:FEATURES IN JAVA:

1. **Object-Oriented:** Java is built on an object-oriented paradigm, allowing for the creation of modular and reusable code through classes and objects.

2. **Platform Independence:** Java programs can run on any device that has a Java Virtual Machine (JVM), making it platform-independent. This is possible due to its "Write Once, Run Anywhere" (WORA) philosophy.

3. **Simple and Familiar Syntax:** Java's syntax is similar to C and C++, making it easier for developers familiar with these languages to learn and use Java.

4. **Automatic Memory Management:** Java manages memory allocation and deallocation through garbage collection, freeing developers from manual memory handling and reducing the risk of memory leaks.

Memory leaks:
Memory leaks in Java occur when objects that are no longer needed are not properly removed from memory, leading to a continuous increase in memory usage over time. This can eventually cause the application to run out of memory (OutOfMemoryError).

5. **Security:** Java places a strong emphasis on security, providing a secure runtime environment with features like sandboxing to prevent unauthorized access.

Sandboxing:
In Java, sandboxing refers to the practice of creating a controlled environment for running untrusted code or applications to prevent them from causing harm to the system. This is achieved through the use of the Java Security Manager and the concept of a "sandbox."

6. **Multithreading:** Java supports multithreading, enabling concurrent execution of multiple threads within a program, leading to better resource utilization and performance enhancement.

7. **Rich API:** Java offers a vast collection of built-in APIs for various tasks such as networking, I/O, database connectivity, and more, simplifying development by providing ready-to-use functionalities.

8. **Portability:** Java's platform independence allows applications to be easily ported across different systems without significant modifications.

9. **High Performance:** While not as fast as lower-level languages, Java's optimizations and Just-In-Time (JIT) compilation contribute to relatively high performance.

10. **Community Support:** Java has a large and active developer community, providing extensive resources, libraries, frameworks, and tools to aid in software development.

1.Object-Oriented

```
class Car {
    private String brand;
    private String model;
    // Constructor
    public Car(String brand, String model) {
        this.brand = brand;
        this.model = model;
    }
    // Method to display car details
    public void displayDetails() {
        System.out.println("Brand: " + brand + ", Model: " + model);
    }
}
public class Main {
    public static void main(String[] args) {
        // Creating instances(objects) of the Car class
        Car myCar = new Car("Toyota", "Corolla");
        Car anotherCar = new Car("Honda", "Civic");
        myCar.displayDetails(); // Output: Brand: Toyota, Model: Corolla
        anotherCar.displayDetails(); // Output: Brand: Honda, Model: Civic
    }
}
```

Explanation:

- `Car` is a class that represents a car and has attributes (`brand` and `model`) along with a method (`displayDetails`) to show car details.
- `Main` is another class containing the `main` method, the entry point for execution.
- Inside `main`, two `Car` objects (`myCar` and `anotherCar`) are created using the `new` keyword, initializing them with specific brand and model values.
- The `displayDetails` method is called on both objects to print their details to the console.

2.Platform Independence

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Explanation:

- The `HelloWorld` class contains the `main` method which prints "Hello, World!" to the console using `System.out.println`.
- This code can be compiled and executed on any platform (Windows, Linux, macOS) without modification because Java code is compiled into bytecode that runs on a JVM.

3.Automatic Memory Management (Garbage Collection)

```
public class MemoryExample {
    public static void main(String[] args) {
        for (int i = 0; i < 10000; i++) {
            String temp = "Object_" + i;
        }
        // Garbage collection occurs after the loop when objects are no longer in use
    }
}
```

Explanation:

- Inside the `main` method, a loop creates 10,000 `String` objects (`Object_0`, `Object_1`, ..., `Object_9999`) and assigns them to the variable `temp`.
- As the loop progresses, each `String` object is created, but after each iteration, the `temp` variable loses its reference to the previously created object.
- Once an object loses all references, it becomes eligible for garbage collection, freeing up memory automatically.