1. Interview-Style Opening

"These are two powerful features—introduced in Java 8 and Java 10 respectively—that help make our code more concise and readable.

Method References (Java 8) are basically syntactical sugar for Lambda expressions. Instead of writing `x -> System.out.println(x)`, you can write `System.out::println`, which is cleaner.

Variable Inference (Java 10), often called 'var', allows us to let the compiler figure out the type of a local variable, like `var list = new ArrayList<String>()`, reducing boilerplate without sacrificing static type safety."

2. Method References (Java 8)

What is it?
It is a shorthand notation of a lambda expression to call a method. If your lambda does nothing but call an existing method, you can replace it with a Method Reference.

Syntax: `ClassName::methodName` or `instance::methodName`

Four Types:

1. **Static Method Reference:** `ClassName::staticMethod`
   - *Lambda:* `(s) -> Integer.parseInt(s)`
   - *Ref:* `Integer::parseInt`
2. **Instance Method of a Specific Object:** `instance::instanceMethod`
   - *Lambda:* `(s) -> System.out.println(s)`
   - *Ref:* `System.out::println`
3. **Instance Method of an Arbitrary Object (Type):** `ClassName::instanceMethod`
   - *Lambda:* `(s) -> s.toUpperCase()`
   - *Ref:* `String::toUpperCase`
4. **Constructor Reference:** `ClassName::new`
   - *Lambda:* `() -> new ArrayList<>()`
   - *Ref:* `ArrayList::new`

Java Code Example:

```java
List<String> names = Arrays.asList("alice", "bob", "charlie");

// 1. Using Lambda
names.forEach(s -> System.out.println(s));

// 2. Using Method Reference (Cleaner)
names.forEach(System.out::println);
```

3. Variable Inference (Java 10)

What is it?
Local Variable Type Inference (LVTI) allows you to use the keyword `var` instead of explicitly declaring the variable type. The compiler infers the type from the right-hand side (RHS).

Rules:

1. **Local Variables Only:** Cannot be used for class fields, method parameters, or return types.
2. **Must be Initialized:** `var x;` is illegal. You must provide a value: `var x = 10;`.
3. **Static Typing:** It is **not** dynamic typing (like Python/JS). Once inferred as `String`, you cannot assign an `Integer` to it later.

Java Code Example:

```java
// Verbose (Old Way)
Map<String, List<Integer>> map = new HashMap<String, List<Integer>>();

// Concise (With var)
var map = new HashMap<String, List<Integer>>();

// Loop Example
for (var entry : map.entrySet()) {
    System.out.println(entry.getKey());
}
```

4. How to Explain This to the Interviewer

"I use **Method References** whenever a lambda simply passes its arguments directly to another method. It makes stream pipelines like `.map(String::toUpperCase)` much easier to read.

I use **Variable Inference (`var`)** to reduce verbosity, especially with complex generic types. However, I follow a rule: only use `var` if the type is obvious from the context. If `var result = service.process()` hides what `result` actually is, I prefer explicit typing for clarity."

5. Edge Cases & Constraints

- **Nulls:** `var x = null;` is a compile-time error because the compiler can't guess the type.
- **Lambdas:** `var f = (s) -> s.length();` is illegal because lambdas need an explicit target type (Functional Interface).

```
*   **Polymorphism:** `var list = new ArrayList<String>();` infers `ArrayList<String>`, r
```

⁂

1. https://www.geeksforgeeks.org/java/java-method-references/
2. https://www.tutorialspoint.com/java/java_method_references.htm
3. https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html
4. https://www.baeldung.com/java-method-references
5. https://codefinity.com/courses/v2/190d2568-3d25-44d0-832f-da03468004c9/c0bcd017-ff39-46ec-bc93-acd569f3497d/f79fe204-c582-4430-9d5e-d6b6377d741e
6. https://herovired.com/home/learning-hub/topics/method-reference-in-java-8
7. https://www.upgrad.com/blog/method-reference-in-java-8/
8. https://www.danvega.dev/blog/java-method-references
9. https://www.scaler.com/topics/method-reference-in-java/
10. https://www.youtube.com/watch?v=DELCbBuCHHE