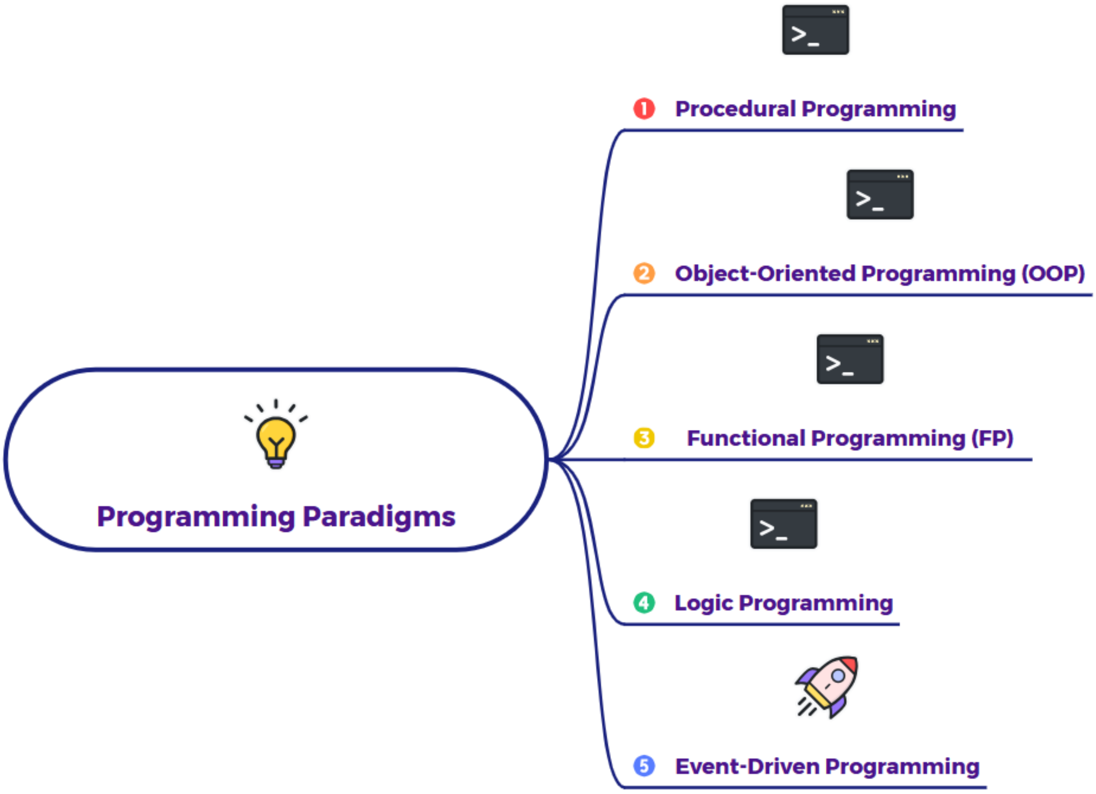
Explain Event driven programming with an example


Programming Paradigms
1 Procedural Programming
2 Object-Oriented Programming (OOP)
3 Functional Programming (FP)
4 Logic Programming
5 Event-Driven Programming

The "Event-Driven Coffee Shop" Way:

Imagine a coffee shop where the staff doesn't just follow a fixed, pre-written script from start to finish. Instead, they primarily **react to things that happen (events)** in the shop.

1. The "Events" (Things That Happen):

In our coffee shop, "events" are anything that demands attention or triggers a response.

- A customer walks in.
- A customer orders a coffee.
- The coffee machine finishes brewing.
- The cash register makes a "ding" sound.
- A new shipment of beans arrives.
- It starts raining outside (might trigger a change in sales strategy).

2. The "Event Listeners/Handlers" (Staff Who Respond):

For each type of event, there's someone (or something) specifically "listening" for it and knows how to respond. These are your "event handlers" or "event listeners."

- **Door Greeter:** Listens for "customer walks in" event. Response: "Welcome!"
- **Barista:** Listens for "customer orders coffee" event. Response: Starts making coffee.
- **Cashier:** Listens for "customer wants to pay" event. Response: Calculates bill, takes money.
- **Coffee Machine:** Listens for "start brewing" command. Generates "brewing finished" event when done.
- **Inventory Manager:** Listens for "new shipment arrives" event. Response: Updates stock, stores beans.

3. The "Event Loop" (The Central Manager):

There's no single manager telling everyone exactly what to do at every microsecond. Instead, there's an unspoken understanding, a continuous cycle:

- "Is anything happening?"
- "Yes, the customer just ordered!" -> Tell Barista to handle.
- "Is anything happening now?"
- "Yes, the coffee machine just dinged!" -> Tell Barista to serve.
- "Is anything happening now?"
- (And so on, constantly checking and delegating based on what just occurred).

This constant checking and delegating of events is like the **"event loop."**

How it Works in the Coffee Shop:

1. **Customer walks in (Event).**
2. The **Door Greeter (Event Listener)** notices this.
3. The Door Greeter gives a "Welcome!" (Action/Response).
4. **Customer says "I'd like a latte" (Event).**
5. The **Barista (Event Listener)** hears the order.
6. The Barista starts making the latte (Action/Response).
7. While the Barista is busy, another **customer puts money on the counter (Event).**
8. The **Cashier (Event Listener)** sees the money.
9. The Cashier processes the payment (Action/Response).
10. **The latte machine beeps "finished!" (Event).**
11. The **Barista (Event Listener)** hears the beep.
12. The Barista serves the latte to the customer (Action/Response).

Notice that the order of operations isn't fixed from the start. The staff only do something *when an event occurs* that they are configured to handle. They don't just stand there following a pre-written script, waiting for step 5, then step 6, etc. They are always ready to react.

Key Points of Event-Driven Programming in this Example:

- **Reactivity:** The system (coffee shop staff) primarily reacts to external occurrences.
- **Decoupling:** The Barista doesn't need to know *who* ordered the coffee, just that an "order coffee" event happened. The Cashier doesn't care if it's a coffee or a pastry order, just that a "payment" event happened. Different parts of the system are less dependent on each other.
- **Asynchronous:** Multiple things can happen concurrently. The Barista can be making coffee while the Cashier takes payment for another customer. They don't block each other's work waiting for a strict sequence.

Why is this "Event-Driven" way useful?

- **Responsiveness:** Great for things where you need to react instantly to user actions or external changes (like clicking buttons on a website, sensor readings, or messages arriving).
- **Flexibility:** Easily add new features by just adding new "event listeners" for new types of events without changing existing code.
- **Efficiency:** Can handle many things happening at once without getting stuck waiting.

Event-Driven Programming is like a busy coffee shop where staff don't follow a fixed script but constantly watch for and react to things that happen (events), making the whole operation responsive and flexible.