1. **Why do you need static keyword in Java Explain with an example?**

**Ans.** The static keyword is mainly used for memory management in Java. A static keyword can be applied to variable, blocks, methods, and classes. The static keyword is a property of a class rather than an instance of the class. The static keyword is used for a constant variable or a method that is the same for every instance of a class.

1. **What is class loading and how does the Java program actually executes?**

**Ans.** Classloading is the process of loading class files into the JVM(Java Virtual Machine) at runtime.

The class loading process in Java is divided into three phases: loading, linking, and initialization.

1. **Can we mark a local variable as static?**

**Ans.** No, it is not possible to mark a local variable as static in Java. The static keyword is used to define class-level members (variables and methods) that are shared among all instances of a class. Local variables are temporary variables defined within a method, constructor, or block and are not associated with the class itself.

1. **Why is the static block executed before the main method in Java?**

**Ans.** The static blocks are executed before the **main** method. This order ensures that any necessary initialization tasks defined in the static blocks are performed before the execution of the **main** method or any other code within the class.

1. **Why is a static method also called a class method?**

**Ans.** It belongs to the class itself, rather than to individual instances (objects) of the class. Unlike instance methods, which are invoked on specific objects, static methods are invoked on the class itself.

1. **What is the use of static blocks in Java?**

**Ans.** It is used to initialize static data members. It is used to initialize before the main method at the time of class loading.

1. **Difference between Static and Instance variables.**

**Ans.**

1. Associated with Class vs. Object:
   * Static Variables: Static variables, also known as class variables, are associated with the class itself rather than with individual objects or instances of the class. There is only one copy of a static variable that is shared among all instances of the class.
   * Instance Variables: Instance variables, on the other hand, are associated with individual objects or instances of the class. Each object has its own set of instance variables, and the values of these variables can vary between different objects.
2. Memory Allocation:
   * Static Variables: Static variables are allocated memory in a special area known as the "Method Area" or "Static Memory." They are initialized only once when the class is loaded into memory and retain their values throughout the lifetime of the program.
   * Instance Variables: Instance variables are allocated memory in the "Heap" area for each individual object or instance of the class. They are initialized whenever a new object is created and have different values for each object.
3. Access and Scope:
   * Static Variables: Static variables can be accessed using the class name directly, without the need to create an instance of the class. They have class-level scope and are accessible throughout the class, including static methods, instance methods, and other static methods.
   * Instance Variables: Instance variables are accessed through object references and are specific to each object. They have instance-level scope and are accessible within the instance methods or by using object references.
4. Sharing of Data:
   * Static Variables: Since static variables are shared among all instances of a class, modifying the value of a static variable affects all objects or instances of the class. Changes made to a static variable by one object are visible to other objects.
   * Instance Variables: Instance variables are unique to each object or instance of the class. Modifying the value of an instance variable affects only the specific object on which the modification is performed and does not impact other objects.
5. Usage Scenarios:
   * Static Variables: Static variables are commonly used to represent properties or attributes that are shared among all instances of a class, such as constants, configuration settings, or counters.
   * Instance Variables: Instance variables are used to represent properties or attributes that vary between different objects or instances of a class. They store the state of individual objects and hold object-specific data.
6. **Difference between static and non static members.**

**Ans.**

1. Associated with Class vs. Instance:
   * Static Members: Static members, such as static variables and static methods, are associated with the class itself rather than with individual objects or instances of the class. They are shared among all instances of the class.
   * Non-Static Members: Non-static members, also known as instance members, are associated with individual objects or instances of the class. Each object has its own set of non-static members.
2. Memory Allocation:
   * Static Members: Static members are allocated memory in a special area known as the "Method Area" or "Static Memory." They are initialized only once when the class is loaded into memory and retain their values throughout the lifetime of the program.
   * Non-Static Members: Non-static members are allocated memory in the "Heap" area for each individual object or instance of the class. They are initialized whenever a new object is created and have different values for each object.
3. Access and Scope:
   * Static Members: Static members can be accessed using the class name directly, without the need to create an instance of the class. They have class-level scope and are accessible throughout the class, including other static methods and static blocks.
   * Non-Static Members: Non-static members are accessed through object references and are specific to each object. They have instance-level scope and are accessible within the instance methods or by using object references.
4. Sharing of Data:
   * Static Members: Since static members are shared among all instances of a class, modifying the value of a static member affects all objects or instances of the class. Changes made to a static member by one object are visible to other objects.
   * Non-Static Members: Non-static members are specific to each object or instance of the class. Modifying the value of a non-static member affects only the specific object on which the modification is performed and does not impact other objects.
5. Method Dispatch:
   * Static Methods: Static methods are resolved at compile-time based on the class name. The specific static method to be executed is determined by the compiler, not by the runtime polymorphism mechanism. Therefore, static methods do not participate in dynamic method dispatch.
   * Non-Static Methods: Non-static methods support dynamic method dispatch, where the actual method to be executed is determined at runtime based on the object's type. The most specific version of the method associated with the object's actual type is invoked.