To avoid confusion between your `SagaMessage` class and the `Panier` class, and to help you visualize the flow of the Saga pattern, here is a summarized workflow, including the roles of each class in the system.

Workflow of the Saga Pattern

The main goal of the Saga is to handle distributed transactions and ensure that each service involved in the transaction either commits or compensates in case of failure. Here's how the services interact in the Saga, focusing on `Panier`, `Commande`, and the `SagaMessage`.

Schema of the Workflow:

```
+--------------------+              +--------------------+              +-------
-----------+
|                    |              |                    |              |
|
|  Panier Service    |     ->       | Saga Orchestrator |     ->        |
Commande Service |
|                    |              |                    |              |
|
+--------------------+              +--------------------+              +-------
-----------+
        |                              |                                  |
        | (1) Panier prepares order    |                                  |
        |     - Check stock            |                                  |
        |     - Send SagaMessage       |                                  |
        |                              |                                  |
        +----------------------------->|                                  |
        |                              | (2) Saga Orchestrator decides     |
        |                              |     - Prepare the next step       |
        |                              |     - Send SagaMessage to         |
        |                              |       Commande service            |
        |                              |                                  |
        |                              +--------------------------------->|
        |                              |                                  |
        |                              | (3) Commande processes the order |
        |                              |     - Verify inventory, charge   |
        |                              |       payment                    |
        |                              |     - Send SagaMessage           |
        |                              |     - Commit/Compensate          |
        |                              |                                  |
        |<-----------------------------+                                  |
        | (4) Panier receives result   |                                  |
        |     - Complete/Fail order    |                                  |
        |     - Send SagaMessage       |                                  |
        +----------------------------->|                                  |
        |                              |                                  |
+--------------------+              +--------------------+              +-------
-----------+
|                    |              |                    |              |
|
|   Panier Service  |  <----->     | Saga Orchestrator |  <----->     |
Commande Service |
|                   |              |                   |              |
|
+-------------------+              +-------------------+              +-------
-----------+
```

Explanation of Each Role:

1. **Panier Service:**
   - **Role:** Handles inventory and availability. The Panier service checks if the items requested in an order are available. If the items are available, it proceeds to confirm the order; if not, it sends a compensation message to rollback the process.
   - **Saga Message:**
     - The Panier service creates and sends a `SagaMessage` with the initial status, item availability (`dispo`), quantity (`qte`), and required quantity (`requiredQte`).
     - If the Panier service cannot fulfill the order, it sends a failure message to the Saga Orchestrator to trigger a compensation process.
2. **Saga Orchestrator:**
   - **Role:** Coordinates the Saga across the services. The orchestrator decides whether to proceed with the next step or trigger a rollback in case of a failure. It routes messages between the services.
   - **Saga Message:**
     - The orchestrator sends the `SagaMessage` to the next service in the saga (in this case, `Commande`) with the necessary data (e.g., `orderId`, `requiredQte`, and `status`).
     - The orchestrator handles the response from the services and initiates compensation if any service fails.
3. **Commande Service:**
   - **Role:** Processes the order, which includes verifying payment, handling inventory, and updating the database. Once the order is successfully processed, the service sends a confirmation or compensation message back to the orchestrator.
   - **Saga Message:**
     - The `Commande` service receives the `SagaMessage` and checks if it can process the order (e.g., charging payment).
     - If successful, it sends a success message back; if something goes wrong, it sends a failure message to trigger the compensation process.

How `SagaMessage` Will Not Be Confused with `Panier` Class:

- **Naming:** The `SagaMessage` class is specifically used for communication in the Saga pattern between the services (Panier and Commande). It is a distinct class with its own responsibility: to carry the data between services during the Saga. It's clear that this class is used for the orchestration of distributed transactions, while the `Panier` class is focused on inventory and order management.

- **Class Separation:**
  - The `SagaMessage` class should be part of the orchestration mechanism and should not be mixed up with business classes like `Panier`.
  - In your `Panier` service, the business logic (inventory management, etc.) is separate from the orchestration logic (sending messages between services). The `SagaMessage` will simply carry the relevant data about the order without conflicting with the domain logic in `Panier`.
- **Service Boundaries:**
  - Since `Panier` is responsible for managing inventory, and `SagaMessage` is related to coordinating the Saga steps, these two classes have different roles and will not overlap.
  - The `SagaMessage` is part of the communication layer (likely using messaging systems like Kafka or RabbitMQ), while `Panier` is part of the business logic layer of your application.

Detailed Role of `SagaMessage`:

- **Panier Service sends a `SagaMessage` to the Saga Orchestrator.**
  - This message contains information such as `orderId`, `qte`, `dispo`, and `requiredQte`.
  - The orchestrator decides whether to proceed or trigger compensation.
- **Saga Orchestrator sends the `SagaMessage` to the `Commande` service.**
  - The `Commande` service processes the message and checks if the order can be processed.
- **If any service (Panier or Commande) encounters a failure, the orchestrator initiates a compensation process and sends a rollback `SagaMessage` to reverse actions taken.**

Summary of Roles:

1. **Panier Service:** Manages inventory and availability. It uses the `SagaMessage` to communicate availability and inventory status to the orchestrator.
2. **Saga Orchestrator:** Coordinates the saga process, sending `SagaMessages` between services (Panier and Commande).
3. **Commande Service:** Handles order processing (payment, inventory) and sends success or failure back to the orchestrator.

This setup ensures that the `SagaMessage` is not confused with the `Panier` service, as they perform very distinct roles in the architecture.