Here are the `SagaPanierService` and `SagaCommandeService` classes based on your requirements, adapted to the workflow and logic of the provided documentation.

SagaPanierService.java

```
package com.example.PanierService.service;

import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Service;

import com.example.PanierService.model.SagaMessage;

@Service
public class SagaPanierService {

    private final RabbitTemplate rabbitTemplate;

    public SagaPanierService(RabbitTemplate rabbitTemplate) {
        this.rabbitTemplate = rabbitTemplate;
    }

    /**
     * Initiates the Saga by invoking the get API to check Panier details.
     */
    public void startPanierSaga() {
        System.out.println("Starting Panier Saga...");
        rabbitTemplate.convertAndSend("saga-exchange", "panier-get-routing-
key", new SagaMessage("CHECK_PANIER"));
    }

    /**
     * Handles the response from the get API (Panier Check).
     */
    @RabbitListener(queues = "panier-get-queue")
    public void handlePanierGetResponse(SagaMessage message) {
        System.out.println("Panier Check Response: " + message.getStatus());
        if ("success".equals(message.getStatus()) && message.getDispo() &&
message.getQte() >= message.getRequiredQte()) {
            System.out.println("Panier validation successful. Proceeding to
Commande...");
            rabbitTemplate.convertAndSend("saga-exchange", "commande-routing-
key",
                    new SagaMessage("PROCESS_COMMANDE", message.getOrderId(),
message.getRequiredQte()));
        } else {
            System.out.println("Panier validation failed. Insufficient
quantity or item unavailable.");
        }
    }

    /**
     * Handles the response from the Commande process.
     */
    @RabbitListener(queues = "commande-response-queue")
    public void handleCommandeResponse(SagaMessage message) {
        System.out.println("Commande Response: " + message.getStatus());
        if ("success".equals(message.getStatus())) {
            System.out.println("Commande processed successfully. Updating
Panier...");
            rabbitTemplate.convertAndSend("saga-exchange", "panier-post-
routing-key",
                    new SagaMessage("UPDATE_PANIER", message.getOrderId(),
message.getRequiredQte()));
        } else {
            System.out.println("Commande failed. No Panier updates
required.");
        }
    }

    /**
     * Handles the response from the post API (Panier Update).
     */
    @RabbitListener(queues = "panier-post-queue")
    public void handlePanierPostResponse(SagaMessage message) {
        System.out.println("Panier Update Response: " + message.getStatus());
        if ("success".equals(message.getStatus())) {
            System.out.println("Panier successfully updated. Saga
completed.");
        } else {
            System.out.println("Panier update failed. Compensation may be
required.");
        }
    }
}
```

SagaCommandeService.java

```
package com.example.CommandeService.service;

import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Service;

import com.example.CommandeService.model.SagaMessage;

@Service
public class SagaCommandeService {

    private final RabbitTemplate rabbitTemplate;

    public SagaCommandeService(RabbitTemplate rabbitTemplate) {
        this.rabbitTemplate = rabbitTemplate;
    }

    /**
     * Handles the incoming request to process a commande.
     */
    @RabbitListener(queues = "commande-routing-queue")
    public void processCommande(SagaMessage message) {
        System.out.println("Processing Commande for Order ID: " +
message.getOrderId());
        try {
            // Simulate Commande processing logic
            boolean success = true; // Replace with actual processing logic
            if (success) {
                System.out.println("Commande processed successfully.");
                rabbitTemplate.convertAndSend("saga-exchange", "commande-
response-routing-key",
                        new SagaMessage("success", message.getOrderId(),
message.getRequiredQte()));
            } else {
                throw new Exception("Commande processing failed.");
            }
        } catch (Exception e) {
            System.out.println("Commande failed: " + e.getMessage());
            rabbitTemplate.convertAndSend("saga-exchange", "commande-response-
routing-key",
                    new SagaMessage("fail", message.getOrderId()));
        }
    }
}
```

Key Points:

1. **Workflow**:
    - `SagaPanierService` starts by validating the Panier via the `GET` API.
    - On successful validation, it invokes the `POST` API via `SagaCommandeService` to process the Commande.
    - After the Commande is successful, it updates the Panier via the `POST` API.
2. **Queues and Routing Keys**:
    - `panier-get-queue`: Handles the response of the Panier `GET` API.
    - `commande-routing-queue`: Routes messages to process the Commande.
    - `commande-response-queue`: Handles the response of the Commande process.
    - `panier-post-queue`: Handles the response of the Panier `POST` API.
3. **Error Handling**:
    - If Panier validation fails, the saga stops.
    - If Commande processing fails, no Panier updates are made.
4. **SagaMessage**:
    - Add fields like `dispo`, `qte`, `requiredQte`, and `orderId` to the `SagaMessage` class to carry the necessary data.

Let me know if you need the `SagaMessage` class implementation!