Factory constructor in Dart **is a special type of constructor that allows you to return an instance of the class, potentially reusing an existing instance or returning a subclass instance. Unlike a regular constructor, a factory constructor doesn't necessarily create a new instance every time it is called.**

When to Use a Factory Constructor

1. **Singleton Pattern**: To ensure a class has only one instance, you can use a factory constructor to implement the Singleton pattern.
2. **Instance Reuse**: When you want to control the instance creation, possibly reusing an existing instance, or creating instances conditionally.
3. **Returning Subtypes**: If you want to return a subtype of the class instead of the class itself.

How to Declare a Factory Constructor

Here's a simple example:

```
class Logger {
  // Private named constructor
  Logger._internal();

  // Static variable to hold the single instance
  static final Logger _instance = Logger._internal();

  // Factory constructor to return the single instance
  factory Logger() {
    return _instance;
  }

  void log(String message) {
    print('Log: $message');
  }
}

void main() {
  // Both logger1 and logger2 will point to the same instance
  var logger1 = Logger();
  var logger2 = Logger();
```

```
  if (logger1 == logger2) {
    print('Both loggers are the same instance.');
  }

  logger1.log('This is a log message.');
}
```

Explanation of the Example:

- **Private Constructor (`Logger._internal()`)**: This constructor is private and cannot be accessed directly from outside the class.
- **Static Instance (`_instance`)**: A static final variable holds the single instance of the `Logger` class.
- **Factory Constructor**: The factory constructor (`factory Logger()`) returns the existing instance stored in `_instance` rather than creating a new one.

Key Points to Remember:

- **Return Types**: A factory constructor can return an instance of its class or any subclass.
- **No Implicit `this` Reference**: Since factory constructors do not create an instance, they do not have access to `this`.

Factory constructors are particularly useful in scenarios where instance management or control over object creation is needed.