Static methods are methods that belong to the class itself rather than to any specific instance (object). They can be called directly on the class without needing to create an object.

**Key Characteristics:**

* **Belongs to the Class:** Associated with the class blueprint, not individual objects.
* **Accessed by Class Name:** Can be called directly using the class name (e.g., MyClass.staticMethod()).
* **Limitations:**
  + Can only directly access other static members (variables or methods) of the same class.
  + Cannot directly access instance variables or instance methods because instance members are tied to specific objects. To access instance members, a static method would need to receive an object of the class as a parameter.

**When to Use Static Methods:**

1. **Utility Methods:** Performing general operations that don't depend on the state of a specific object (e.g., mathematical functions, string manipulation).

Java

public class MathUtils {

public static int add(int a, int b) {

return a + b;

}

}

1. **Factory Methods:** Providing controlled ways to create instances of a class.

Java

public class LoggerFactory {

public static Logger createLogger(String name) {

return new Logger(name);

}

}

1. **Accessing Static Variables:** Providing controlled access to read or modify static class-level data.

Java

public class AppConfig {

private static String version = "1.0";

public static String getVersion() {

return version;

}

}

1. **Singleton Pattern:** Implementing the Singleton design pattern to ensure only one instance of a class exists.

Java

public class Singleton {

private static Singleton instance;

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

}

1. **main Method:** The entry point of a Java application must be a static method.

Java

public class MyApp {

public static void main(String[] args) {

System.out.println("Application started.");

}

}

Static members (variables and methods) belong to the class itself and are shared or accessible without creating objects. They are useful for representing class-level data and providing utility functions that don't depend on object-specific state. Remember that static methods have limitations in directly accessing instance members.