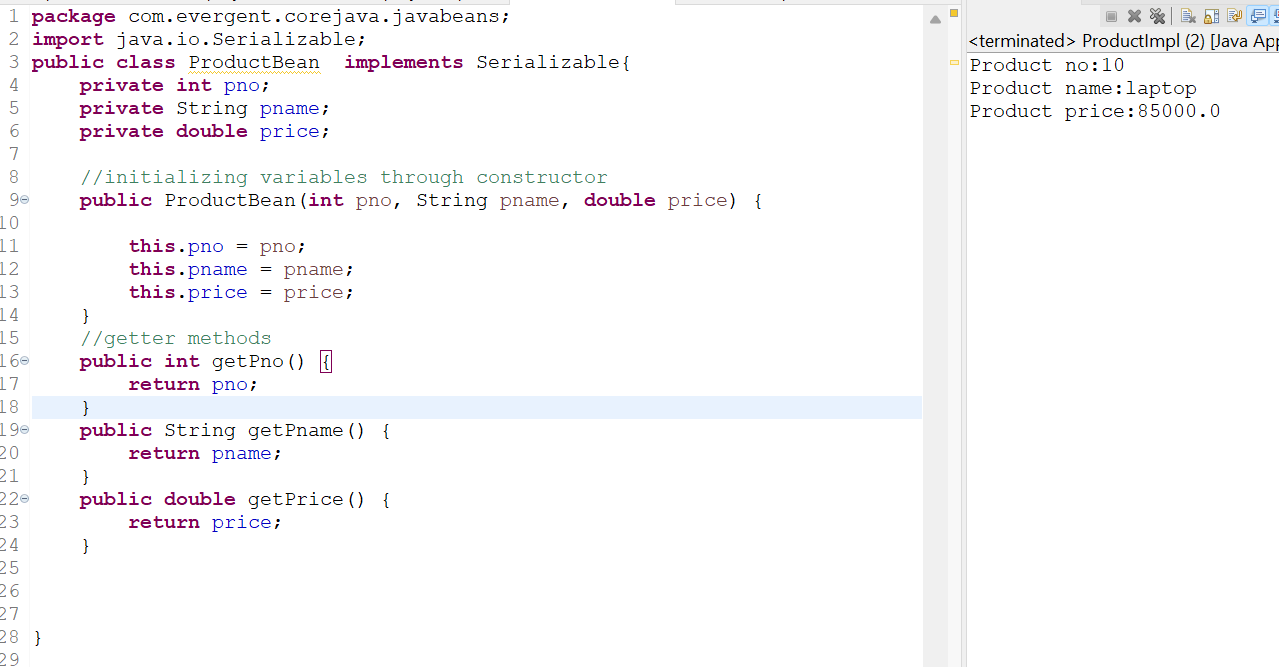
**EmployeeBean:**



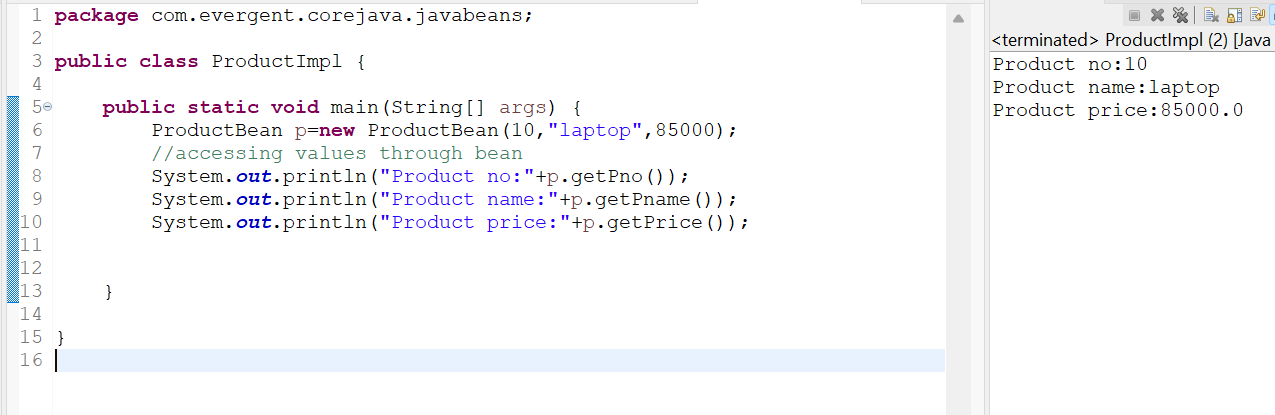
**EmployeeImpl:**



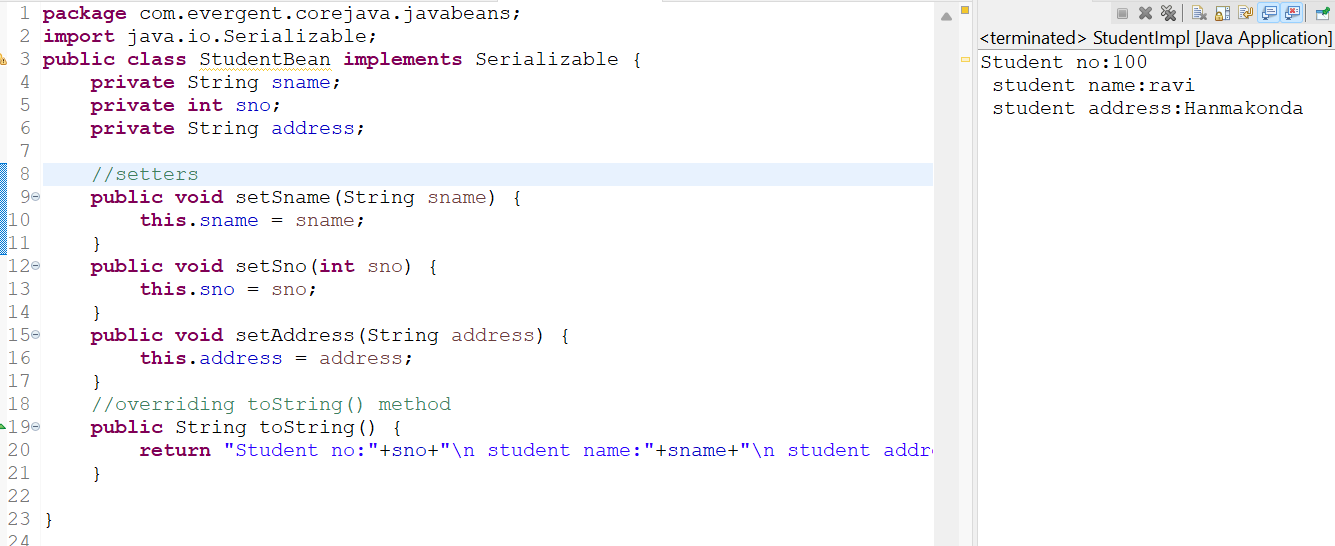
**ProductBean:**



**ProductImpl:**



**StudentBean:**



**StudentImpl:**

