Flowable Hands-On Exercise: Using a Script Task

Objective:

Learn how to use a **Script Task** in Flowable to execute custom logic within a BPMN process.

Step 1: Set Up the Process Model

1. **Open Flowable Modeler**
   - Navigate to **Flowable Modeler** (Web UI).
   - Click **New Model** → Enter Name and Key → Click **Create**.
2. **Design the BPMN Process**
   - Drag a **Start Event** onto the canvas.
   - Drag a **Script Task** next to it.
   - Drag an **End Event**.
   - Connect all elements: **Start Event** → **Script Task** → **End Event**.
   - Click on the **Script Task** and configure it:
     - **Name:** Execute Script
     - **Script Format:** `groovy` (or JavaScript)
     - **Script Content:**

       ```
       execution.setVariable("greeting", "Hello, Flowable!");
       ```

Step 2: Deploy and Test the Process

1. **Save and Publish the Model**
   - Click **Save** and then **Deploy** the process.
2. **Start the Process**
   - Open **Flowable work**
   - Click **Start Process** → Select the deployed process.
   - Complete the form (if any) and start the instance.
3. **Verify Execution**
   - Open **Flowable Control App** → Navigate to **Process Instances**.
   - Click on the completed instance.
   - Check the **Process Variables** tab → You should see `greeting = "Hello, Flowable!"`.

Step 3: Extend the Script Task

Modify the script to add dynamic variables:

```
def userName = execution.getVariable("userName") ?: "Guest"
execution.setVariable("greeting", "Hello, ${userName}!")
```

- Before starting the process, pass `"userName"` as a process variable.
- Verify that the script customizes the greeting.

Key Takeaways:

- **Script Tasks** allow inline execution of code (Groovy, JavaScript, etc.).
- Process variables can be read (`execution.getVariable()`) and written (`execution.setVariable()`).
- Use the **Flowable Control App** to inspect execution results.

Here are **three additional hands-on exercises** using the Flowable **Script Task**

Exercise 1: Perform Arithmetic Operations in a Script Task

Objective: Use a script task to perform calculations and store the results as process variables.

Steps:

1. **Create a new BPMN model** in Flowable Modeler.
2. Add a **Start Event** → **Script Task** → **User Task** → **End Event**.
3. **Configure the Script Task:**
   - **Name:** Perform Calculation
   - **Script Format:** `groovy`
   - **Script Content:**

   ```
   def num1 = execution.getVariable("num1") ?: 10
   def num2 = execution.getVariable("num2") ?: 5
   execution.setVariable("sum", num1 + num2)
   execution.setVariable("difference", num1 - num2)
   execution.setVariable("product", num1 * num2)
   execution.setVariable("quotient", num2 != 0 ? num1 / num2 :
   "undefined")
   ```

4. **Deploy and Start the Process** with `num1 = 20` and `num2 = 4`.
5. **Verify Output** in the User Task by displaying process variables.

Exercise 2: Conditional Execution Using a Script Task

Objective: Use a script task to check conditions and set process variables dynamically.

Steps:

1. **Create a new BPMN model** with a **Start Event** → **Script Task** → **Exclusive Gateway** → Two **User Tasks** (Approve & Reject) → **End Event**.

2. **Configure the Script Task:**
    - **Name:** Evaluate Score
    - **Script Format:** `groovy`
    - **Script Content:**

    ```
    def score = execution.getVariable("score") ?: 50
    execution.setVariable("status", score >= 60 ? "Approved" :
    "Rejected")
    ```

3. **Configure the Gateway Conditions:**
    - Path to "Approve Task" → `${status == 'Approved'}`
    - Path to "Reject Task" → `${status == 'Rejected'}`
4. **Deploy and Test** the process with different `score` values (e.g., 75, 45) to check the path taken.

Exercise 3: Fetch and Process JSON Data in a Script Task

Objective: Extract values from JSON input using Groovy in a Script Task.

Steps:

1. **Create a new BPMN model** with **Start Event** → **Script Task** → **User Task** → **End Event**.
2. **Configure the Script Task:**
    - **Name:** Process JSON Data
    - **Script Format:** `groovy`
    - **Script Content:**

    ```
    import groovy.json.JsonSlurper

    def jsonData = execution.getVariable("customerData")
    def parsedData = new JsonSlurper().parseText(jsonData)

    execution.setVariable("customerName", parsedData.name)
    execution.setVariable("customerAge", parsedData.age)
    execution.setVariable("isAdult", parsedData.age >= 18)
    ```

3. **Start the Process** with the following JSON input as a variable (`customerData`):

```
{
  "name": "John Doe",
  "age": 25
}
```

4. **Verify Results** in the User Task Form (display `customerName`, `customerAge`, and `isAdult`).

Real-World Scenario: Order Processing with Discount Calculation

Objective:

Use a **Script Task** to calculate discounts dynamically based on order value and customer type.

Scenario:

A company processes customer orders, offering discounts based on:

- **Regular Customers** → 5% discount if the order amount is above **$100**.
- **Premium Customers** → 10% discount if the order amount is above **$100**.
- **VIP Customers** → 15% discount if the order amount is above **$100**.

The process will:

1. Accept order details.
2. Use a **Script Task** to calculate the discount.
3. Pass the final amount to an **Approval Task**.

BPMN Process Model:

1. **Start Event → Script Task (Calculate Discount) → User Task (Order Approval) → End Event**.

Step 1: Configure the Script Task

- **Task Name:** Calculate Discount
- **Script Format:** `groovy`
- **Script Content:**

```
def orderAmount = execution.getVariable("orderAmount") ?: 0
def customerType = execution.getVariable("customerType") ?: "Regular"

def discountRate = 0
if (orderAmount > 100) {
    if (customerType == "Regular") {
        discountRate = 0.05
    } else if (customerType == "Premium") {
        discountRate = 0.10
    } else if (customerType == "VIP") {
        discountRate = 0.15
    }
}
```

```
def discountAmount = orderAmount * discountRate
def finalAmount = orderAmount - discountAmount

execution.setVariable("discountRate", discountRate * 100)  // Store as
percentage
execution.setVariable("discountAmount", discountAmount)
execution.setVariable("finalAmount", finalAmount)
```

Step 2: Configure User Task for Order Approval

- This task will display:
  - **Original Order Amount**
  - **Discount Percentage**
  - **Final Amount After Discount**
  - **Approve or Reject Buttons**

Step 3: Deploy and Test

1. Start the process with different values:
   - Order Amount = **120**, Customer Type = **Premium** → Should apply **10% discount**.
   - Order Amount = **80**, Customer Type = **VIP** → Should apply **0% discount**.
   - Order Amount = **150**, Customer Type = **Regular** → Should apply **5% discount**.
2. Open **Flowable Admin App** → Check if variables are correctly calculated.

Expected Output:

For an order of **$120** from a **Premium Customer**:

- **Discount Rate:** `10%`
- **Discount Amount:** `$12`
- **Final Amount:** `$108`

Enhanced Real-World Scenario: Order Processing with Auto-Approval

Objective:

Extend the previous **Order Processing Workflow** by adding an **Exclusive Gateway** for **auto-approval** when the final amount is below a threshold.

Updated BPMN Process Flow:

1. **Start Event**
2. **Script Task (Calculate Discount)**
3. **Exclusive Gateway (Auto-Approval Check)**
   - If **Final Amount ≤ $100 → Auto-Approved → End Event**
   - If **Final Amount > $100 → User Task (Manager Approval) → End Event**

Step 1: Modify Script Task (Calculate Discount)

- **Script Name:** Calculate Discount
- **Script Format:** `groovy`
- **Script Content:**

```
def orderAmount = execution.getVariable("orderAmount") ?: 0
def customerType = execution.getVariable("customerType") ?: "Regular"

def discountRate = 0
if (orderAmount > 100) {
    if (customerType == "Regular") {
        discountRate = 0.05
    } else if (customerType == "Premium") {
        discountRate = 0.10
    } else if (customerType == "VIP") {
        discountRate = 0.15
    }
}

def discountAmount = orderAmount * discountRate
def finalAmount = orderAmount - discountAmount

execution.setVariable("discountRate", discountRate * 100)  // Store as 
percentage
execution.setVariable("discountAmount", discountAmount)
execution.setVariable("finalAmount", finalAmount)
```

Step 2: Configure the Exclusive Gateway Conditions

1. Add an **Exclusive Gateway** after the **Script Task**.
2. Create two outgoing paths:
   - **Auto-Approve Path →** Connect to **End Event**

        - Condition: `${finalAmount <= 100}`
    - **Manual Approval Path** → Connect to **User Task (Order Approval)**
        - Condition: `${finalAmount > 100}`

Step 3: Configure User Task (Order Approval)

- Assign to **Manager**.
- Display:
    - **Original Order Amount**
    - **Discount Percentage**
    - **Final Amount**
    - **Approve/Reject Buttons**

Step 4: Deploy and Test

Order Amount	Customer Type	Final Amount	Path Taken
120	Premium	108	Manager Approval
80	VIP	80	Auto-Approved
150	Regular	142.5	Manager Approval
90	Regular	90	Auto-Approved

Enhanced Order Processing Workflow with Email Notifications

Objective:

Extend the **Order Processing Workflow** to include **email notifications** when an order is **approved or rejected**.

Updated BPMN Process Flow:

1. **Start Event**
2. **Script Task (Calculate Discount)**
3. **Exclusive Gateway (Auto-Approval Check)**
   - If **Final Amount ≤ $100 → Auto-Approved → Email Task (Approval Notification) → End Event**
   - If **Final Amount > $100 → User Task (Manager Approval)**
4. **Exclusive Gateway (Approval Check)**
   - If **Approved → Email Task (Approval Notification) → End Event**
   - If **Rejected → Email Task (Rejection Notification) → End Event**

Step 1: Modify the Script Task (Calculate Discount)

Use the same script as before, ensuring the **finalAmount** variable is set.

Step 2: Configure the User Task (Order Approval)

- **Task Name:** Manager Approval
- **Assignee:** `Manager`
- **Form Fields:**
  - **Original Order Amount**
  - **Discount Percentage**
  - **Final Amount**
  - **Approval Decision (Approve/Reject Button)**

Step 3: Add Email Tasks

1. **Auto-Approval Path**
   - Add a **Service Task** after the auto-approval gateway.
   - Set **Task Type:** Send Email
   - **To:** `${customerEmail}`
   - **Subject:** `"Your order has been auto-approved!"`
   - **Body:**

```
Hello,

Your order of ${orderAmount} has been processed.
After applying a discount of ${discountAmount}, your final
amount is ${finalAmount}.
Since your final amount is below $100, it has been
automatically approved.

Thank you for your order!
```

2. **Manager Approval Path**
   - Add another **Exclusive Gateway** after the **User Task (Approval)**.
   - If **Approved → Send Email Task (Approval Notification)**
   - If **Rejected → Send Email Task (Rejection Notification)**
3. **Approval Notification Email Task**
   - **Task Type:** Send Email
   - **To:** `${customerEmail}`
   - **Subject:** `"Your order has been approved"`
   - **Body:**

```
Hello,

Your order of ${orderAmount} has been approved.
The applied discount is ${discountAmount}, making your final
amount ${finalAmount}.

Thank you for shopping with us!
```

4. **Rejection Notification Email Task**
   - **Task Type:** Send Email
   - **To:** `${customerEmail}`
   - **Subject:** `"Your order has been rejected"`
   - **Body:**

```
Hello,

Unfortunately, your order of ${orderAmount} has been rejected
after review.

If you have any questions, please contact our support team.

Best regards,
Order Management Team
```

Step 4: Deploy and Test

✅ Test Case 1 (Auto-Approval)

- Order Amount = **$80**
- Expected: **Auto-approved, email sent**

✅ Test Case 2 (Manager Approval - Approved)

- Order Amount = **$150**
- Manager Approves → Expected: **Approval email sent**

✅ **Test Case 3 (Manager Approval - Rejected)**

- Order Amount = **$200**
- Manager Rejects → Expected: **Rejection email sent**
