**Date:** *19-Jun-2025*

**Name:** *Vivek Samant*

**Empid:** *109085619*

**alias:** *samantvs*

**Task1: What are streams?**

Streams are a feature introduced since JAVA8*.* These are not data structure but a way to work with underlying DS such as array, arraylist, stacks. These facilitate easy operation like sorting, filtering etc. on underlying DS without changing the original DS. These work with predefined method like forEach(), filter() in functional way similar to lambda expression getting rid of writing lengthy code. These allow 2 types of operations: intermediate and terminal.

**Task2: Boiler plate code**

This is a basic/ fundamental piece of code that the program has to reuse at multiple places. This is like a basic structure such as for/ while/ do-while loop

Lack of parallelism: Normal loops are single threaded by default. Streams provide a way to parallelize like parallelStream() – uses multiple parts simultaneously. Using this no need of maintaining multiple threads, java does it on its own.

Lack of composition: Without streams, one needs to painstakingly write code sequentially, like filter()-> map()-> sort() which is disjointed and each is a separate step and error prone. With stream, operations are chained together: list.stream().filter(s->s.startWith(‘A’)).map(String;;toUpperCase).sorted().toList(). (like a pipeline, intermediate and terminal operation chained together)

**Task3: List down intermediate and terminal operation in streams:**

Intermediate: filter()-🡪 filtering based on a condition

Distinct(), sorted(), peek(), limit(), skip()

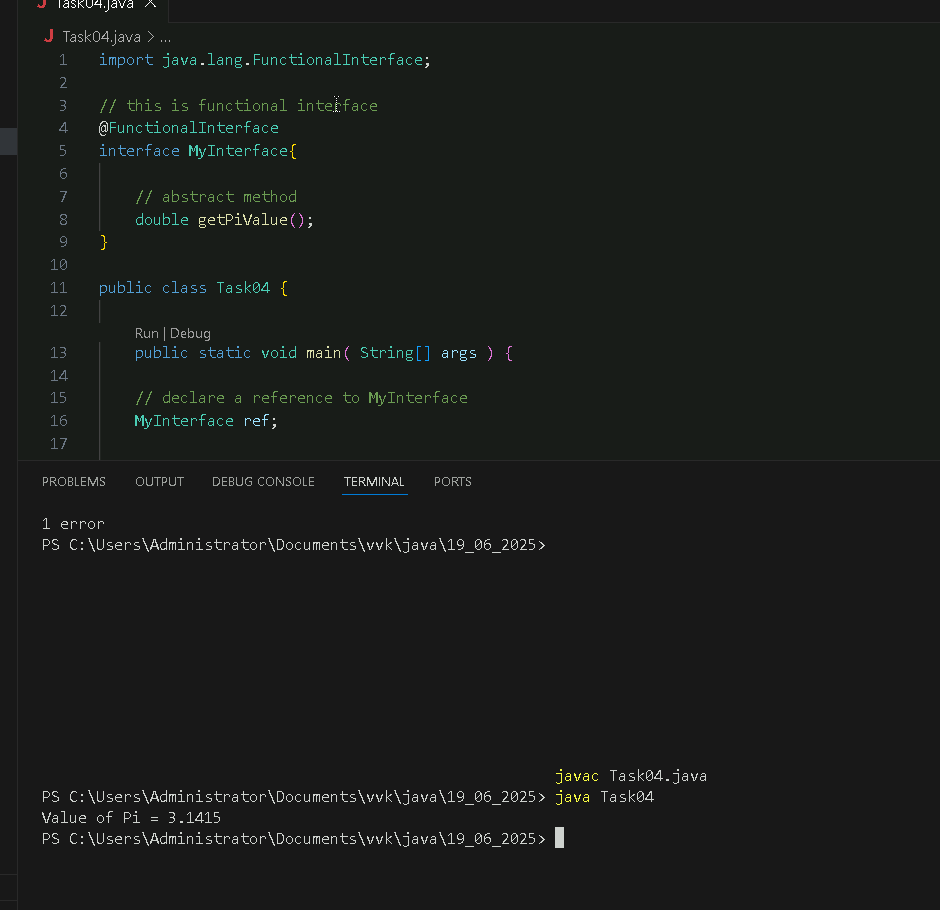
These are classified as intermediate because they perform operation on the underlying element in collection while keeping stream going

Terminal: forEach(), collect(), reduce(), count(), min(), max(), toarray()etc.

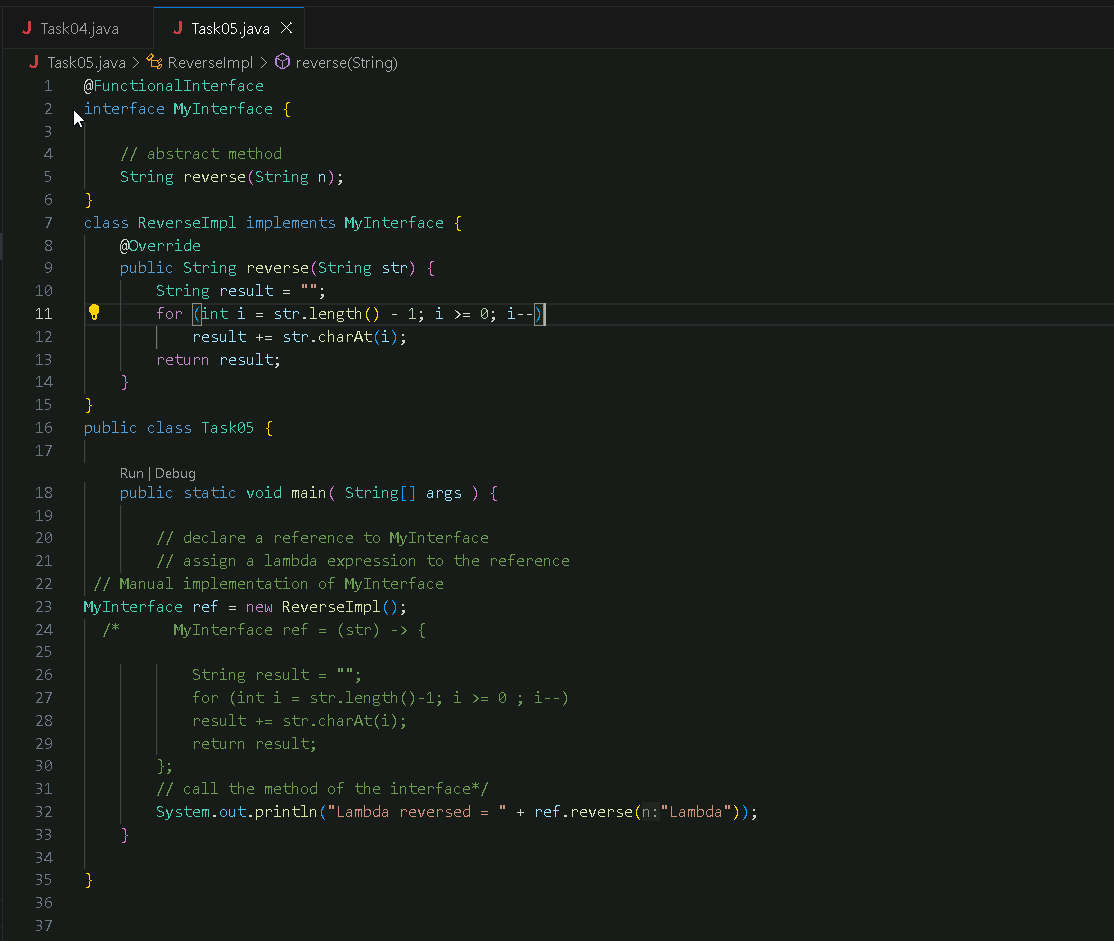
These give final o/p ending the stream.

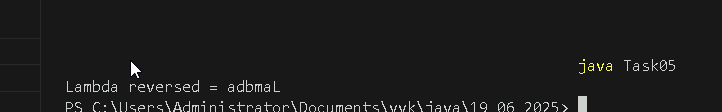
After these terminal operation, stream is consumed and no further operation possible.

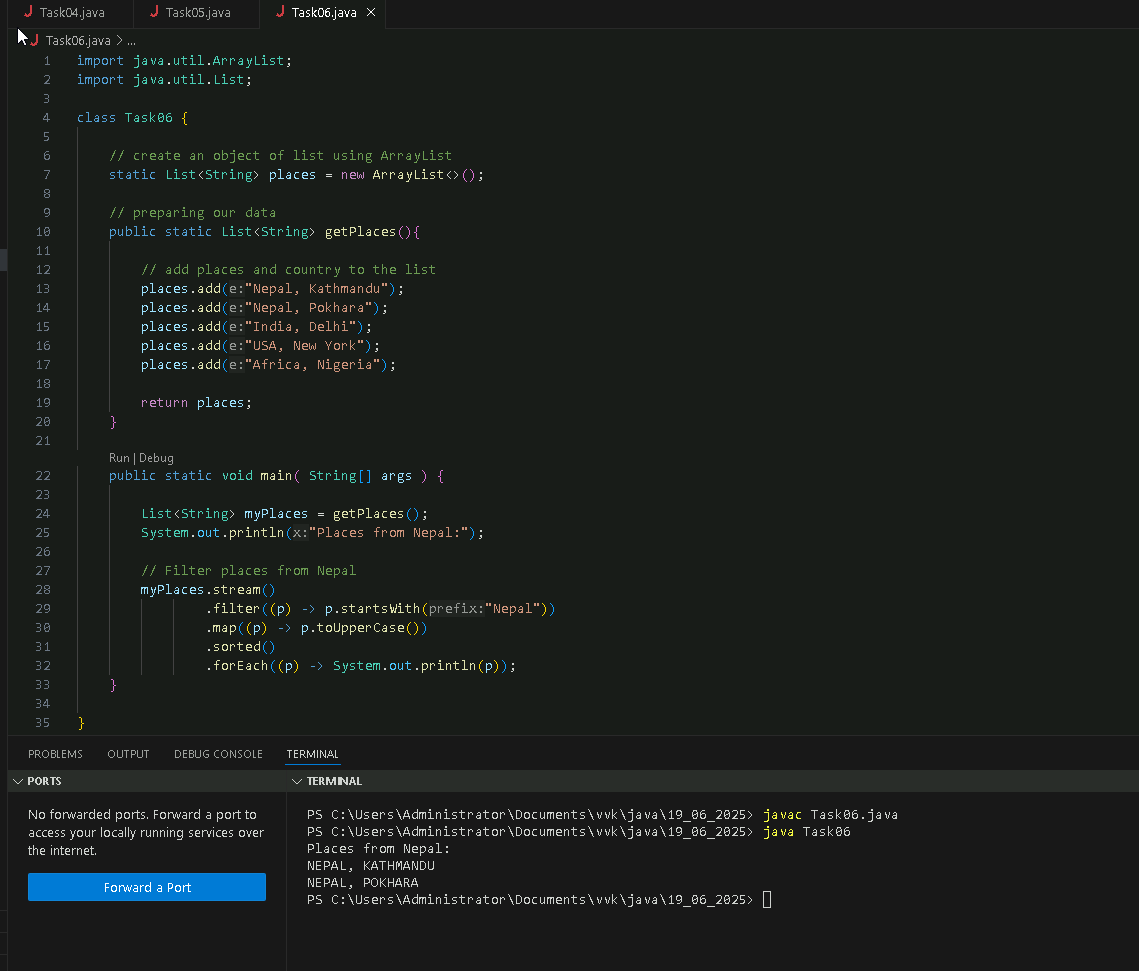
**Task 04:**



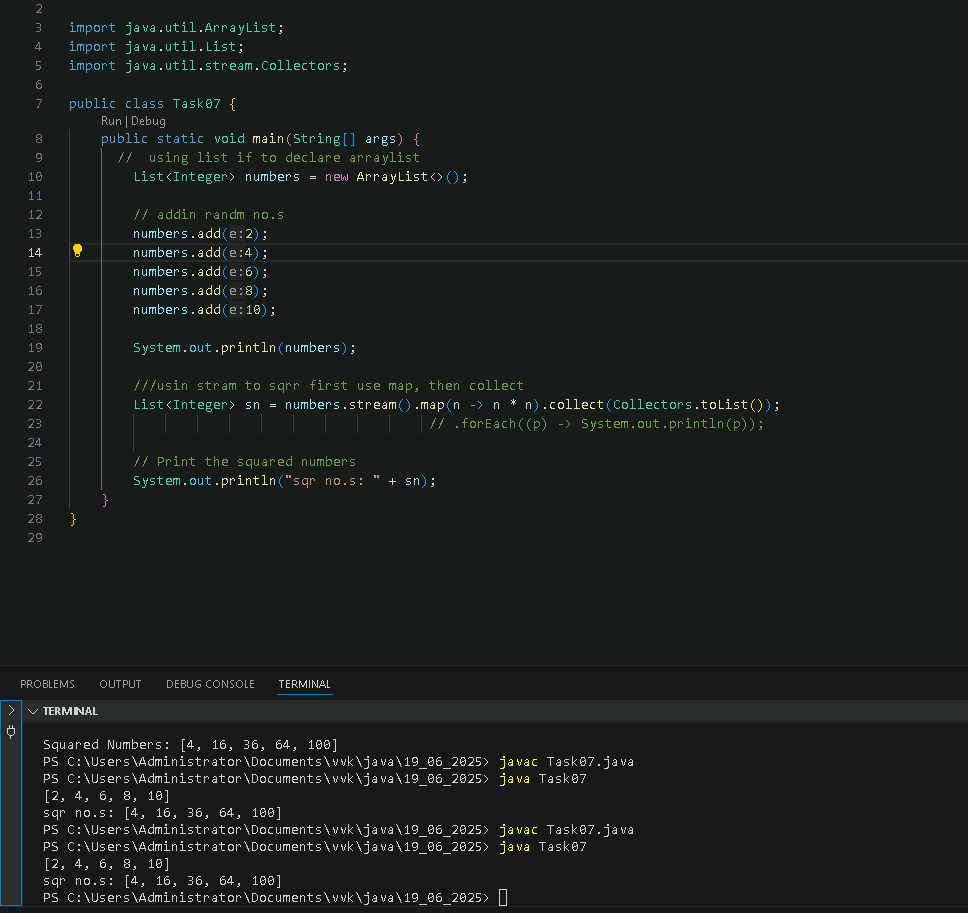
**Task 05:**





**Task06:**

**Task 07:**



**Task 8: Map function in stream():**

This map() is an intermediate operation that transform each element in the stream as per a given function.

Eg:

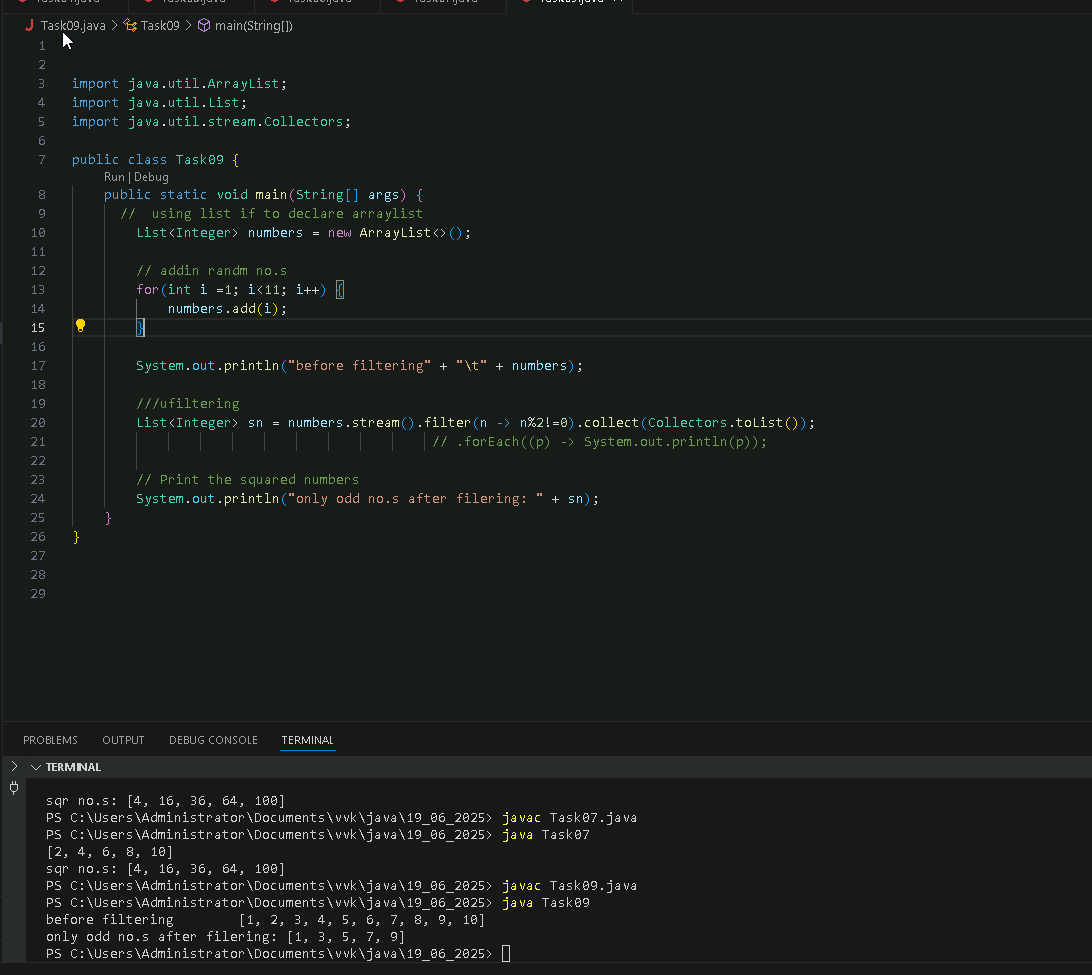
Numbers.stream().Map(n-> n\*1)

It can also be used multiple times

Numbers.stream().map(n->n\*1). Map(n->n\*2)

names.stream().map(String::toUpperCase).forEach(System.out::println); // JOHN, ALICE, BOB

**Task 9:**



**Task 10:**

filter() is an intermediate operation that filters elements passed to it during stream pipeline as per condition and allows the ones which pass the condition (similar to true or false)

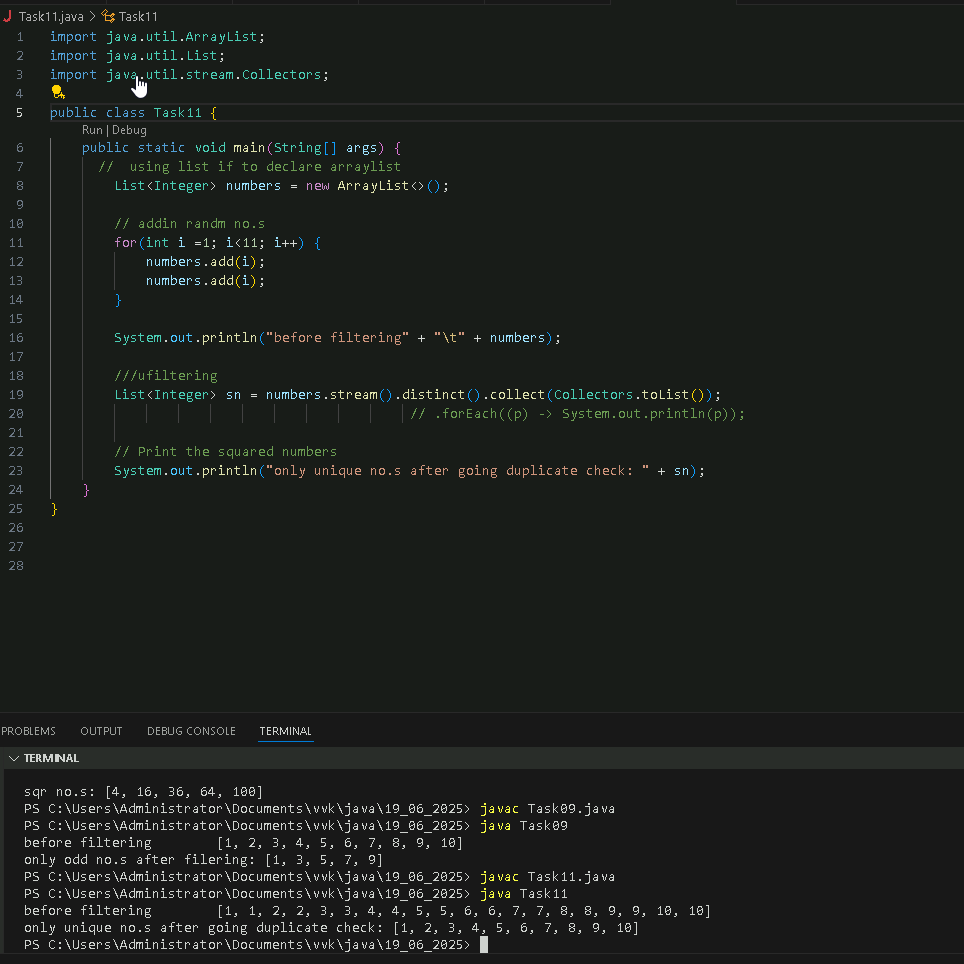
eg:

names.stream()

.filter(s -> s.startsWith("v")) // keep names starting with v

.forEach(System.out::println);

**Task 11:**



**Task12:**

What do you understand by distinct()?

This is an intermediate operation that removes duplicates and returns only unique elements in the stream

Sort: sorts elements in natural order or by custom comparator given by the user.

Limit: restricts elements to first n elements where n is the number inputted (similar to limit in mysql).

Skip: skips elements as many as given by the user (n)

All thee above intermediated

The given below are terminal:

Foreach : iterates through each element (similar to enhanced for loop usual). After iterating usually we can print using (sout::println)

Collect It gathers elements into a collection like arraylist, map, hashset,

List<Integer> list = Arrays.asList(1,2,3).stream().collect(Collectors.toList()); // o/p: [1,2,3]

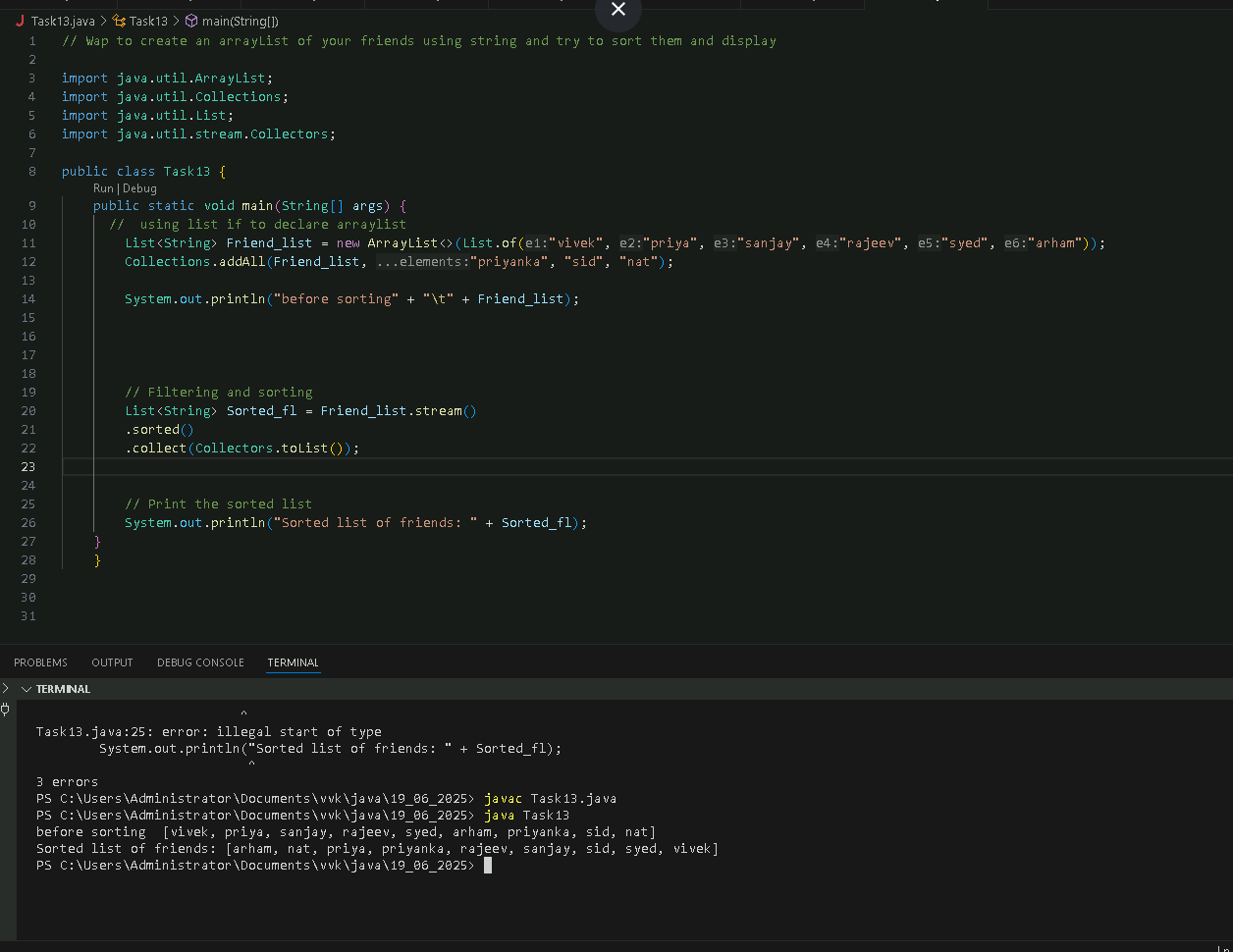
Set<Integer> set = Arrays.asList(1,1,2,3).stream().collect(Collectors.toSet()); // o/p; [1,2,3]

Reduce Reduces stream to single value using accumulator (final aggregation)

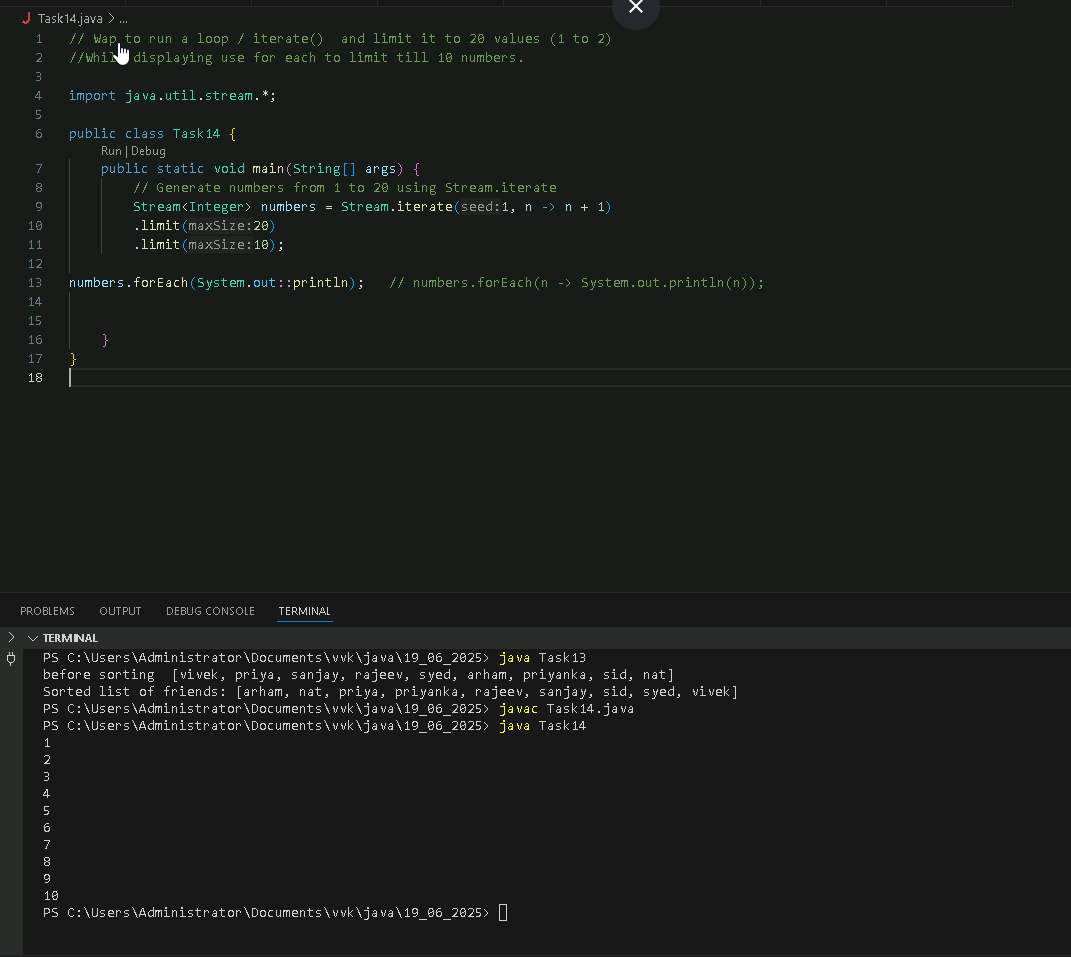
int sum = Arrays.asList(1,2,3).stream().reduce(0, (a,b) -> a + b); // 6

String concat = Arrays.asList("a","b","c").stream().reduce("", String::concat); // "abc"

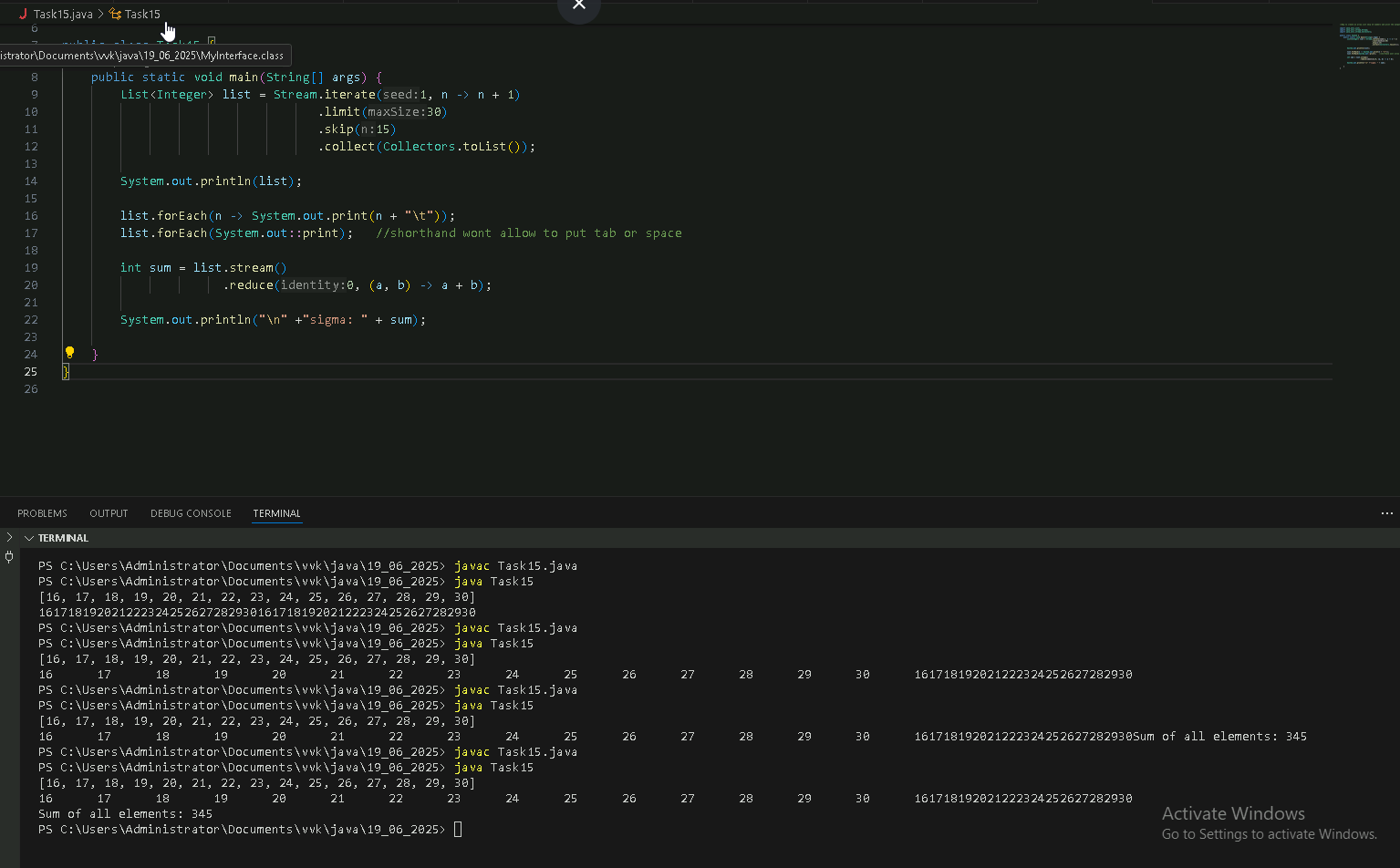
Task13:



**Task14:**



**Task 15:**



**Task 16:**

Explain limit and skip methods..

Limit: gives the first n elements (n input by user)

Skip: skips the first n elements and then prints the rest (n given by user)

**Task 18:**

**Advantage of streams:**

**They work with variety of collections such as arrays, lists etc.**

**They are workable with Optional which is useful to handle null values safely**

**Streams are reusable, promoting code reusability and modularity**

**Reduced boiler plate code, by using functional (lambda expressions) to directly achieve and get rid of redundant code. Lambda also allow functional programming style and allow method referednce (e.g.: forEach(System.out::print)**

**Allow parallel processing and no need to manually make different threads and java manages on own.**

**Task 19:**

What are the debugging tools in Java.. list down a few

* **IDEs (intellij, vscode, netbeans, eclipse)**
* **Unit testing (Junit, Mockito)**

** Monitoring Tools**

* **JMX (Java Management Extensions)**
* **Java Mission Control**
* **Spring Boot Actuator**

** Logging Tools**

* **Log4j**
* **SLF4J**
* **Java Logging API (java.util.logging)**
* **Logback**

** JDB (Java Debugger)**

* **Command-line debugger**
* **Comes with JDK**
* **Good for remote debugging**

**Task 20:**

**Types of errors:**

1. **Compile time**
2. **Run time**

**Compile time: missing braces, grammatical error, checked exception not checked using try-catch, method doesn’t exist, accesss modifier violation,**

**Run time: Array index out of bound, null pointer, IO exception, stack overflow, division by 0, parsing error, can’t cast integer to a string.**

**Task 21:**

