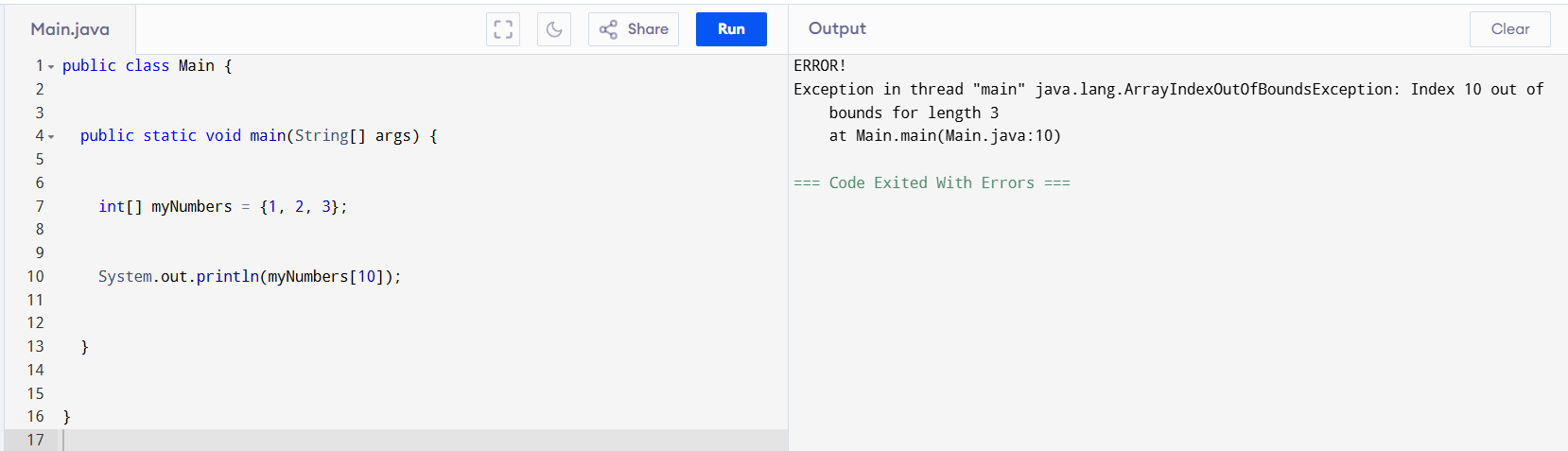
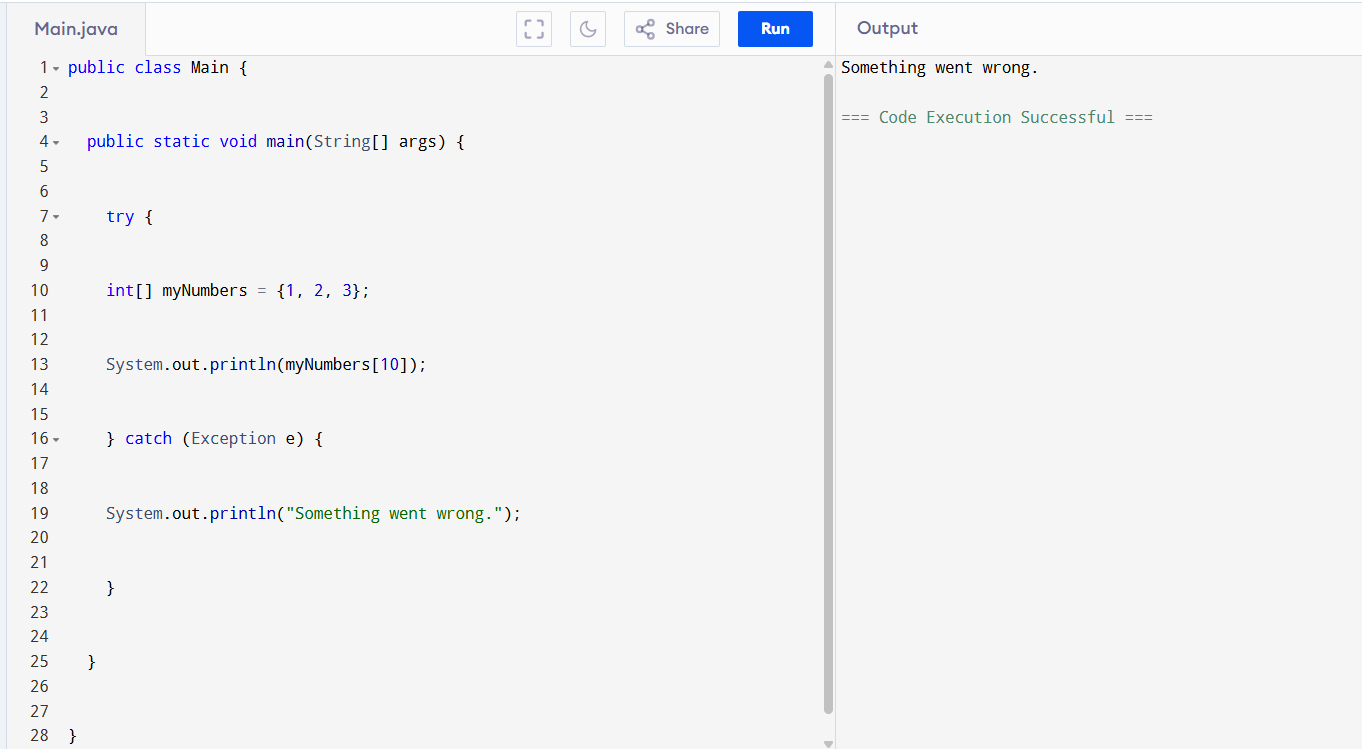
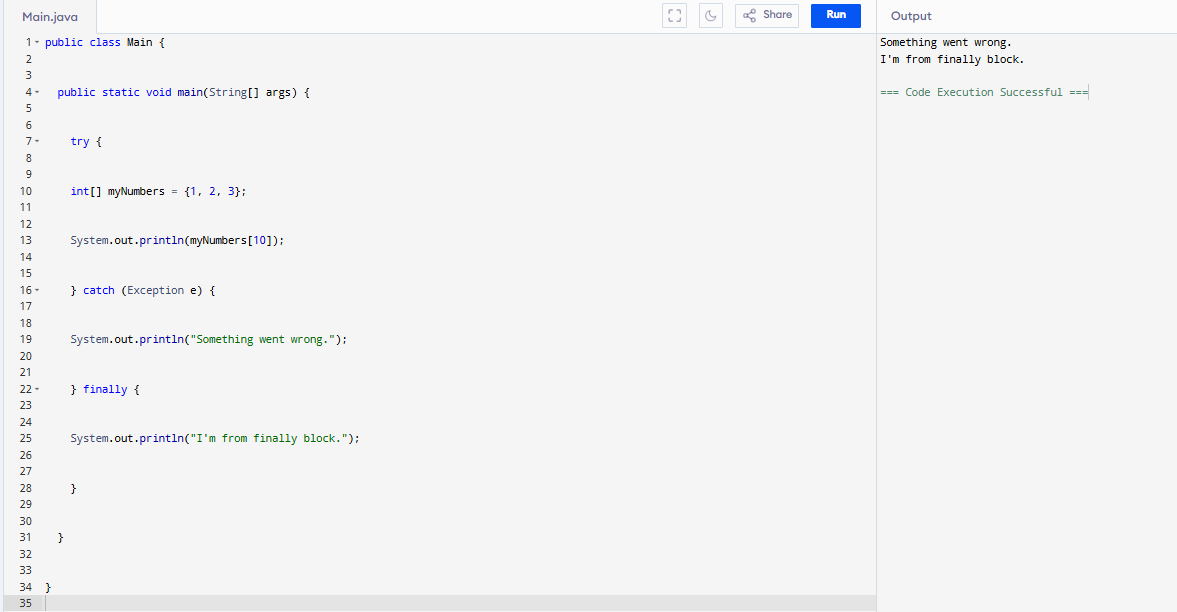
TASK 1:



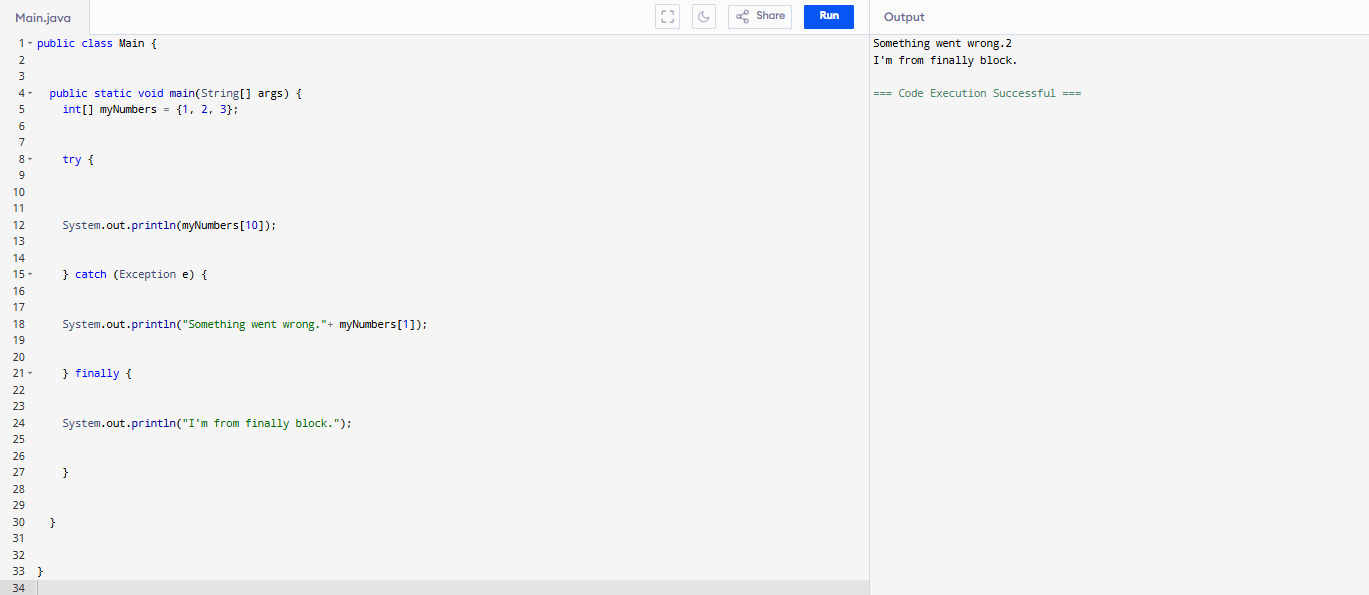
Task 2:



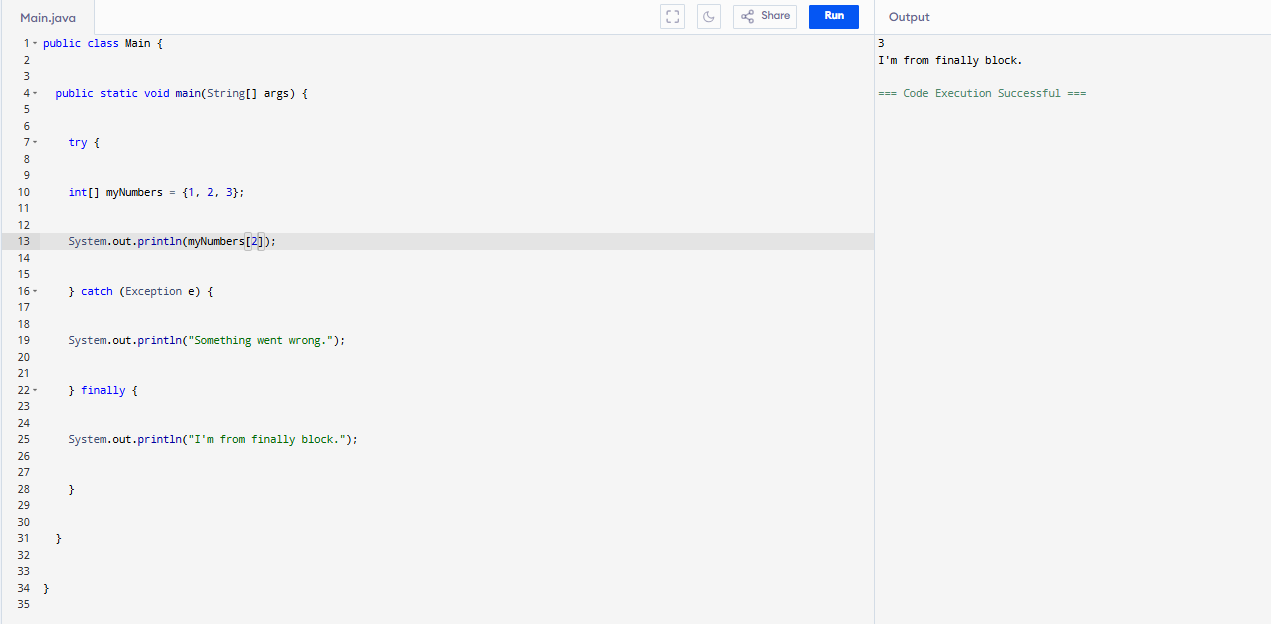
Task3 :



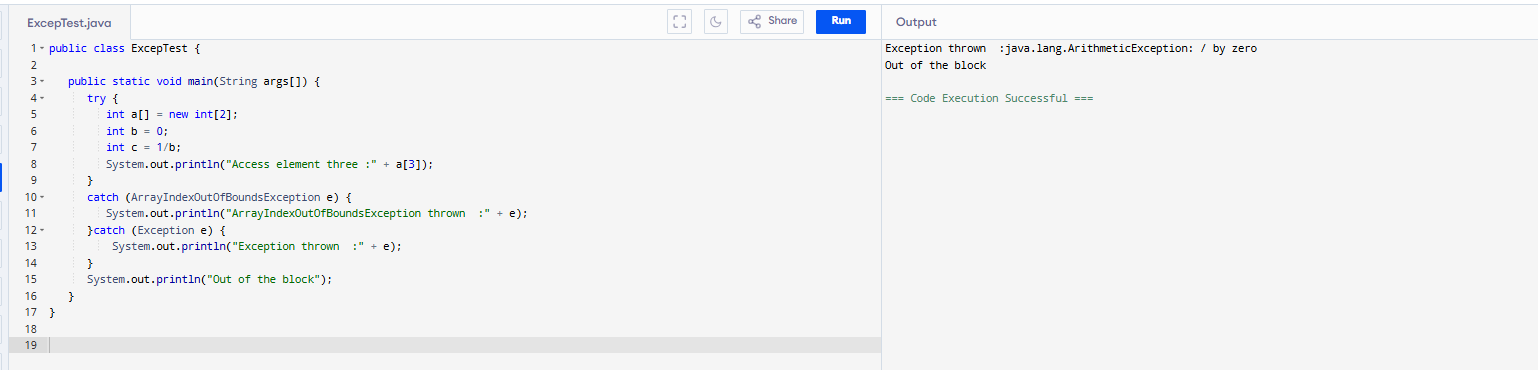
Task 3.1:



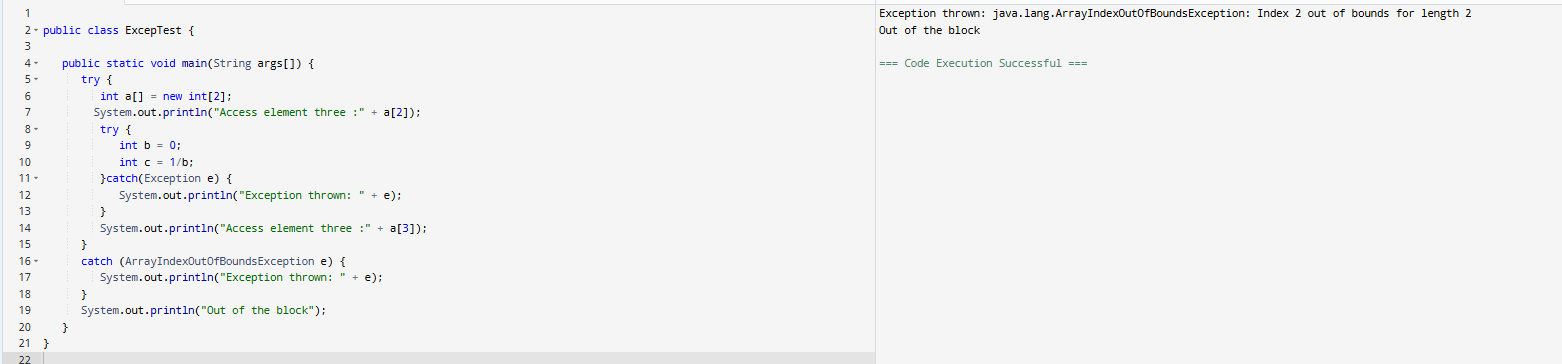
Task 4:



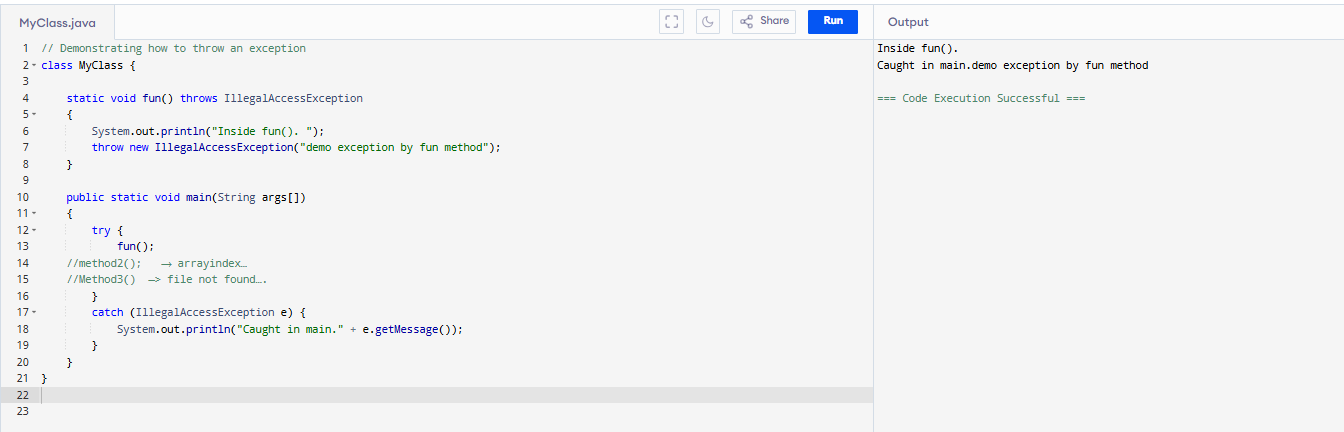
Task 5:

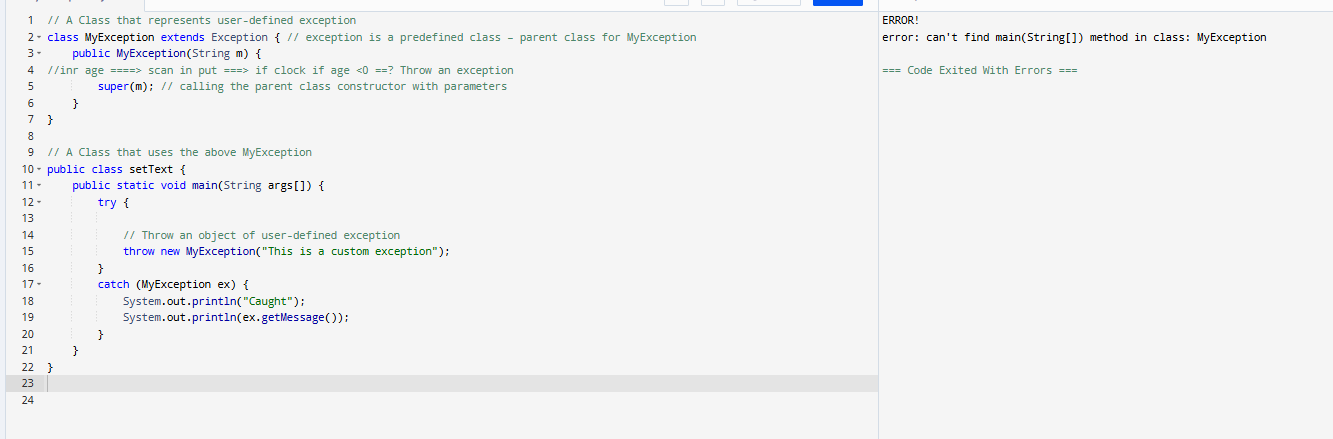


Task 6:

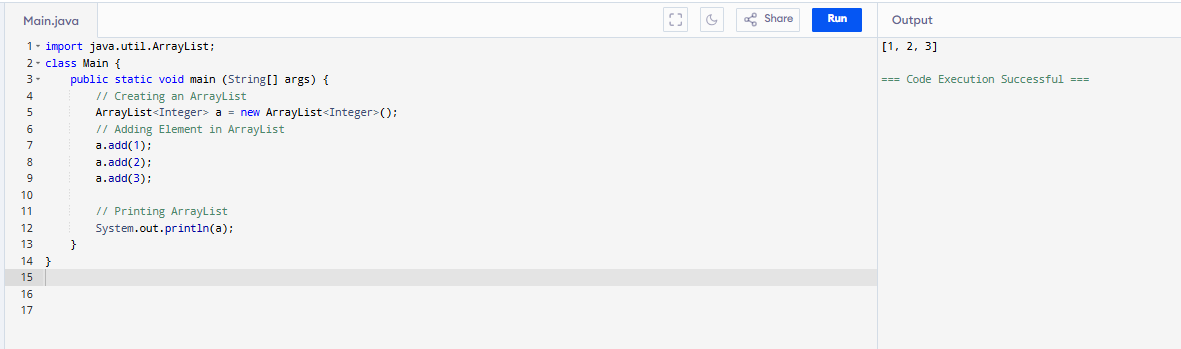


Task 7:

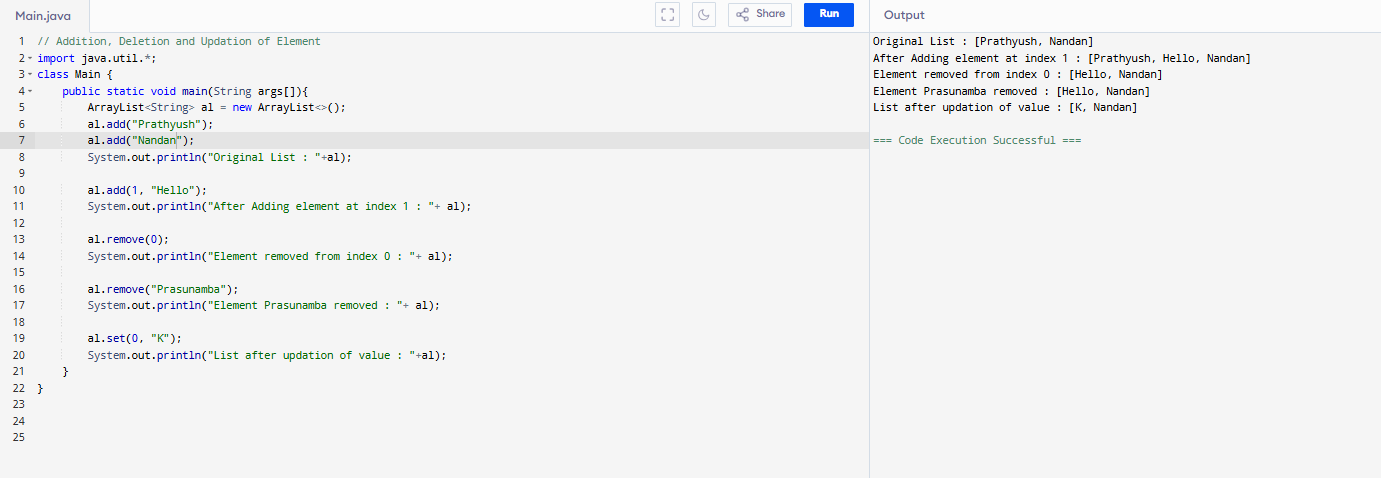
Task 8:



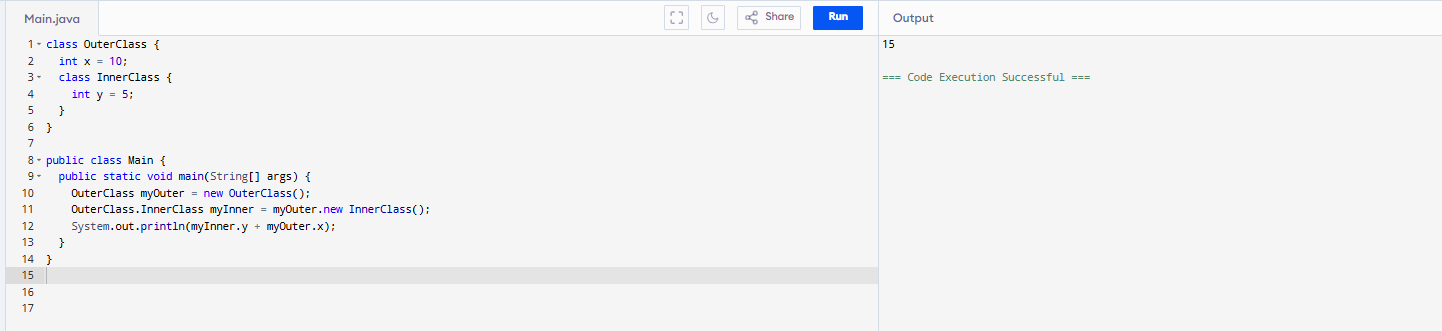
Task 9:



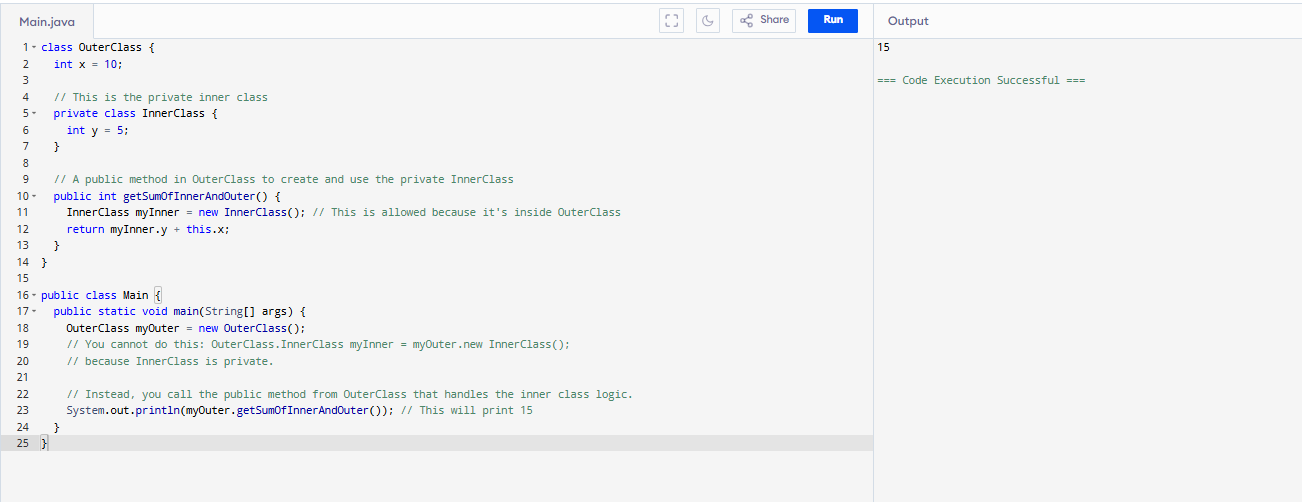
Task 10:



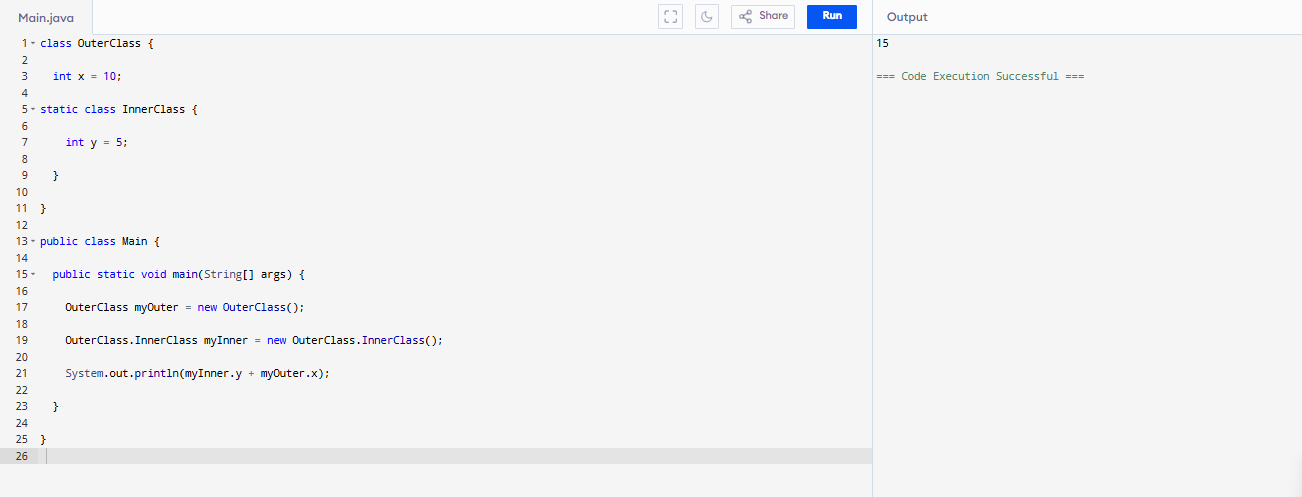
Task 11:



Task 12:



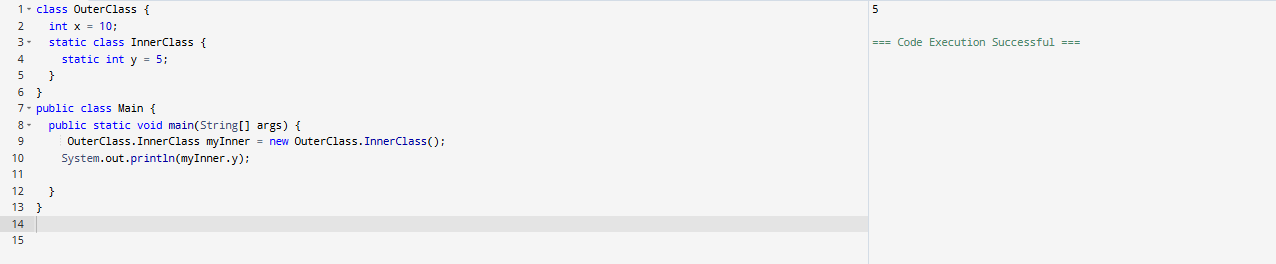
Task 13:



Task 14:



Task 15:



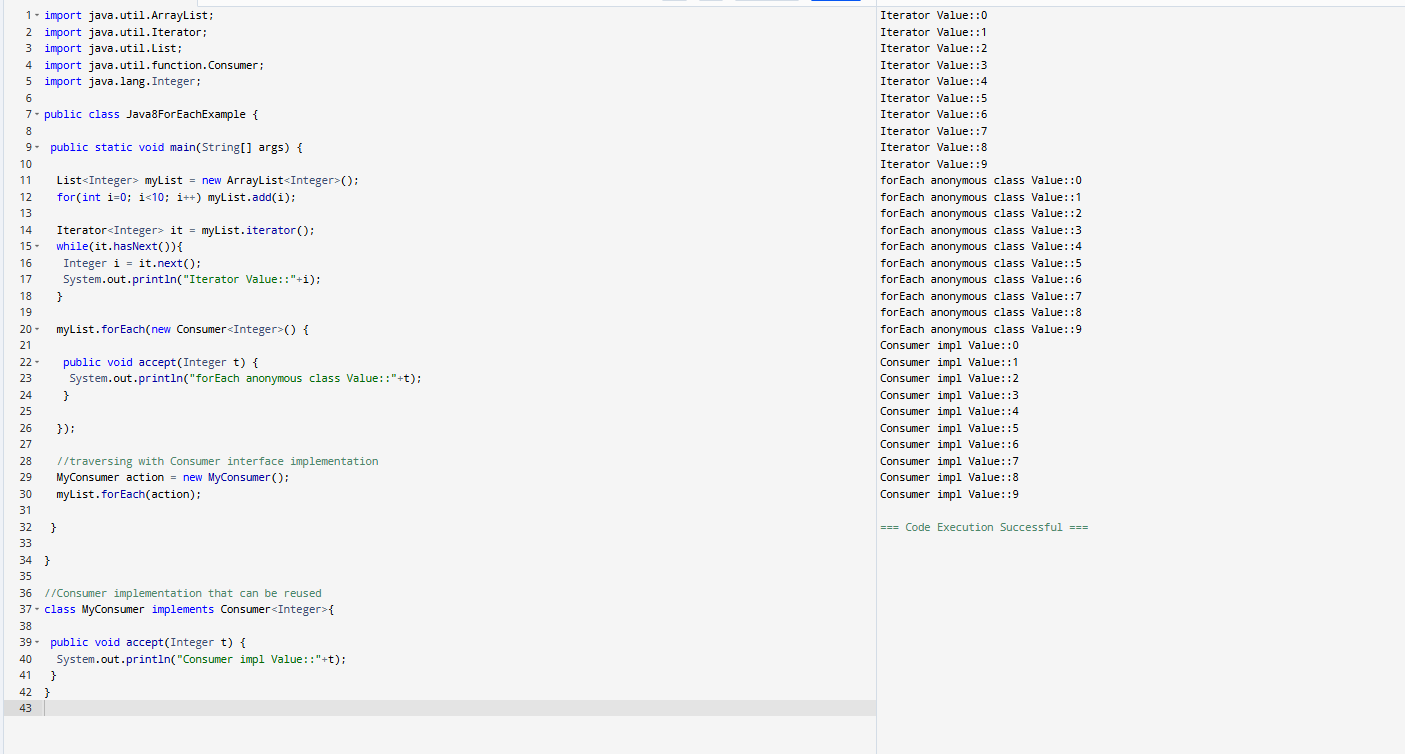
Task 16:

What are the features of java 8?

1. **Lambda Expressions:** This is arguably the most significant feature. Lambda expressions provide a concise way to represent anonymous functions, allowing you to treat functionality as a method argument or to define anonymous methods more cleanly. They enable functional programming in Java.
2. **Functional Interfaces:** An interface with exactly one abstract method is called a functional interface (also known as a Single Abstract Method or SAM interface). Lambda expressions are used to provide the implementation for these interfaces. The @FunctionalInterface annotation is used to mark an interface as a functional interface, helping to avoid accidental addition of abstract methods.
3. **Stream API:** The Stream API (in the java.util.stream package) provides a powerful and expressive way to process sequences of elements (like collections) in a functional style. It allows for operations like filtering, mapping, and reducing data, and supports both sequential and parallel execution, making it efficient for large datasets.
4. **Default and Static Methods in Interfaces:** Prior to Java 8, interfaces could only have abstract methods. Java 8 introduced the ability to add default methods (with implementations) and static methods to interfaces.
   * **Default methods** provide backward compatibility, allowing new methods to be added to existing interfaces without breaking existing implementations.
   * **Static methods** in interfaces are utility methods associated with the interface itself, not with any specific implementation.
5. **Date and Time API (java.time package):** Java 8 introduced a completely new and improved Date and Time API, addressing the shortcomings of the old java.util.Date and java.util.Calendar classes. The new API is immutable, thread-safe, and provides clear distinctions between dates, times, and date-times, as well as better handling of time zones. Key classes include LocalDate, LocalTime, LocalDateTime, ZonedDateTime, and DateTimeFormatter.
6. **Optional Class:** The java.util.Optional<T> class is a container object that may or may not contain a non-null value. It helps in dealing with null references more gracefully and reduces the risk of NullPointerExceptions by forcing developers to explicitly handle the absence of a value.
7. **Method References:** Method references provide a concise syntax for referring to methods or constructors without invoking them. They can be used in conjunction with lambda expressions for even more compact and readable code when a lambda expression just calls an existing method.
8. **forEach() Method in Iterable Interface:** The Iterable interface now has a forEach() default method, which allows for a cleaner way to iterate over collections using lambda expressions, reducing boilerplate code compared to traditional for-each loops.
9. **Parallel Array Sorting:** Java 8 introduced Arrays.parallelSort() methods, which leverage the Fork/Join framework to sort arrays in parallel, offering performance improvements for large arrays on multi-core systems.
10. **Nashorn JavaScript Engine:** This is a new JavaScript engine that allows developers to embed JavaScript code within Java applications and provides improved performance and compliance with ECMAScript standards.
11. **Base64 Encoding and Decoding:** Java 8 added built-in support for Base64 encoding and decoding, simplifying common data manipulation tasks.
12. **Concurrency Enhancements:** While the core concurrency model remained similar, Java 8 brought improvements and new classes (like CompletableFuture) for more efficient parallel processing.

These features collectively made Java 8 a very influential release, significantly enhancing the language's capabilities for modern software development, especially in areas like functional programming and concurrency.

Task 18:



Task 19:

<https://www.digitalocean.com/community/tutorials/java-8-features-with-examples>

Plz go through this link for Java 8 features..

Task 20:

import java.util.\*;

public class CollectionsDemo {

   public static void main(String[] args) {

      // ArrayList

      List a1 = new ArrayList();

      a1.add("Zara");

      a1.add("Mahnaz");

      a1.add("Ayan");

      System.out.println(" ArrayList Elements");

      System.out.print("\t" + a1);

      // LinkedList

      List l1 = new LinkedList();

      l1.add("Zara");

      l1.add("Mahnaz");

      l1.add("Ayan");

      System.out.println();

      System.out.println(" LinkedList Elements");

      System.out.print("\t" + l1);

      // HashSet

      Set s1 = new HashSet();

      s1.add("Zara");

      s1.add("Mahnaz");

      s1.add("Ayan");

      System.out.println();

      System.out.println(" Set Elements");

      System.out.print("\t" + s1);

      // HashMap

      Map m1 = new HashMap();

      m1.put("Zara", "8");

      m1.put("Mahnaz", "31");

      m1.put("Ayan", "12");

      m1.put("Daisy", "14");

      System.out.println();

      System.out.println(" Map Elements");

      System.out.print("\t" + m1);

   }

}

