Here's an implementation of the `SagaMessage` class for the Panier service. This class will carry the necessary information for the Saga communication between the `Panier` and `Commande` services.

`SagaMessage` Class

```
package com.example.PanierService.model;

import java.io.Serializable;

public class SagaMessage implements Serializable {

    private String status;       // Status of the request (e.g., success,
fail)
    private String orderId;      // Order ID related to the Panier and
Commande
    private int requiredQte;     // Required quantity for the Panier
    private int qte;             // Quantity available in the Panier
    private boolean dispo;       // Availability of the item in the Panier

    // Default constructor
    public SagaMessage() {}

    // Constructor for initial message (e.g., for initiating the saga)
    public SagaMessage(String status) {
        this.status = status;
    }

    // Constructor for detailed message (e.g., during Panier or Commande
processing)
    public SagaMessage(String status, String orderId, int requiredQte) {
        this.status = status;
        this.orderId = orderId;
        this.requiredQte = requiredQte;
    }

    // Constructor with all fields (for more detailed Saga steps)
    public SagaMessage(String status, String orderId, int requiredQte, int
qte, boolean dispo) {
        this.status = status;
        this.orderId = orderId;
        this.requiredQte = requiredQte;
        this.qte = qte;
        this.dispo = dispo;
    }

    // Getters and Setters
    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }
```

```
    public String getOrderId() {
        return orderId;
    }

    public void setOrderId(String orderId) {
        this.orderId = orderId;
    }

    public int getRequiredQte() {
        return requiredQte;
    }

    public void setRequiredQte(int requiredQte) {
        this.requiredQte = requiredQte;
    }

    public int getQte() {
        return qte;
    }

    public void setQte(int qte) {
        this.qte = qte;
    }

    public boolean isDispo() {
        return dispo;
    }

    public void setDispo(boolean dispo) {
        this.dispo = dispo;
    }

    @Override
    public String toString() {
        return "SagaMessage{" +
                "status='" + status + '\'' +
                ", orderId='" + orderId + '\'' +
                ", requiredQte=" + requiredQte +
                ", qte=" + qte +
                ", dispo=" + dispo +
                '}';
    }
}
```

Explanation:

1. **Fields:**
    - `status`: Represents the status of the current step in the saga (e.g., `success`, `fail`).
    - `orderId`: A unique identifier for the order, shared across services in the saga.
    - `requiredQte`: The required quantity for the item in the Panier.
    - `qte`: The available quantity in the Panier.
    - `dispo`: Availability of the item in the Panier (boolean value).
2. **Constructors:**
    - A default constructor is provided for serialization purposes.

    - A constructor with `status` is used for initiating the saga.
    - A constructor with `status`, `orderId`, and `requiredQte` is used for passing details during Panier and Commande processing.
    - A constructor with all fields (`status`, `orderId`, `requiredQte`, `qte`, and `dispo`) is used for detailed messages.
3. **Getters and Setters:**
    - Getters and setters are provided for all the fields, allowing you to retrieve and set values for each property.
4. **`toString` method:**
    - The `toString()` method is overridden to provide a string representation of the `SagaMessage` object, which is useful for logging and debugging.

How it works:

- The `SagaMessage` class will be used in the communication between the Panier and Commande services.
- When Panier checks the availability and quantity, it sends a `SagaMessage` containing the necessary details (`orderId`, `requiredQte`, `qte`, and `dispo`).
- If the Panier validation is successful, the `SagaMessage` is passed along to the Commande service to process the order.
- The services communicate via RabbitMQ queues, and the `SagaMessage` contains the relevant data for each step.

Let me know if you need further customization or explanations!