UNIVERSIDAD POPULAR DEL CESAR
ESPECIALIZACION DE INGENIEIRA DE SOFTWARE
GUÍA DE ACTIVIDADES PARA LA CONSTRUCCIÓN DE CONOCIMIENTO

NOMBRE DE LA ASIGNATURA	Patrones De Diseño de Software							
NOMBRE DE LA ACTIVIDAD	Introducción a UML y Diseño Orientado a Objetos							
TIPO DE ACTIVIDAD	Sincrónica		Asincrónica	x	Individual	x	Grupal	

TEMÁTICA REQUERIDA PARA LA ACTIVIDAD	**OBJETIVOS**
Unidad 3. Patrones de diseño creacionales. Unidad 2. Principios de diseño de software Unidad 1. Principios de POO y UML	Evaluar la capacidad para diseñar un API aplicando el patrón de diseño Factory Method, con el propósito de crear una solución escalable, flexible y mantenible.
COMPETENCIAS	**INSUMOS PARA EL DESARROLLO DE LA ACTIVIDAD / REFERENCIAS BIBLIOGRÁFICAS**
• Aplicación de principios de diseño SOLID	• Presentaciones unidad 1, 2 y 3 • Código de ejemplo desarrollado en aula • Bibliografía recomendada

CONOCIMIENTOS PREVIOS REQUERIDOS

Conceptos fundamentales de POO, UML, Principios SOLID, Factory Method Design Pattern

ESPECIFICACIONES DE LA ACTIVIDAD

Caso Práctico: API de Aprovisionamiento de Máquinas Virtuales Multi-Cloud

La empresa necesita que, al aprovisionar una máquina virtual en un proveedor de nube, el sistema también gestione automáticamente la red asociada y el disco de almacenamiento necesarios para que la VM pueda operar. Cada proveedor tiene su propia manera de representar estos recursos, con parámetros específicos.

Los detalles de los nuevos recursos por cada proveedor son los siguientes:

Networking:

- **AWS**: vpcId, subnet, IdsecurityGroup.
- **Azure**: virtualNetwork, subnetName, networkSecurityGroup
- **Google Cloud**: networkName, subnetworkName, firewallTag
- **On-Premise**: physicalInterface,vlanId,firewallPolicy

Storage (Disk):

- **AWS**: volumeType (gp2, io1, etc.), sizeGB,encrypted (boolean)
- **Azure**: diskSku (Standard_LRS, Premium_LRS), sizeGB,managedDisk (boolean)
- **Google Cloud**: diskType (pd-standard, pd-ssd), sizeGB, autoDelete (boolean)

- **On-Premise**: storagePool, sizeGB, raidLevel

La empresa requiere que la solución se mantenga extensible para integrar **nuevos proveedores en el futuro**, sin modificar la lógica central de la API.

1. **Requerimientos Funcionales (RF)**

   - **RF1:** La API debe permitir aprovisionar familias de recursos (VM + Red + Disco) en un único request.

   - **RF2:** El sistema debe exponer un endpoint REST unificado para la provisión de los tres tipos de recurso, recibiendo los parámetros del proveedor y los datos básicos del recurso.

   - **RF3:** El aprovisionamiento de VM debe ser consistente:
     - Una VM de AWS solo puede asociarse con Red y Disco de AWS.
     - Una VM de Azure solo puede asociarse con Red y Disco de Azure.
     - Y así sucesivamente.
   - **RF4:** La API debe devolver un resultado que incluya el recurso creado, con sus datos, incluyendo IDs generados, indicando si el recurso fue creado con éxito, y en ese caso un estado aprovisionado.

   - **RF5**: Debe existir la posibilidad de extender el sistema para nuevos proveedores de nube sin modificar el controlador central.

2. **Requerimientos Funcionales Adicionales (RNF)**

   RNF1 – Consistencia: No se debe crear una VM sin que exista su Red y Disco asociados.

3. **Justificación del Patrón Abstract Factory Method**

   El **Factory Method** permite:

   - **Encapsular la creación** de familias de recursos según el proveedor.
   - Permite **mantener coherencia** de familias de objetos relacionados.
   - Desacoplar el **controlador REST** de las implementaciones concretas de cada proveedor.

4. **Extensiones para los Estudiantes (opcional)**

   - Extender el caso a multi-servicios (Database, Monitoring, Object storage)
   - Incluir **validaciones de parámetros** específicas para cada proveedor.

5. **Entregables**

   - Diagrama de clases de la solución
   - Repositorio con acceso al código implementado
   - Sustentación de la solución

RECOMENDACIONES / OBSERVACIONES	Para el diseño UML del Diagrama de clases se sugiere utilizar cualquiera de las siguientes herramientas: StartUML, PlantUML, Draw.io, GenMyModel o Visual Paradigm. Puede utilizar el lenguaje de programación de su preferencia, preferiblemente Java, es opcional el desarrollo de interfaces Graficas de Usuario.

Elaboro: Ing. Jairo Seoanes, Msc Ingeniería de Sistemas y Computación