1. **Reference to a static method** – `ClassName::staticMethod`
2. **Reference to an instance method of a particular object** – `instance::instanceMethod`
3. **Reference to an instance method of an arbitrary object of a particular type** – `ClassName::instanceMethod`
4. **Reference to a constructor** – `ClassName::new`

💡 Examples with Real Scenarios

1️⃣ Static method reference — `ClassName::staticMethod`

Scenario: Logging or utility-based operations.

```
import java.util.Arrays;

public class StaticMethodReference {
    public static void printName(String name) {
        System.out.println("Student: " + name);
    }

    public static void main(String[] args) {
        String[] names = {"Amit", "Priya", "Kiran"};
        Arrays.stream(names).forEach(StaticMethodReference::printName);
    }
}
```

✅ **When to use:**
When your lambda calls a static method like `x -> Class.method(x)`.

2️⃣ Instance method of a particular object — `instance::method`

Scenario: Writing logs to a file using a common logger instance.

```
import java.util.Arrays;
import java.util.List;

public class InstanceMethodReference {
    public static void main(String[] args) {
        List<String> logs = Arrays.asList("Login", "ViewPage", "Logout");
        Logger logger = new Logger();
        logs.forEach(logger::logEvent);
    }
}

class Logger {
    public void logEvent(String event) {
        System.out.println("Event Logged: " + event);
    }
}
```

✅ **When to use:**
When all actions use the same instance (e.g., one database connection, one logger).

3️⃣ Instance method of an arbitrary object — `ClassName::instanceMethod`

Scenario: Sorting employee names alphabetically.

```
import java.util.Arrays;
import java.util.List;

public class ArbitraryMethodReference {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Rahul", "Amit", "Suman");
        names.sort(String::compareToIgnoreCase);
        names.forEach(System.out::println);
    }
}
```

✅ **When to use:**
When you don't care *which* instance calls the method — just the type.

4️⃣ Constructor reference — `ClassName::new`

Scenario: Creating threads or objects dynamically.

```
import java.util.function.Supplier;

public class ConstructorReference {
    public static void main(String[] args) {
        Supplier<Thread> threadSupplier = Thread::new;
        Thread t = threadSupplier.get();
        System.out.println("Thread created: " + t);
    }
}
```

✅ **When to use:**
When lambdas are creating new objects, like factories or parallel streams.

🚀 Advantages

- Cleaner, readable code.
- Reduces boilerplate lambdas.
- Encourages reuse of existing methods.
- Improves maintainability in large codebases.

🏢 Real-World Scenarios

1. **Data processing pipelines** – using `Stream` APIs for filtering, mapping, and collecting.
2. **Logging frameworks** – passing logging methods as references.
3. **Thread creation** – `Thread::new` or `Runnable` lambdas for parallel tasks.
4. **Event listeners** – linking UI events to existing handler methods.
5. **Factory patterns** – creating object suppliers with constructor references.