1. Why do we need static keyword in Java? Explain with an example.
   Answer: The ‘static’ keyword in Java is used to indicate that a member belongs to a class rather than an instance of the class.

   For example, consider the following class ‘Human’:

```
public class Human {
        static int population = 0;
        String name;

        // constructor
        Human(String name) {
                population++;
                this.name = name;
        }

        public static void main(String[] args) {
                Human humanOne = new Human("Aditya");
                Human humanTwo = new Human("Rahul");

                System.out.println("Human One : Name =  " + humanOne.name + ",
                Population = " + humanOne.population);

                System.out.println("Human Two : Name =  " + humanOne.name + ",
                Population =  " + humanTwo.population);
        }
}
```

   Output:

```
        Human One : Name = Aditya, Population = 2
        Human Two : Name = Rahul, Population = 2
```

   Here the population variable is declared as ‘static so it belongs to the class ‘Human’ rather than an instance of ‘Human’. This means that the values of ‘population’ is shared all across instances of the class and can be accessed using the class name.

   As you can see, the value of ‘population’ is shared all across instances of the class but the value of ‘name’ is unique to each instance of the class.
   :
2. What is class loading and how does the Java program actually executes?
   Answer: Class loading is the process by which the Java Virtual Machine (JVM) dynamically loads class files into memory. When a Java program is executed, the JVM loads the necessary classes into memory, creates instances of objects, and executes their methods to produce the desired output.

Here's a high-level overview of how a Java program is executed:

a. Compilation: The Java source code is compiled into bytecode using the javac compiler. The resulting class files contain the bytecode that will be executed by the JVM.

b. Loading: When the Java program is executed, the JVM loads the necessary classes into memory. This is done by the class loader, which is responsible for finding the class files and loading them into memory.

c. Linking: After a class is loaded, the JVM performs three steps: verification, preparation, and resolution. Verification checks the bytecode to ensure it is valid and complies with the Java specification. Preparation involves allocating memory for the class variables and initializing their values to the default values specified in the class. Resolution involves resolving symbolic references to other classes and methods.

d. Initialization: Once a class is linked, the JVM initializes the class by executing its static initializer blocks, if any, and any other static initializations.

e. Execution: Finally, the JVM executes the methods of the objects in the program to produce the desired output. This involves creating instances of objects, calling methods, and processing the results.

3. Can we mark a local variable as static?
   Answer: No, local variables cannot be marked as ‘static’ in Java. The ‘static’ keyword is only applicable to class-level variables (i.e. fields of variables declared directly within a class but outside of any method or a constructor). Local variables, on the other hand, are declared within a method or a constructor and are only accessible within that method or constructor.

   Local variables do not have a class-level scope, so they cannot be marked as ‘static’. Attempting to do so would result in a compile-time error

4. Why is the static block executed before the main method in Java?
   Answer: In Java, the static block is executed before the main method because the static Block is executed when the class is first loaded into the JVM and before any objects of the class are created.

5. Why is a static method also called a class method?

Answer: A static method in Java is also called a class method because it is associated with the class and not with any specific instance of the class. This means that a static method can be called directly on the class, without creating an instance of the class.

6. What is the use of static blocks in Java?
   Answer: Static blocks are used for the following:
   a. Initializing static variable: Static blocks can be used to initialize static variables that are used by the class.
   b. Loading resources: Static block can be used to load resources that are required by the class, such as reading data from a file or setting up a database connection.
   c. Performing complex initialization: When the initialization of a class requires complex logic that cannot be performed in a constructor, a static block can be used to perform the initialization.

7. Difference between static and instance variables.
   Answer:
   Static variables, also known as class variables, are declared using the static keyword. There is only one copy of a static variable shared by all instances of a class. This means that if a static variable is modified by one instance of a class, the change will be reflected in all other instances. Static variables are typically used to store information that is common to all instances of a class, such as a constant value or a counter of the number of instances created.

   Instance variables, on the other hand, are created for each instance of a class. Each instance has its own copy of instance variables, and changes to the value of an instance variable in one instance will not affect the value of the same variable in another instance. Instance variables are typically used to store information that is specific to each instance of a class, such as an object's state.

8. Difference between static and non-static members.
   Answer:
   Static members, also known as class members, are declared using the static keyword. There is only one copy of a static member shared by all instances of a class. This means that if a static member is modified by one instance of a class, the change will be reflected in all other instances. Static members can be accessed without creating an instance of a class, and are typically used to store information that is common to all instances of a class, such as constant values or counters.

   Non-static members, also known as instance members, are created for each instance of a class. Each instance has its own copy of instance members, and changes to the value of an instance member in one instance will not affect the value of the same member in another instance. Non-static members can only be accessed through an instance of a

class, and are typically used to store information that is specific to each instance of a class, such as an object's state.