**CAMUNDA [2025]**

**Chapter 01: Getting Familiar with BPMN & Camunda**

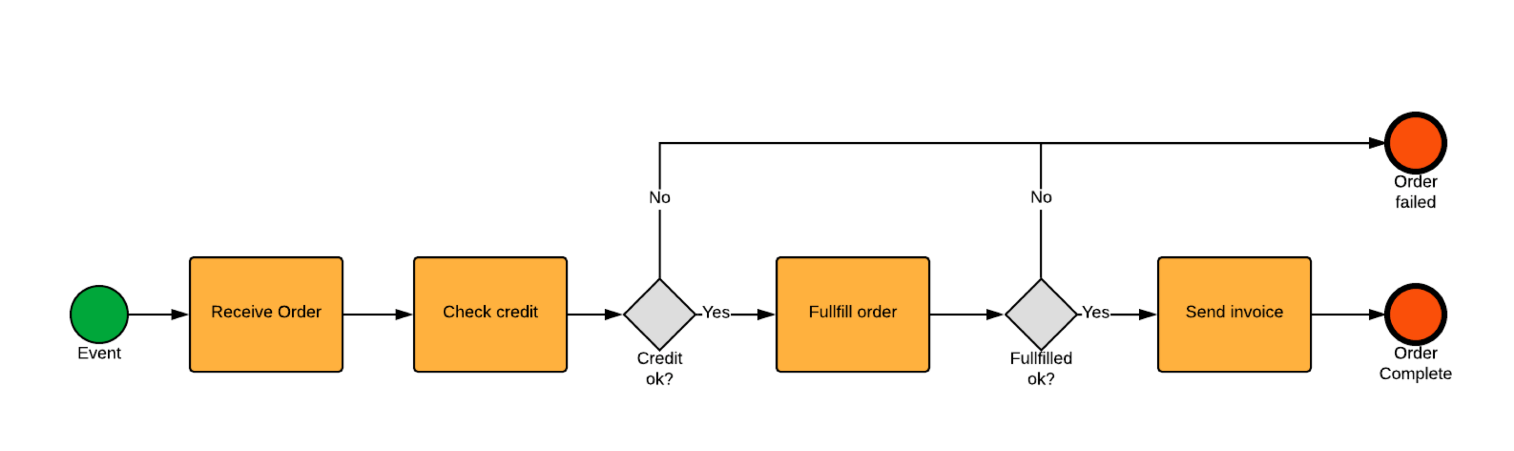
# BPMN: **BPMN** stands for **Business Process Model and Notation**. It is a **visual language** (like a flowchart) used to design and understand **business processes**.

**Imagine this:** You want to explain a process like:"When a user places an order, check if payment is done. If yes, ship the product. If not, send a reminder."

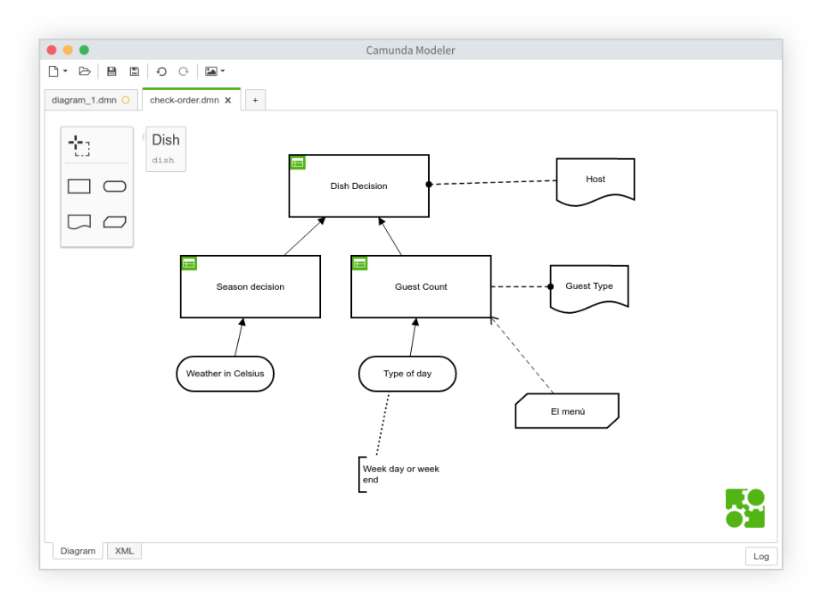
With BPMN, you can draw this process using **symbols**, not long documents.

# Elements in BPMN

|  |  |
| --- | --- |
| Symbol | Meaning |
| ○ | Start Event (where process begins) |
| ▭ | Task (an action, like "check payment") |
| ◇ | Gateway/Decision (yes or no) |
| ◎ | End Event (where process ends) |
| → | Arrows (to show flow) |



**# Camunda Modeler: It** is a free desktop application that helps you **create BPMN diagrams** (business workflows) and **DMN diagrams** (decision tables).You use it to **draw the process** that Camunda Engine will later **execute**.



**Q-Why Do We Use BPMN?**

1. **Easy to Understand:** BPMN looks like a flowchart, so both technical and non-technical people can follow it easily.
2. **Visualizes the Entire Process:** It clearly shows each step in a process, who does what, and the flow of tasks and decisions.
3. **Improves Communication:** Everyone (managers, developers, clients) can understand the same diagram, reducing confusion.
4. **Supports Automation with Tools like Camunda:** You can design a process and run it using a process engine like Camunda.
5. **Helps Find and Fix Problems Faster:** Visual workflows make it easier to spot issues, improve steps, and save time.

**Note**: Business Analyst, Technical Developers and Business Managers are the users for creating the process, implementing the process and monitoring the process respectively.

# **BPMN Tools (Open-Source vs Paid)**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Tool Name | Description | Platform |
| 🔓 Open Source | **Camunda Modeler** | Desktop app to design BPMN, DMN, and forms. Works with Camunda Engine. | Windows, macOS, Linux |
|  | **bpmn.io** | Web-based BPMN editor (from Camunda team). Simple and developer-friendly. | Web Browser |
|  | **Flowable** | Java-based BPMN engine + modeler. Great for embedding in applications. | Web/Desktop |
|  | **ProcessMaker OSS** | Workflow automation platform with BPMN 2.0 support. | Web |

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Tool Name | Description | Platform |
| 💰 Paid | **Signavio (by SAP)** | Cloud-based BPM suite for modeling, collaboration, and optimization. | Web |
|  | **Bizagi Modeler** | Easy BPMN diagram tool. Free to model; cloud automation features are paid. | Windows/Web |
|  | **IBM Blueworks Live** | Professional BPM tool for process discovery and modeling. | Web |
|  | **ARIS (Software AG)** | Enterprise BPM platform for modeling, analytics, and governance. | Web/Desktop |

**# Camunda & Camunda BPMN**

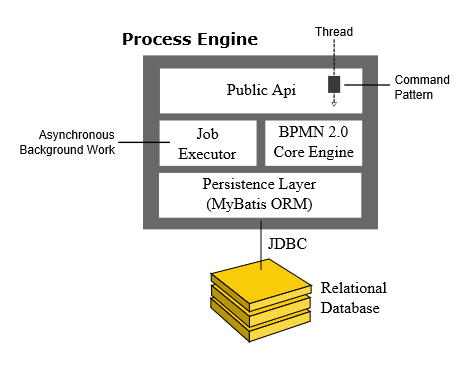
**Camunda** is a Java Framework that helps you **create, run, and manage business processes** using diagrams. You **design the process using BPMN**, and Camunda makes it **work automatically**.

**Camunda BPMN** is a light-weight and open-source platform for Business Process Management. In this, we use **BPMN diagrams** (flowcharts of your process) **inside Camunda**.

# **More About Camunda**

1. **Camunda runs inside Java:** Camunda is a tool that works using the **Java language**. It runs inside something called the **Java Virtual Machine (JVM)** like a container for running Java programs.
2. **Works well with Java frameworks:** Camunda works very smoothly with **Java EE 6** and **Spring Framework**, which are popular tools used to build backend applications.
3. **Useful for managing tasks:** Camunda gives you tools to manage **human tasks**, **processes**, and to **monitor** what is happening like tracking who did what and when.

# **Camunda Architecture**

It consists of an engine and a modeler, where the **engine runs** your process while the **modeler designs** your process. Camunda handles both!

**🔄 Process Engine & Infrastructure**

1. The **process engine** is like the **brain** of Camunda. It runs your **BPMN workflows** (like "Approve leave"), **CMNN cases** (used for flexible processes), and **DMN decisions** (used for rules like "If age > 18, approve").

Note: A process engine is a Java library that understands BPMNs and DMNs and is responsible for executing them.

1. Camunda includes an **H2 database** inside it (for saving data). This is useful for testing or trying things quickly.
2. It uses **MyBatis** to connect with the database. MyBatis is a tool that helps Java code and database work together easily.

**Process Engine Architecture**

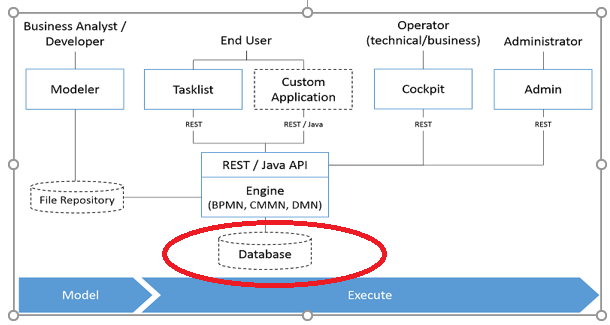
1. **Public API**: This is the entry point through which your Java application communicates with the **Camunda Process Engine**. It provides methods and interfaces to start processes, complete tasks, query process status, etc. Think of it as a bridge between your business logic (Java code) and the engine running the workflow.
2. **BPMN Core Engine**: This component is responsible for reading and interpreting **BPMN files** (Business Process Model and Notation). BPMN is a standard way to design and describe business workflows. The core engine makes sense of these files and runs the processes as per their design—like flows, decisions, user tasks, etc.
3. **Job Executor**: Not everything in a process happens immediately—some steps are scheduled, like timers, asynchronous tasks, or retries. The Job Executor handles such **background jobs**. It picks up these jobs from the database and executes them in the background without blocking the main process.
4. **Persistence Layer**: A running business process has state—like what step it is at, what data it holds, etc. The **Persistence Layer** saves this process state into a **relational database** (like PostgreSQL, MySQL). This ensures that even if the system restarts or crashes, the process can resume from where it left off.

**📝 Modeler**

1. The **Camunda Modeler** is a **free tool** you use to **draw your processes**.
2. You can design:

* **BPMN diagrams** (for workflows),
* **CMMN diagrams** (for case-based tasks),
* **DMN tables** (for rule-based decisions).

# **Camunda Ecosystem**



The **Camunda ecosystem** is a set of tools that work together to help you **design, run, and manage business processes**.

1. At the center is the **Camunda Process Engine**. All these tools **talk to the Camunda Engine**, which **runs the BPMN, CMMN, and DMN**. It understands and executes workflows designed using **BPMN diagrams**. You can also use **DMN tables** to define rules (like "If age > 18, approve"), and **CMMN** for flexible case management.
2. **A Business Analyst / Developer** uses the **Camunda Modeler** to **draw the process diagram** (like a flowchart). It is a free desktop tool where you drag and drop flow elements.
3. Camunda also gives you tools like:

* **Tasklist** – For users to see and complete their assigned tasks.
* **Cockpit** – For developers/admins to monitor running processes.
* **Operate** (in Camunda 8) – Helps track and manage live processes.
* **Zeebe** (in Camunda 8) – A cloud-native workflow engine for high performance.

Note: Camunda connects well with **Java, Spring Boot**, REST APIs, and many databases. It's flexible and useful for automating any real-world process like approvals, registrations, or customer support.

**End Users** (like employees) use **Tasklist** to see and complete tasks assigned to them Or a **Custom Application** (your own app) built using Java or REST API.

**Operators** (tech/business team) use **Cockpit** to **monitor and track** what’s going on in the process.

**Administrators** use the **admin tool** to handle **user access, settings, etc.**

1. The engine connects with a **Database [H2, MySQL etc.]** (red circle in image) to **save all data** like tasks, decisions, process status, etc. There's also a **File Repository** to store files if needed.
2. Once your process is designed, you **deploy it to the engine**, which then manages tasks, decisions, and user inputs automatically.

# **Do you know**

1. Camunda Forum or Community: <https://forum.camunda.io/>

Community discussion on Camunda, including business process management and process automation.

1. Camunda GitHub: <https://github.com/camunda>

You can check this too: <https://github.com/camunda/camunda-bpm-examples>

1. Camunda Blogs: <https://camunda.com/blog/>
2. Camunda Documentation: <https://docs.camunda.org/manual/latest/>
3. To make a Spring Boot Camunda Project: <https://start.camunda.com/>

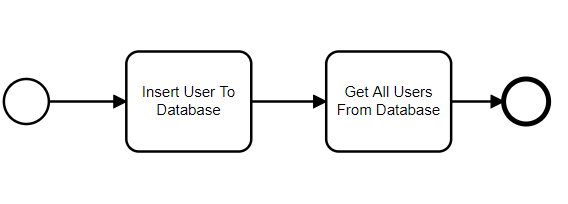
# **Complete Camunda Environment Setup Installation Steps**

1. Install JAVA 17 or 21
2. Install IntelliJ
3. Setup Path in “environment variables”
4. Install Camunda Modeler: <https://downloads.camunda.cloud/release/camunda-modeler/4.12.0/>

Extract the downloaded zip and run “Camunda Modeler.exe”

Note: You can add new plugins to enhance its functionalities: Camunda Modeler\resources\plugins and download it from <https://github.com/rob2universe/plugins/archive/20210303.zip>

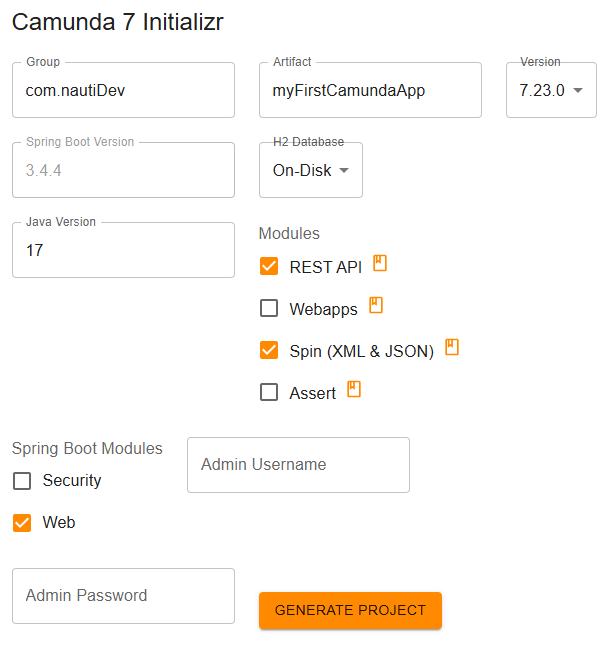
# **Camunda Modeler**



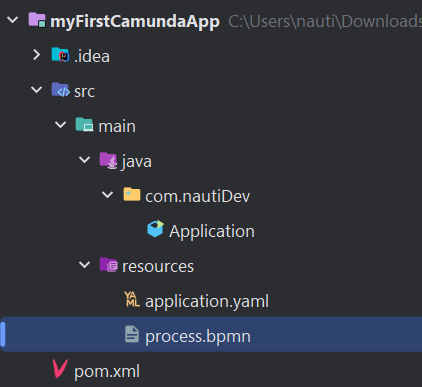
* Here, it is a simple BPMN, two tasks have been carried out.
* Light circle is starting point while dark circle is the ending point, describing overall execution.

**Chapter 02: Creating Our First Spring Boot Camunda Application**

1. Go to <https://start.camunda.com/>



1. Project Structure



**Note:**

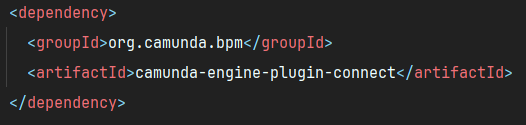
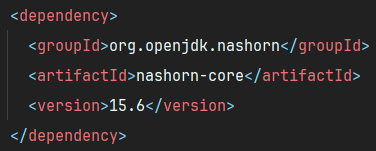
**Camunda Spin: Camunda Spin** is a **data processing library** used inside **Camunda BPM** to make it easy to work with structured data like **JSON** and **XML** in your BPMN processes.

In business processes, you often need to:

* Read or write JSON or XML documents.
* Extract or update values inside those documents.
* Pass structured data between tasks.

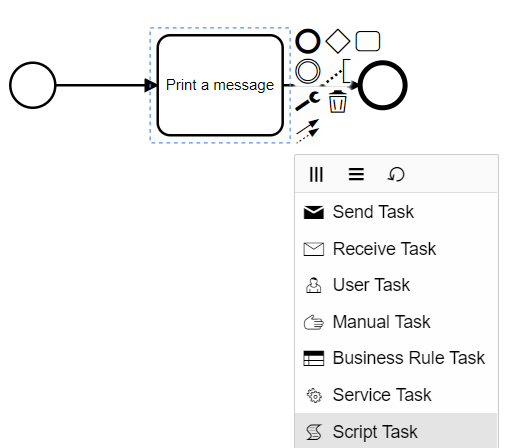
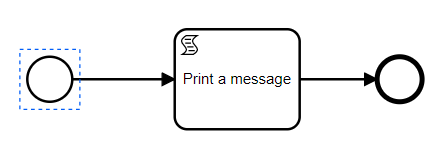
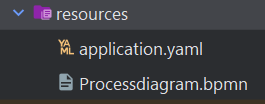
Without Spin, you'd need custom Java code or external tools. Spin simplifies this by giving you powerful tools **inside BPMN expressions and scripts**.

1. Install these dependencies also:



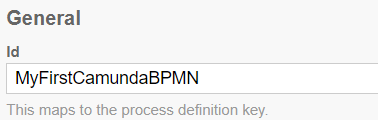
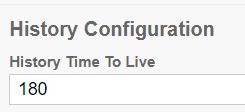
1. Now, in “process.bpmn” in the given folder structure is just a sample. You can create your own using Camunda Modeler and save the file in the same location.

For example: Click on the wrench icon and declare it a SCRIPT TASK. Now, invoking BPMN File form the Controller.

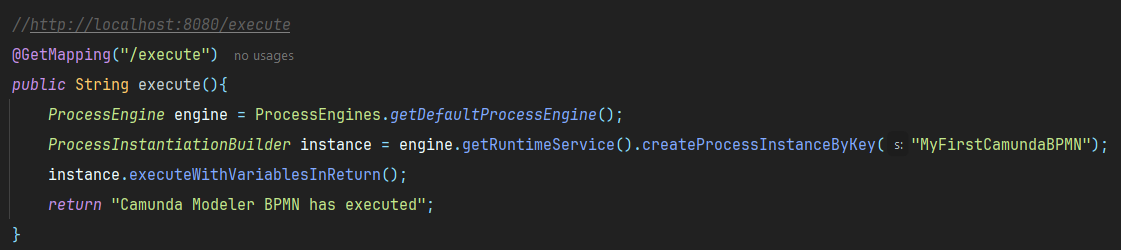


**Controller (Java Code) → BPMN File (Process Flow)**

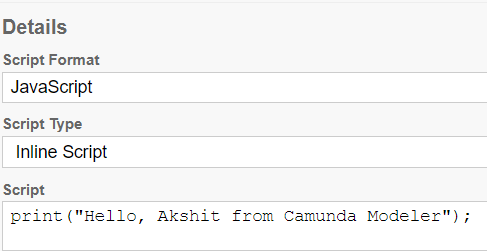
1. **Process Engine**: This is the main part of Camunda that runs and manages the entire process system.
2. **Run Time Service**: This is a service that helps us **start** a new process using its ID or key. We use this from our **controller class** in Java to trigger the process.
3. **Process Instance**: When the process starts, a **new instance** is created. Think of it like starting a new task based on the BPMN file. Each time you start it, a **new copy (instance)** is created.
4. **BPMN**: Finally, the process instance follows the path defined in the **BPMN diagram** (like tasks, gateways, timers, etc.).
5. Click “Outside the MPMN Diagram” and add a ID and also set HTTT [History Time To Live] [**NEW**]

1. Create a controller and add this: [KEY: MyFirstCamundaBPMN]

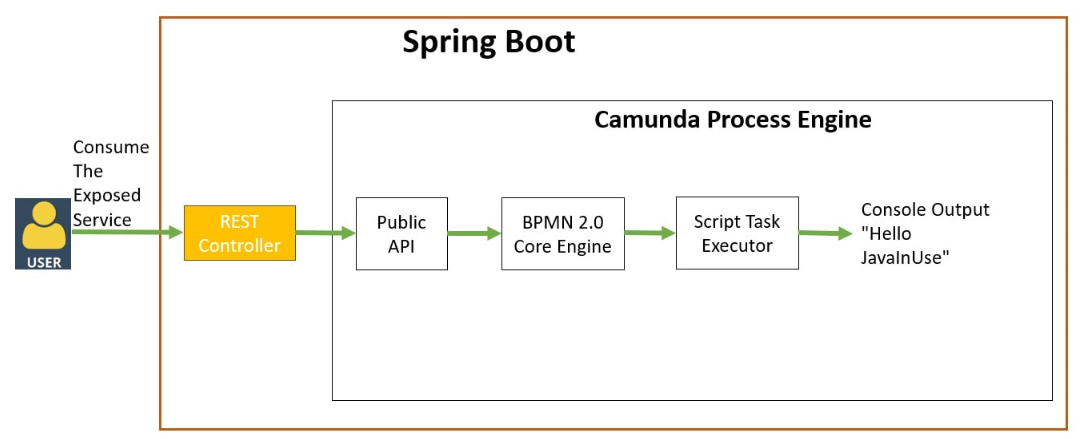


1. Click on the “Print a Message” component of the MPMN diagram and add the following:



1. Hit the API: <http://localhost:8080/execute>
2. Result:





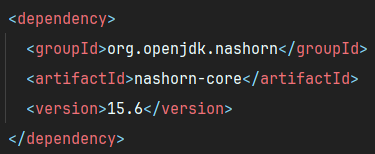
**Q-How JavaScript is running in Java environment?**

Answer: JavaScript runs in a Java environment using the **Java Scripting API (JSR-223)**, which allows Java to execute scripts written in dynamic languages like JavaScript. Camunda uses the scripting engine to evaluate and run the JavaScript during process execution. The engine can access Java objects like execution inside the script, allowing JavaScript to interact with Java code and variables seamlessly. This integration bridges both languages inside the Java Virtual Machine.

For examples:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Engine | Modern JS Support | Speed | Maintenance | Use with Camunda |
| Nashorn | Medium (ES5/6) | Medium | Deprecated | ✅ Easy |
| GraalVM JS | High (ES2020+) | Fast | ✅ Active | ✅ With setup |
| Rhino | Low (ES5) | Slow | ✅ Minimal | ⚠️ Limited |

Note: In earlier Java versions (up to 14), a built-in engine called **Nashorn** handled this. Since Java 15, Nashorn was removed, but it can still be used by adding the **nashorn-core** dependency manually. In tools like **Camunda**, JavaScript is commonly used in script tasks within BPMN models.

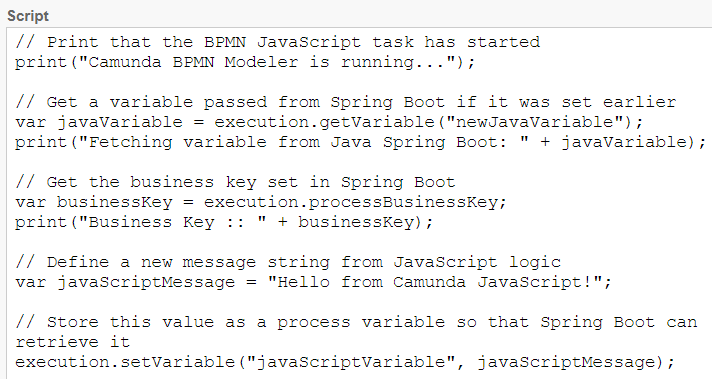


# Program: **myFirstCamundaApp**

**Business Key:** It’s like a **label or tag** you give to a process instance so you can:Search it easily**,** Group related processes **and** connect it with business data (like a customer ID or order number).

**Camunda Modeler 7 and JavaScript**

**In Camunda Modeler 7,** you wrote **JavaScript code inside a Script Task** in the BPMN model like this:



**Execution:** Execution is a Camunda built-in object that lets your script access process data and variables during execution [when the BPMN is running]. So, it is **the running instance of your process**. It's the bridge inside the engine between Java, JavaScript, and BPMN or that is how JavaScript talks to the engine.

**Do you know:** Camunda separates:

1. Temporary JavaScript variables (let, var, const) → only live inside the script
2. Process variables (execution.setVariable(...)) → stored by the Camunda engine, accessible globally

**Q-How Java Talks to Script Task**

Answer: Camunda engine takes care of everything:

1. Java starts a process and sets variables.
2. Script Task executes and uses execution to access/update data
3. Camunda stores the data in memory or DB.
4. Java can read that data back later.

Q- **Why do we use BPMN and JavaScript in Camunda**, when we already have Spring Boot?

Answer: Let us take an example: You’re trying to run a company using just code, no flowcharts, no visual steps. If the **MANAGER** says: "Change step 2 and send email after approval", you need to rewrite code and explain the logic. But **With BPMN + Camunda:** The **MANAGER** **sees the process visually**.They say: “Let’s swap Task B and C.”You change the diagram, not your whole code.

Note: Business logic (like approvals, timers, retries) is handled by Camunda.

Conclusion: We use BPMN + Camunda because it makes **business processes visible, flexible, and manageable** something raw Spring Boot code doesn’t do well.

|  |  |
| --- | --- |
| Spring Boot Only | Spring Boot + Camunda (BPMN) |
| Code is buried in classes/methods | Flow is visible as diagrams (non-tech team can see it) |
| Hard to change process steps | Easy to change order, add conditions, timers visually |
| Developers write all logic | BPMN handles the structure, Java handles key logic |
| No built-in workflow engine | Camunda handles tasks, retries, timers, variables, etc. |

**Q-Why do we use JavaScript in Script Task?**

When a small action is needed (like formatting a message, checking a value, etc.), writing it in a Script Task using JavaScript is faster than creating a full new Java class. So, Use JavaScript for small logic, quick variable manipulation and avoiding over-engineering.

**Full Conclusion**: Camunda acts as a process engine that executes these BPMN diagrams. It can start, monitor, and manage process instances, handle retries, timers, and task assignments automatically, things that are hard to manage manually in pure Spring Boot. JavaScript is used inside **Script Tasks** in BPMN for small logic operations, like setting or checking a variable. It’s quick, lightweight, and doesn't require creating a separate Java class. This keeps the BPMN model self-contained and flexible. For complex tasks like calling APIs, working with databases, or heavy business logic, we still use Spring Boot with Java in **Service Tasks**.

So, in short:

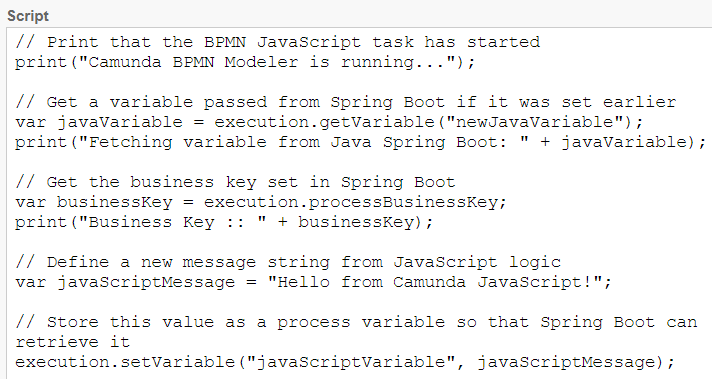
* **BPMN** gives visibility
* **Camunda** manages the flow
* **JavaScript** handles small dynamic parts
* **Java Spring Boot** executes powerful backend logic

This combination makes business automation easy, clear, and maintainable.

**Chapter 03: Script Task**

Task: A single unit of work being performed.

For example: This script is doing [Script Task]



List of different tasks:

|  |  |
| --- | --- |
| Task Type | Purpose |
| Script Task | Executes custom scripts (e.g., JavaScript, Groovy, etc.) inside the process engine. |
| Service Task | Calls external services or Java classes (e.g., via Spring beans). |
| User Task | Waits for human interaction. |
| Manual Task | Represents a manual process not automated. |
| Business Rule Task | Calls decision tables (DMN). |
| Send Task | Sends a message to another process/component. |
| Receive Task | Waits for a message to continue. |
| Call Activity | Calls a subprocess or another BPMN model. |

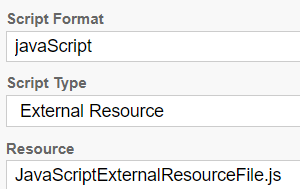
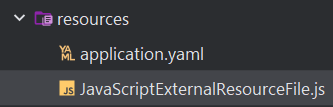
Do you know: Camunda uses **Nashorn JavaScript engine** (in older versions) or **GraalVM** (in newer). It's **not full ECMAScript 6 (ES6)** — so **you can't use const, let, arrow functions, etc.** You should only use **ES5 syntax** to stay safe.

For modern JS (ES6+), use: **Camunda External Tasks** (Java, JS, Python, etc.) Or use **JavaDelegate** with ServiceTask for powerful, testable logic.

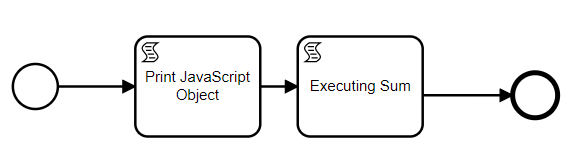
**Q-What is a Script Task?**

Answer: A **Script Task** in Camunda is a task that runs a small code (like JavaScript or Groovy) inside your BPMN workflow, without needing any external service.

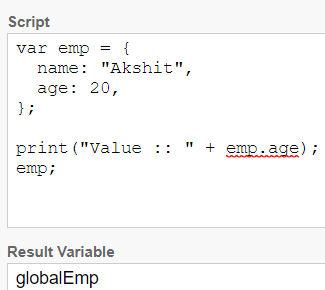
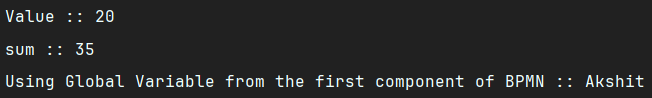
* **"Inline"** means: You write the script **directly** inside the Camunda Modeler (inside the task's properties).
* **"External Resource"** means: You write your script code in a **separate .js file**, and Camunda **reads** and runs it when needed. We use **External Script** for **bigger logic** or **shared code**.

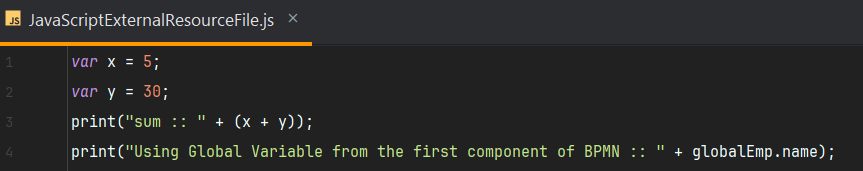
 

Look at this BPMN:



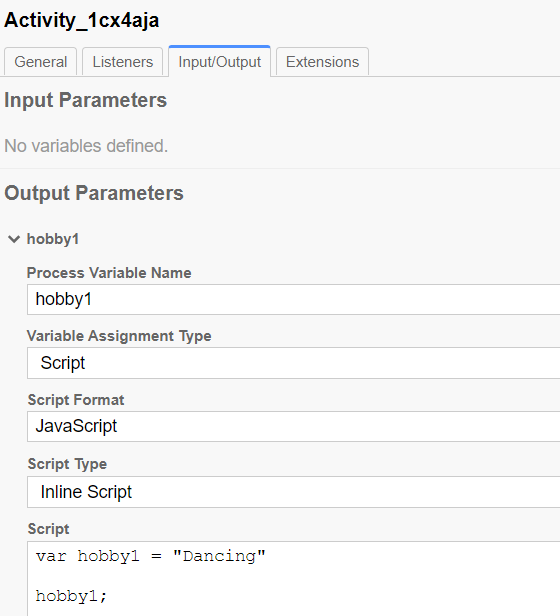
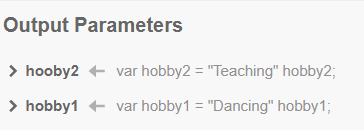
1. There are two components with two different scripts. Now, it is not possible to use variable of first script in the second script. But to do so we need to make the variable as “GLOBAL”. For this do the following:



So, RESULT VARIABLE has a GLOBAL SCORE.

1. To create multiple GLOBAL VARIABLES from one component. Go to input/output tab: [OUTPUT PARAMETERS]

Now, by clicking on “+” sign, you can create multiple.



1. You can do the same using “Execution Object”.





1. INPUT PARAMETERS: The are not globally available, they can only be accessed inside the component.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

**Chapter 04: Service Task**

A **Service Task** in Camunda is used to **invoke or execute business logic automatically**, without human involvement. The actual logic is **defined outside the BPMN diagram**, usually in Java code or another system.

There are **four common ways** to implement a Service Task:

1. **Java Class**: You specify the full class name that implements JavaDelegate.
2. **Delegate Expression**: You use Spring beans like delegateExpression="${myService}", and Camunda will call the bean's execute() method.
3. **Expression:** A simple one-line expression like ${emailService.sendEmail()} to directly call a method.
4. **External Task / REST (with External Task Pattern)**: This is used when another system (outside Camunda) **pulls** tasks using REST API, does the work, and tells Camunda when it’s done.

# **Using Service Task from Java Class**

1. Create a component.

A purple rectangle with black text

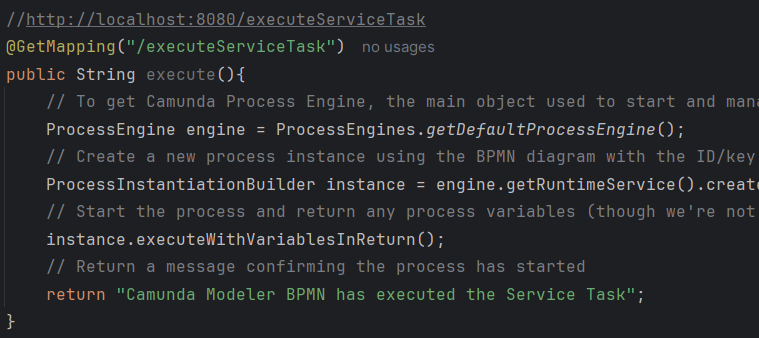
AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

1. Create a package: Delegates and inside it, create a class FetchEmail.

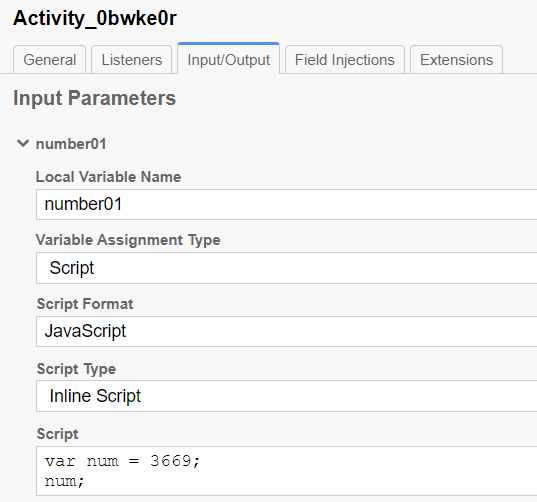


1. Running Service Task Controller



1. Now, passing Global Variables.

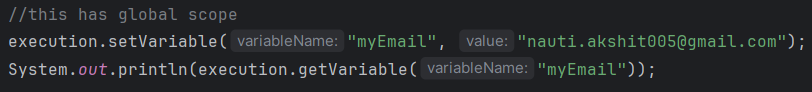
When you create a **Service Task** in **Camunda Modeler**, and assign input parameters (e.g., via "Input Parameters" in the properties panel), those become **process variables** that are available in the context when the task is executed.

A screen shot of a computer code

AI-generated content may be incorrect.

1. Now, setting a variable in my Spring Boot Delegate Class:



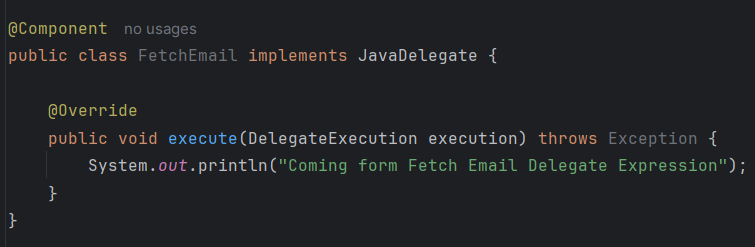
**Chapter 05: Service Task Delegate Expression**

Now, think if me rename the Java Class Name or move it to new package. Then, we need to rename the reference also.

A close up of a web page

AI-generated content may be incorrect.

1. Create a Java Class and implement JavaDelegate interface.
2. Add @Component Annotation



1. Now, go to Camunda Modeler, and write Java Class name in camel letters, starting with small letter.

A close up of a logo

AI-generated content may be incorrect. //FetchEmail is wrong.

**Chapter 06: Service Task Expression Implementation**

In Camunda, a **Service Task** is used to **automate something** — like calling a Java method to do some work (e.g., send an email, calculate tax, etc.).

Normally, we use a JavaDelegate class (which is like a special worker class Camunda understands) to do this. But there's **another way** using something called an **expression**.

With the help of this, we can be able to implement methods from a random java class. Instead of writing a full Java class that implements JavaDelegate, you can just **write a small expression** **(like #{myService.doSomething()})** and Camunda will call that method.

Summary: With Expression Implementation, you don’t need to write special JavaDelegate classes. You can just call **any Spring bean’s method** using a simple expression like **#{someBean.someMethod()}**. It’s shorter and easier for simple tasks.

//Not working but this approach works well with Delegate see 099-Code

**Chapter 07: Connector Implementation**

In **Camunda**, the **Connector Implementation [REST]** allows you to make REST API calls directly from a BPMN process without writing Java code. It is used in **Service Tasks** where you set the implementation type as **"Connector"** and choose a predefined connector like "http-connector" or "rest-connector".

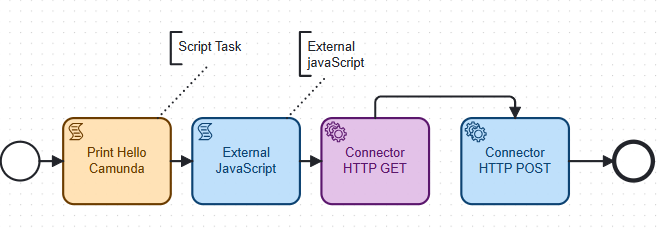
This feature is useful when your process needs to communicate with external systems such as sending data, retrieving information, or triggering actions via APIs. Instead of custom coding, you simply configure the request details like:

* **URL** (the API endpoint)
* **Method** (GET, POST, etc.)
* **Headers** (like Content-Type),
* **Payload/Body** (data to send)
* And map the response if needed.

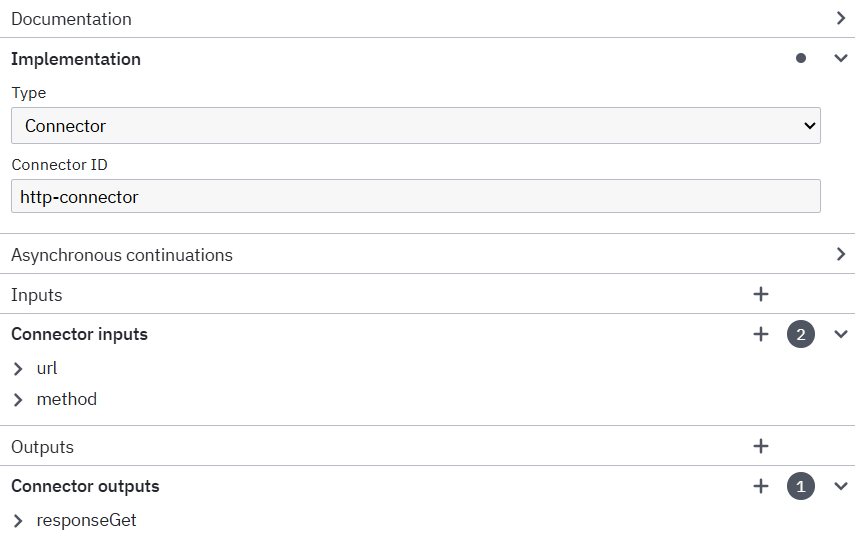
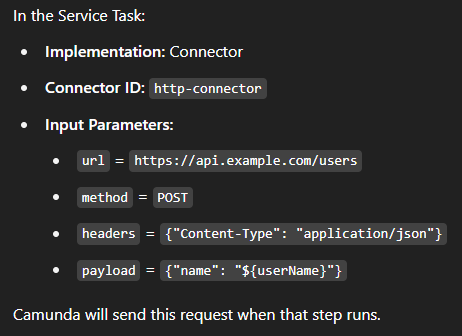
For example, to send user data via a POST request, you provide the API URL, set the method to POST, add necessary headers, and define a JSON payload. Camunda executes the API call when the process reaches that task. This makes Camunda flexible and powerful for integrating with other applications or services in a clean, no-code or low-code way using BPMN models.

Note: This is a free API: <https://api.agify.io/?name=meelad> or this: <https://restful-api.dev/>

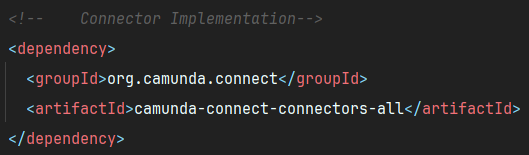
Now, add a new Task make it “Service Task” and choose connector. New, tabs and fields will be added.

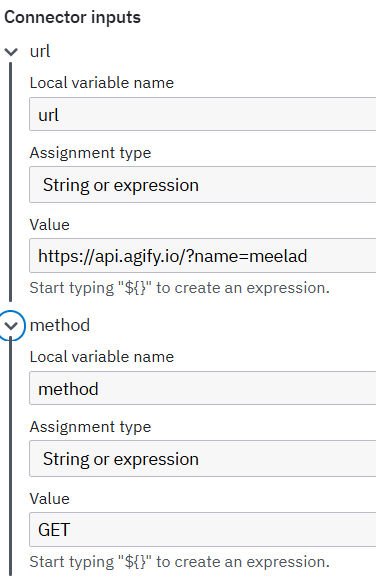
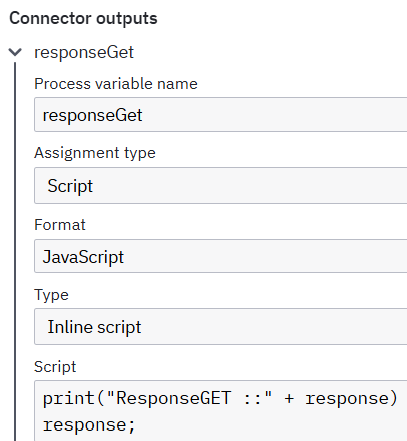


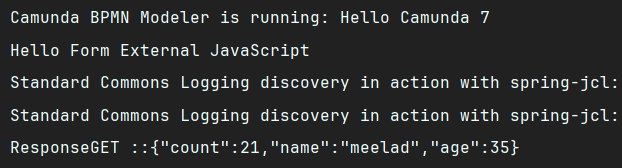
# GET

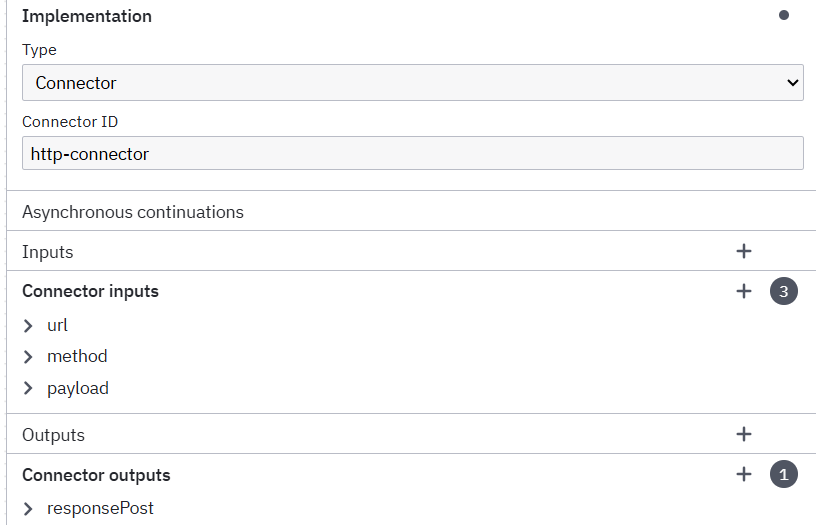
# Add this to POM.xml

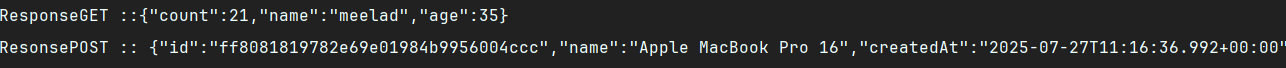


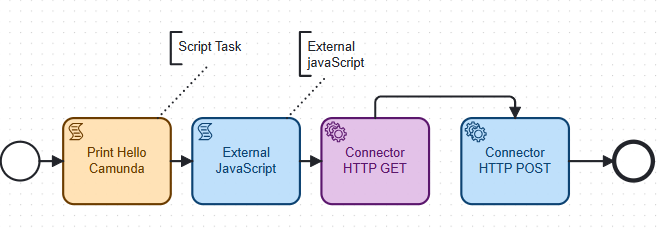


# POST



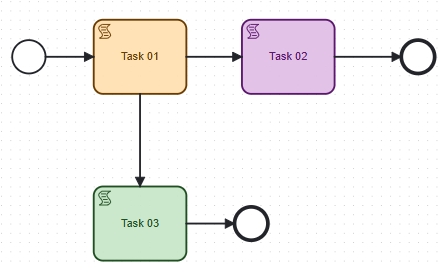
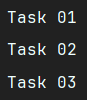


**Chapter 08: Sequence Flow**



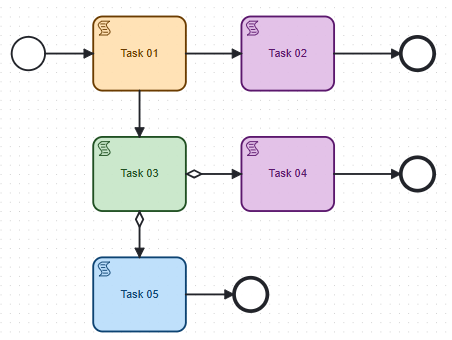
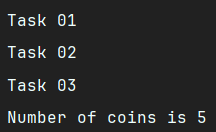
The arrows are connectors between any two elements.

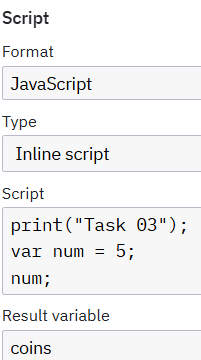
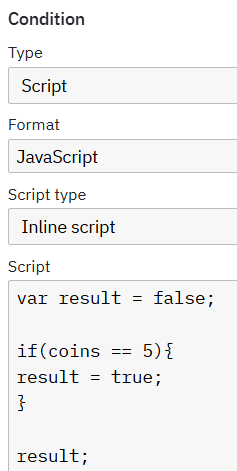
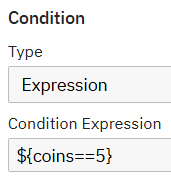
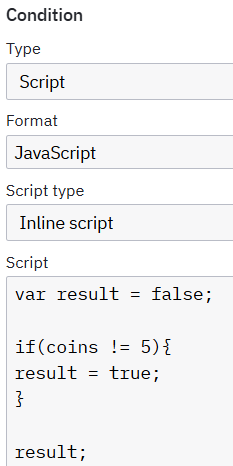
# Outgoing and Incoming Sequence flow: 1 -> 2. For 1 it is outgoing while for 2 it is incoming.

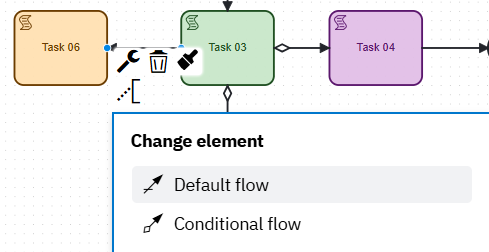
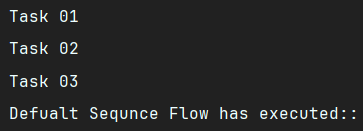
Note: Task 02 and Task 03 will always execute as they are “Non-Conditional Sequence Flow”.

# Conditional Sequence Flow [**See the Arrow**]

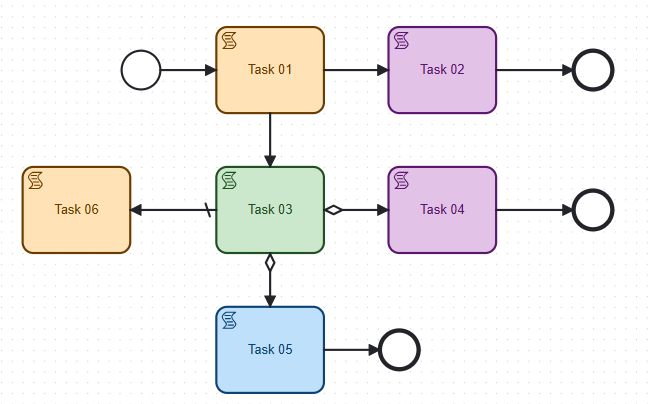
 

  OR  

# **Default Sequence Flow**

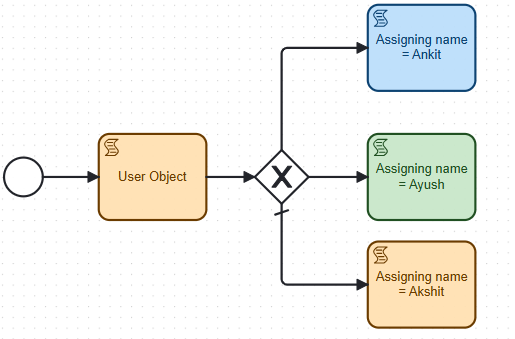
# I have made Task 04 and Task 05 Conditions as FALSE. Like in if-else, we have default condition in similar way, Task 06 has “Default Condition”.



**Chapter 09: Gateway**

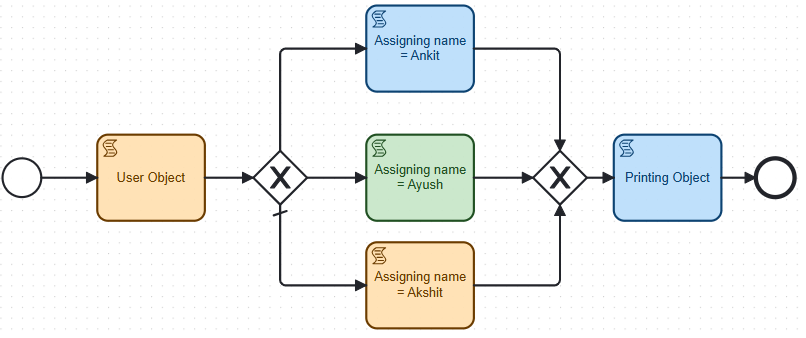
A gateway controls the flow of a process.

# Exclusive Gateway: Only one path of execution is possible.



* If user id == 2 then Ayush is printed else if id == 1 then Ankit will be printed. And if none of the id matches then Akshit will be printed [by default]
* Now, in both, Ankit and Ayush, I assign id == 2 then, whichever is written first in the XML will be executed on matching id == 2.
* If Id == 5, and I remove default condition and then checking conditions which does not exist, then “Runtime Exception” will be thrown. **Therefore, always keep a default sequence flow.**

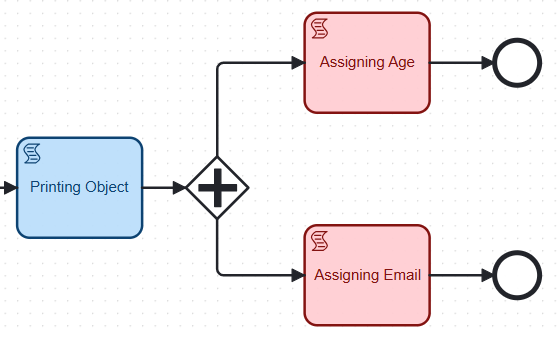
Note: In the above, the exclusive gateway is of DIVERGING TYPE.

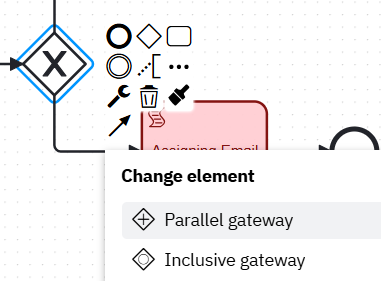
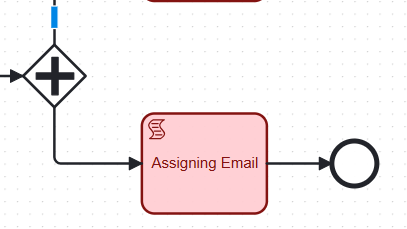
Note: In the above, the exclusive gateway [Right Side] is of CONVERGING TYPE.

**Chapter 10: Parallel Gateway**

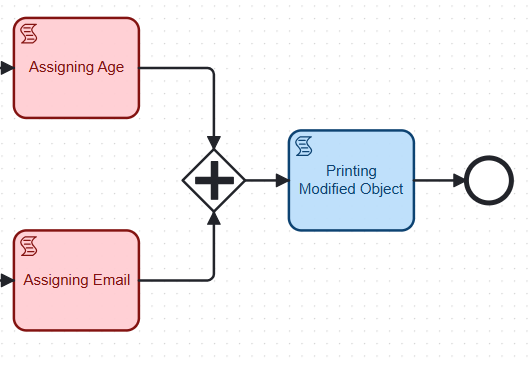
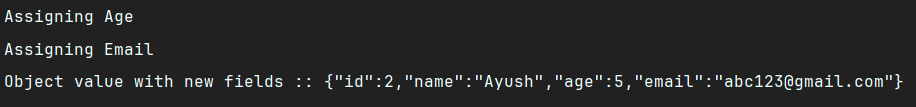
This Gateway is used to bring concurrency in the process model. It is used to execute all the outgoing flows without any condition.



# How to do this:

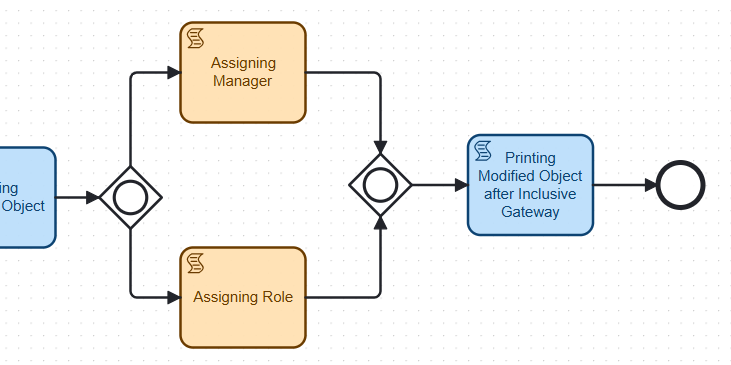
# Converging Parallel Gateway

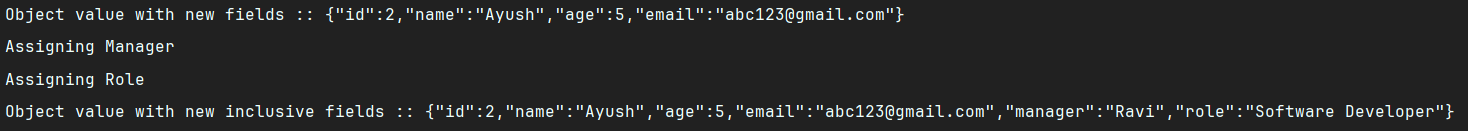
 

Note: Converging Parallel Gateway will wait until both the tasks “Assigning Age as well as Email” does not complete.

**Chapter 11: Inclusive Gateway**

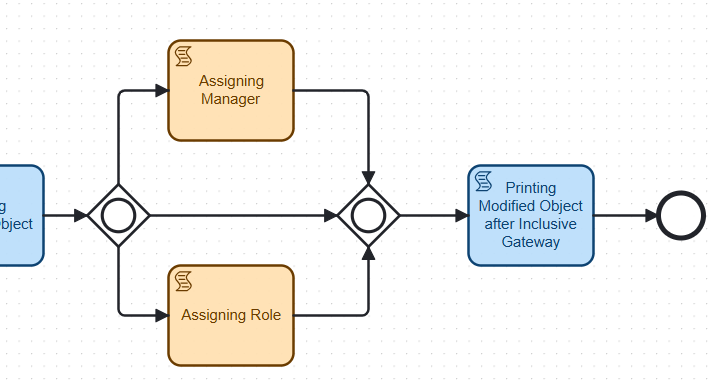
This Gateway is a combination of Exclusive and Parallel Gateways. It can have conditions for outgoing flows which are evaluated. All those outgoing flows whose condition is evaluated as true are executed in parallel.





# Inclusive Default Gateway

This gateway is always evaluated “true”. Suppose both the conditions are evaluated false as a result, the process will be terminated as we get “runtime exception error”.



To handle this, it is a good practice to add a “Inclusive Default Gateway”.

**Chapter 12: Activity: Subprocess Call**

Sub process: It is used to reuse the code and group the code.

Call Activity: It is a sub process which references a process that is external to the process definition. The main use case is to have a reusable process definition that can be called from multiple other process definitions.

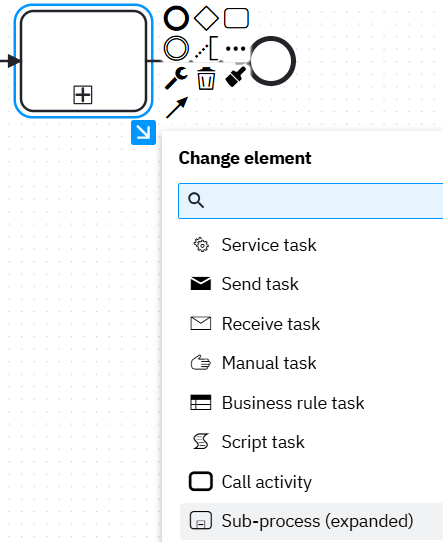
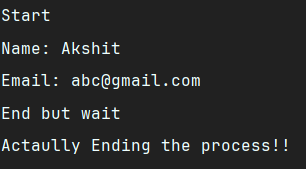
Note: A BPMN can called another BPMN. [Parent and Child BPMN]

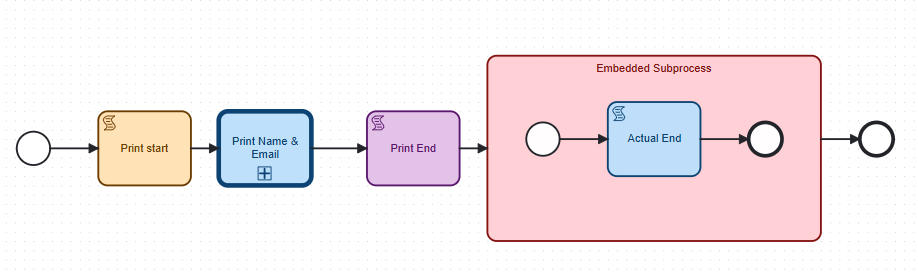
# Sub process BPMN:

**Chapter 13: Embedded Sub process**

It is defined inside the parent process. It contains other tasks, gateways etc.

# First create a sub-process then expand it:



https://www.youtube.com/watch?v=GbVfSWRKyuI&list=PLPySMdlZSMZmQbF8pjX7x6Vpvphunow2t&index=13

# AK Camunda Process Engine

This application [Spring Boot app] is capable of parsing a .bpmn file and extracts information like:

* **Process ID**
* **Process name**
* **All BPMN elements** inside it (e.g., start event, user tasks, service tasks, etc.)

# Requirements

# Actual Package of these maven dependencies:

<dependency>  
 <groupId>org.camunda.bpm.model</groupId>  
 <artifactId>camunda-bpmn-model</artifactId>  
 <version>7.23.0</version>  
</dependency>

# This Camunda Model uses two packages BMPN and XML code inside it:

* <https://github.com/camunda/camunda-bpmn-model>
* <https://github.com/camunda/camunda-xml-model>

# Creating controller and service class:

**Step-by-step when a request is made to /bpmn/parse:**

1. **File Upload (Controller):** The user uploads a .bpmn file using a POST request. Your controller hands it off to the service layer.
2. **Validation + Reading (Service): It** checks if the file is valid (.bpmn, not empty). It reads the input stream from the file.
3. **Parsing BPMN Model: It** uses Bpmn.readModelFromStream(inputStream) from Camunda to parse the XML BPMN file and then it converts it into a BpmnModelInstance, this holds the full structure in memory.
4. **Extracting Elements: The application** extracts all <process> tags and inside those, fetches all <flowElement>s like:

* StartEvent
* EndEvent
* UserTask
* ServiceTask
* SequenceFlow etc.

1. **Returns a Text Summary: It** builds a string describing every element’s **type** and **ID**. It sends that back as a response.

# AK Camunda Expression Evaluator

**Q1-In Camunda, when I write something like ${age > 12} in a gateway, how is it checked? Who checks it?**

Answer: Camunda uses a thing called Expression Language (EL) to check such conditions. So, when you write ${age > 12} or ${age > 12 && height == 20}, Camunda uses EL to check if it's true or false at that moment in the process.

**Q2-Can I create my own “Custom Expression Evaluator”?**

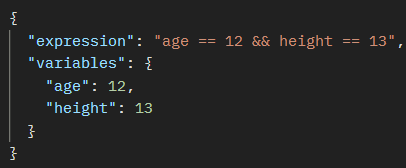
**Answer**: Yes, you can build your own logic checker like Camunda does. There are 3 common ways (tools) used:

1. Apache EL / Jakarta EL: This is the main library Camunda uses to evaluate expressions.
2. JUEL (Java Unified Expression Language): An older way, but still works.
3. Camunda Zeebe (used in Camunda 8): Zeebe is new and doesn’t support ${} style expressions. It uses a different format.

**Q3-I want to create my own Spring Boot service where I send an expression like ${age > 12} and a value, and it should return true or false. How do I do that?**

Answer: You can build your own service using **“Apache EL / Jakarta EL”** (a library which like Camunda uses). This library helps you to read an expression like ${age > 12}, plug in the values (age = 15) and check whether it's true or false.

Now, behind the scenes, these use the **EL APIs** like:

1. ExpressionFactory
2. ELContext [Variable Mapper]
3. ValueExpression

**Q4-How does the flow work behind the scenes?**Answer:

1. **POST API**: <http://localhost:8080/evaluate>
2. Expression
3. Role of EL APIs

|  |  |  |
| --- | --- | --- |
| Step | Class / Part | What it Does |
| 1 | ExpressionFactoryImpl | Creates the engine that understands ${} expressions. |
| 2 | StandardELContext | Stores your variables (like age = 14). It’s the "container" for evaluation. |
| 3 | VariableMapper | Holds each variable (like age) inside the context. |
| 4 | ValueExpression | Represents your expression string like ${age > 12}. |
| 5 | getValue(context) | Finally, this checks if expression is true or false using the values. |