**Q1) Introduction to Object Oriented Concepts, comparison of Java with other object oriented programming languages. Introduction to JDK, JRE, JVM.**

Git hub repository link : - https://github.com/ZeroMeet/First

i) Introduction to Object Oriented Concepts

➢ OOps in java is to improve code readability and reusability by defining a Java program efficiently. The main principles of object-oriented programming are **abstraction, encapsulation, inheritance, and polymorphism**. These concepts aim to implement real-world entities in programs.

### *ii) comparison of Java with c++:-*

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Java** | **C++** |
| Influenced By: | Java was Influenced by Ada 83, Pascal, C++, C#, etc. languages. | C++ was Influenced by  Influenced by Ada, ALGOL 68, C, ML, Simula, Smalltalk, etc. languages. |
| Influenced to: | Java was influenced to develop BeanShell, C#, Clojure, Groovy, Hack, J#, Kotlin, PHP, Python, Scala, etc. languages. | C++ was influenced to develop C99, Java, JS++, Lua, Perl, PHP, Python, Rust, Seed7, etc. languages. |
| Platform Dependency | Platform independent, Java bytecode works on any operating system. | Platform dependent, should be compiled for different platforms. |
| Portability | It can run in any OS hence it is portable. | C++ is platform-dependent. Hence it is not portable. |
| Compilation | Java is both Compiled and Interpreted Language. | C++ is a Compiled Language. |
| Memory Management | Memory Management is System Controlled. | Memory Management in C++ is Manual. |
| Virtual Keyword | It doesn’t have Virtual  Keyword. | It has Virtual keywords. |
| Multiple Inheritance | IIt supports only single inheritance. Multiple inheritances are achieved partially using interfaces. | It supports both single and multiple Inheritance. |
| Overloading | It supports only method overloading and doesn’t allow operator overloading. | It supports both method and operator overloading. |
| Pointers | It has limited support for pointers. | It strongly supports pointers. |
| Libraries | It doesn’t support direct native library calls but only Java Native Interfaces. | It supports direct system library calls, making it suitable for system-level programming. |
| Libraries | Libraries have a wide range  of classes for various highlevel services. | C++ libraries have comparatively low-level functionalities. |
| Documentation Comment | It supports documentation  comments (e.g., /\*\*.. \*/) for source code. | It doesn’t support documentation comments for source code. |
| Thread Support | Java provides built-in support for multithreading. | C++ doesn’t have built-in support for threads, depends on third-party threading libraries. |
| Type | Java is only an objectoriented programming language. | C++ is both a procedural and an object-oriented programming language. |

*iii) Introduction to JDK, JRE, JVM:-*

*JDK:-*

* JDK contains everything that JRE has alongside improvement devices for creating, troubleshooting, and observing Java applications. You require JDK when you have to build Java applications.

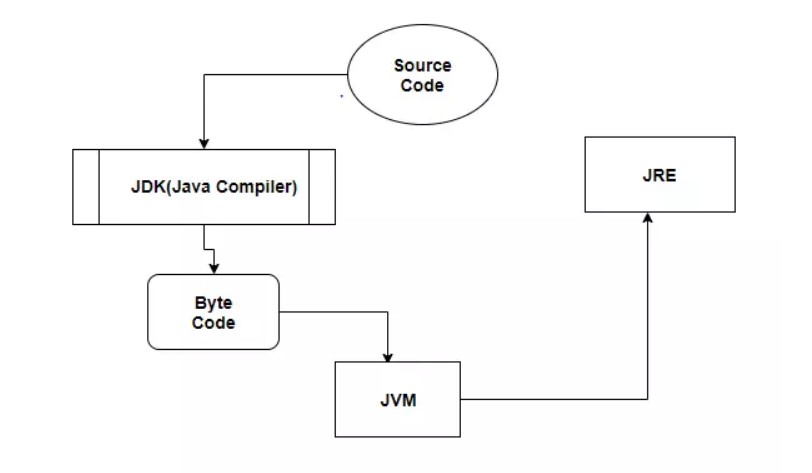
* The Java Development Kit (JDK) provides an environment utilized for creating Java applications and applets. It incorporates the Java

Runtime Environment (JRE), a loader, a compiler (javac), an

archiver (jar), a documentation generator (Javadoc) and different apparatuses required in Java improvement.

# JRE:-

* The Java Runtime Environment (JRE) is a product bundle which packages the libraries (containers) and the Java Virtual Machine, and different parts to run applications written in Java. JVM is only a piece of JRE dispersions.
* To execute any Java application, you require JRE introduced in the machine. It’s base necessity to run Java applications on any device.
* JREs come as a feature of JDKs, or you can download them independently. JREs are dependent on the kind of machine (OS and engineering). You should choose the JRE package to import and introduce.



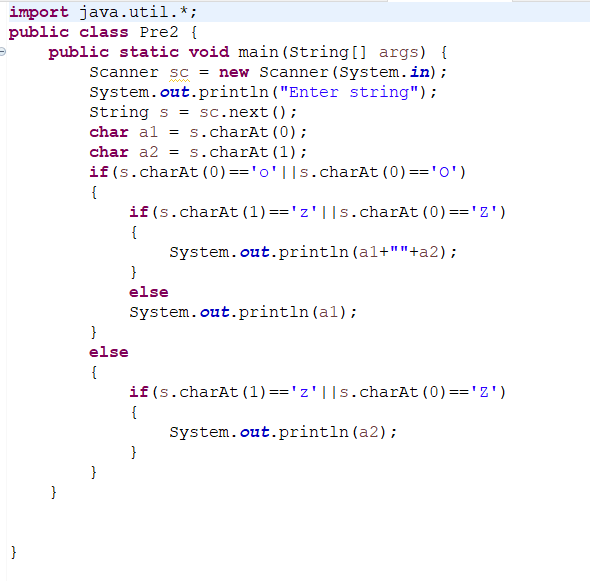
# JVM:-

* It is also known as Java Virtual Machine. It acts as a run-time motor to run Java applications. JVM is the one that calls the entry-point “main” method. JVM is a piece of JRE(Java Runtime Environment).
* An engineer can create Java code on one framework and can anticipate that it should keep running on some other Java empowered framework with no modification. It is all conceivable due to JVM.
* When we assemble a .java document, the Java compiler produces the .class files with similar names present in the .java record. This .class document goes into different advances when we run it. These means together depict the entire JVM.
* JVM conveys the ideal execution for Java applications utilizing many propelled strategies, fusing a best in class memory display/model, garbage collector, and versatile, adaptive optimizer.
* The server VM has been uncommonly tuned to amplify top working rate. It has to run for long-running server applications, which requires the quickest conceivable and active velocity over quick start-up time. Engineers can pick which framework they need by determining – customer or – server.
* The JVM is virtual because it gives a machine interface that does not rely upon the basic working framework and machine equipment design. This freedom from equipment and the working structure is a foundation of the write once-run-anyplace estimation of Java programs.

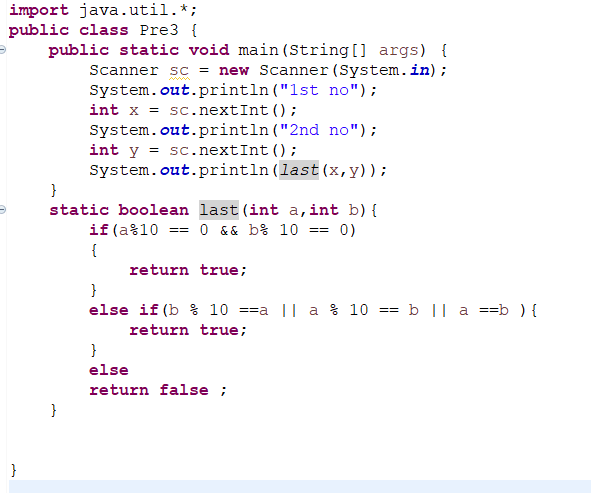
**Compare JVM, JDK, and JRE**

* JVM is the virtual engine and the one which enables byte code support.
* JRE contains JVM and all the other libraries to run Java application. It is enough to run any Java application.
* JDK is a superset which comprises of JVM, JRE, and the tools to develop Java Application.

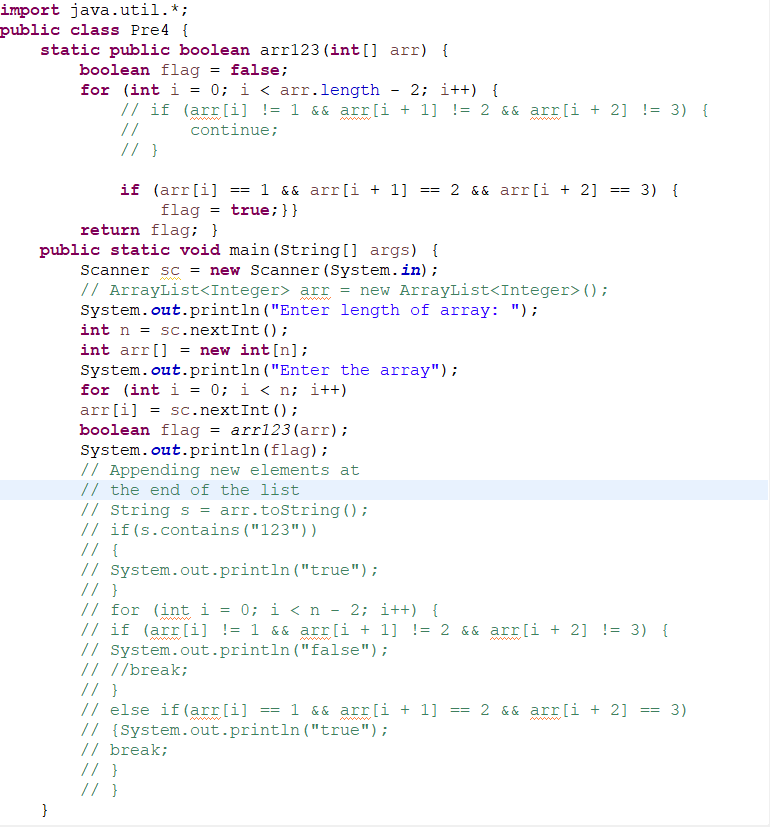
Q 2



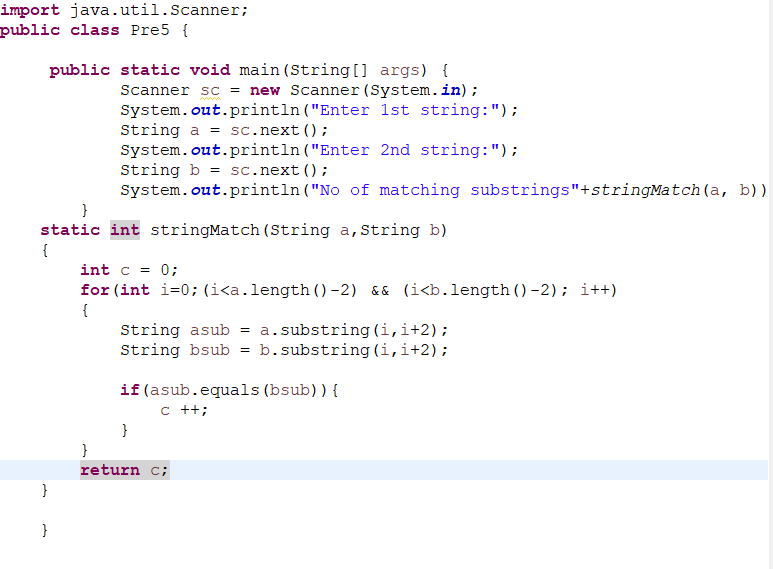
Q3



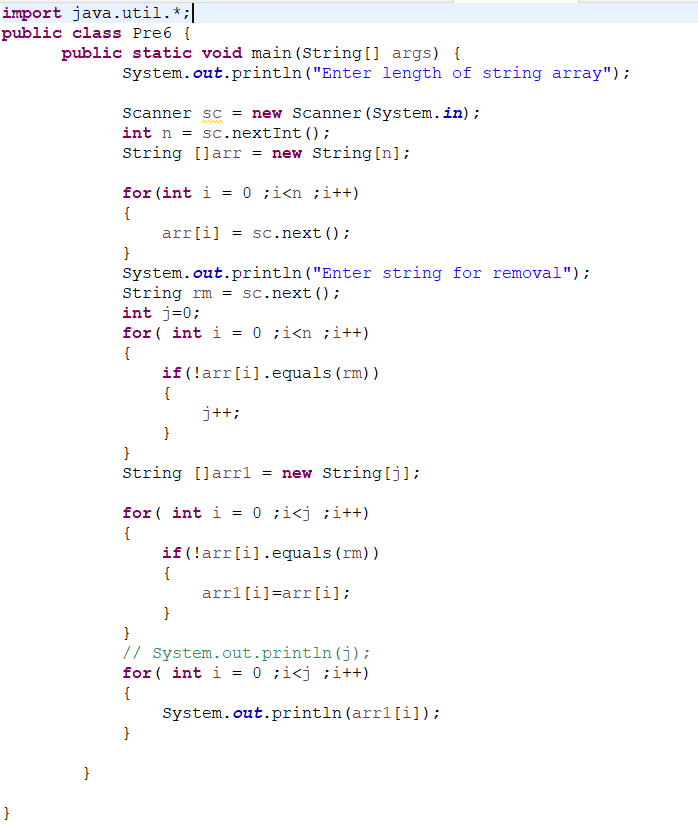
Q4



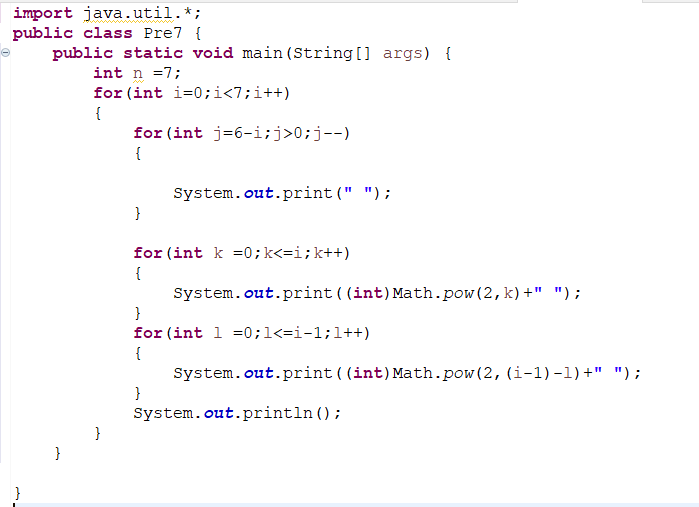
Q5



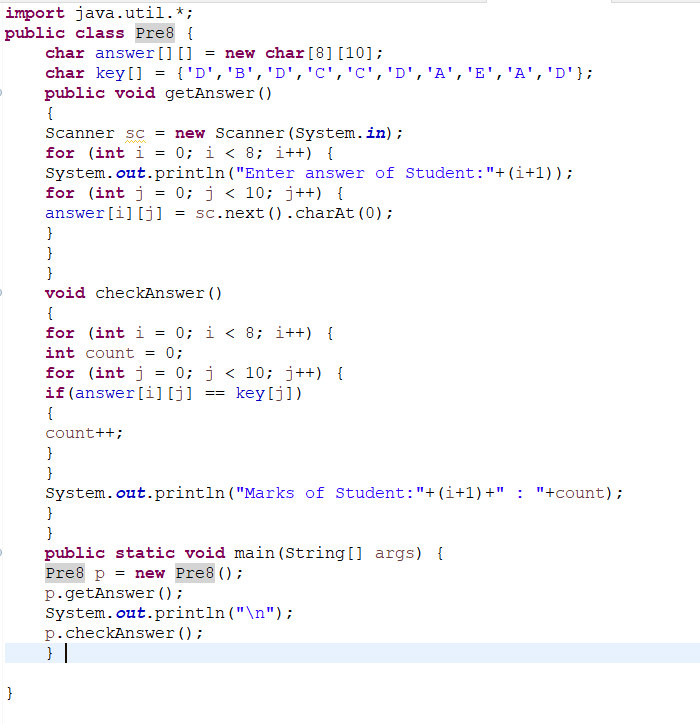
Q6

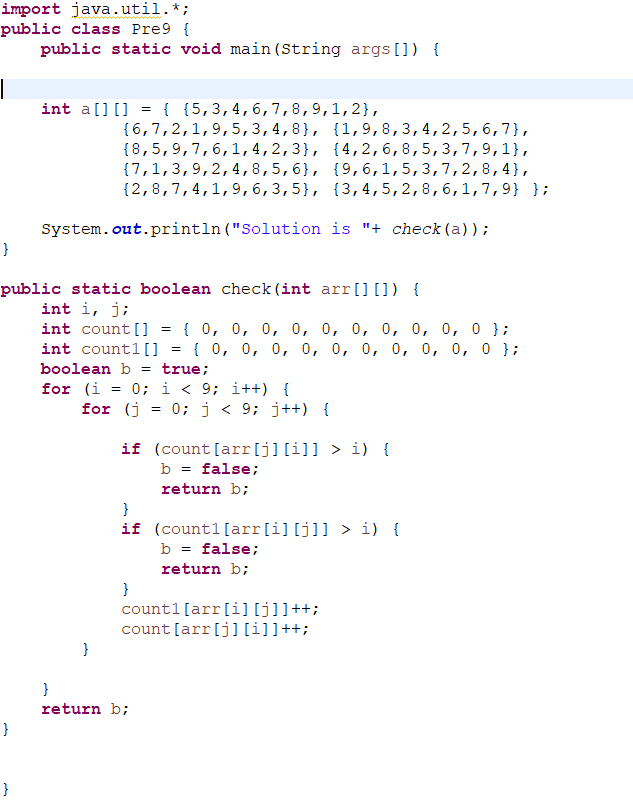


Q7



Q8

Q9



Q10

