

|  |
| --- |
| Informe de Arquitectura |
|  |
| 12 marzo  PREVIRED  Creado por: Christian Carreño Morales |



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Postulante Christian Andrés Carreño Morales  |  |  | | --- | --- | |  | [ch.carreno.m@gmail.com](mailto:ch.carreno.m@gmail.com) | |  | |  | +56 9 67458947 | |  | |  |   Diagramas          Patrones de diseño  **Singleton**: En la implementación 4, se aprecian 2 formas distintas de realizar un patrón Singleton, el Ejemplo2 es del tipo thread-safe inicialización lazy de un check y el Ejemplo1 es de inicialización eager.  **DAO**: En la implementación 2, la clase EmpresaPersona actúa como Data Access Object, extrayendo información de la clase Store (DataSource).  **Factory**: En la implementación 3, la clase Clasificador actúa bajo el patrón Factory, entregando instancias de clases de acuerdo a ciertos criterios.  Errores, falencias & posibles mejoras  Para todos los casos, es recomendable implementar pruebas unitarias y en lo posible, utilizar frameworks que ayuden a cumplir con los estándares mínimos de calidad.  Implementación 1: Con las pruebas unitarias, se hubiesen detectado a tiempo validaciones como dividir por 0, ingreso de caratéres no numéricos, datos obligatorios, largo de valores, control adecuado de excepciones, etc.  En lo posible, desarrollar en base a TDD y principios SOLID.  Implementación 2: Usar otros tipos de Collection, como Set (implementar además hashCode & equals), para este caso, se pueden repetir tanto Personas como Empresa y aunque es un ejemplo, podría asumir que no debe ocurrir ese escenario.  **Implementación 3:** Como el contexto de categorías en base a edad es pequeño, mi sugerencia sería hacer la distinción directamente en el enum:  public enum TipoClasificacionEnum implements ClasificacionPersonaInterface {  *ADOLECENTE*(12,18), *JOVEN*(19,26), *ADULTO*(27,50), *ADULTO\_MAYOR*(51, 100);    private int min, max;    private TipoClasificacionEnum2(int min, int max) {  this.min = min;  this.max = max;  }  @Override  public int getEdadMinima() {  return this.min;  }  @Override  public int getEdadMaxima() {  return this.max;  }  @Override  public TipoClasificacionEnum getTipo() {  return this;  }  @Override  public String getDescripcion() {  return "rango de " + this.name() + " (" + getEdadMinima() + "-" + getEdadMaxima() + " años)";  }  **}**  Y así se reduce el número de implementaciones.  Implementación 4:  Para implementar un Singleton, lo haría si es posible, por medio de la inyección de dependencias de Spring, si no existe otra posibilidad, utilizaría un método Singleton thread-safe de doble check:  public class Ejemplo3 {  private static volatile Ejemplo3 *instance*;  private static Object *lock* = new Object();  private Ejemplo3() {  }  public static Ejemplo3 getInstance() {  Ejemplo3 result = *instance*;  if (result == null) {  synchronized (*lock*) {  result = *instance*;  if (result == null)  *instance* = result = new Ejemplo3();  }  }  return result;  }  }  Implementación 5  Niveles & Principios de Arquitectura:   * Es una Arquitectura por capas. * Utiliza IoC & DI de Spring. * Se puede apreciar que por el uso de interfaces con una sola responsabilidad, se busca estar alineado con los principios SOLID. * Extraño una capa de Negocio, entre el DAO y el Controller. * Se trabajan las entidades como Dominios (Domain Driven Development). * Claramente al bosquejo le faltan anotaciones y detalles, pero se entiende a simple vista la idea.   Implementación 6  Diagrama 1 |
| Podría interpretarse de que existen 4 aplicaciones ejecutándose en servidor de aplicaciones (no clusterizado) con un único DataSource, por tanto, no tiene mucho sentido el Balanceador. |
| Diagrama 2 |
| Podría interpretarse de que existen 4 aplicaciones ejecutándose 2 servidores (no clusterizados) con una única Instancia por Aplicativo, por tanto, tampoco tiene mucho sentido el Balanceador. En este ejemplo, cada aplicación maneja su propio DataSource.  Diagrama 2  Event driven Architecture: es un patrón de arquitectura software que promueve la producción, detección, consumo y reacción a eventos. |