PROGRAMACIÓN ORIENTADA A OBJETOS

CON JAVA 17

ACADEMIA CONSULTEC

www.consultec-ti.com

VALENTÍA | AGILIDAD | EVOLUCIÓN | OPTIMIZACIÓN

Agenda **del día**

5

- Patrones de Diseño
- Testing con JUNIT
- Gestión avanzada de logging (práctica)
- Introducción a Spring Boot (práctica)

ACADEMIA CONSULTEC

Patrones de **Diseño**

ACADEMIA CONSULTEC

Patrones de Diseño

Es una solución reutilizable para un problema que ocurre dentro de un contexto de programación dado. ¿Qué significa eso? Pues que, en ocasiones, los programadores encuentran el mismo problema varias veces en distintos proyectos. Por eso, en vez de que cada uno aporte o diseñe su propia solución, se crean los patrones de diseño en Java.

Los patrones de diseño en Java son, en definitiva, soluciones a problemas recurrentes y que se ha documentado que funcionan y los resuelven.

ACADEMIA CONSULTEC

Tipos de Patrones de Diseño

Los patrones de diseño más utilizados se clasifican en tres categorías principales, cada patrón de diseño individual conforma un total de 23 patrones de diseño. Las cuatro categorías principales son:

Patrones Creacionales

Los patrones de creación proporcionan diversos mecanismos de creación de objetos, que aumentan la flexibilidad y la reutilización del código existente de una manera adecuada a la situación. Esto le da al programa más flexibilidad para decidir qué objetos deben crearse para un caso de uso dado.

Patrones Estructurales

Facilitan soluciones y estándares eficientes con respecto a las composiciones de clase y las estructuras de objetos. El concepto de herencia se utiliza para componer interfaces y definir formas de componer objetos para obtener nuevas funcionalidades.

Tipos de Patrones de Diseño

Patrones de Comportamiento

El patrón de comportamiento se ocupa de la comunicación entre objetos de clase. Se utilizan para detectar la presencia de patrones de comunicación ya presentes y pueden manipular estos patrones.

Estos patrones de diseño están específicamente relacionados con la comunicación entre objetos.

ACADEMIA CONSULTEC

Patrones Creacionales

Factory Method

Proporciona una interfaz para la creación de objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

Abstract Factory

Permite producir familias de objetos relacionados sin especificar sus clases concretas.

Builder

Permite construir objetos complejos paso a paso. Este patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.

Patrones Creacionales

Prototype

Permite copiar objetos existentes sin que el código dependa de sus clases.

Singleton

Permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

Patrones estructurales

Adapter

Permite la colaboración entre objetos con interfaces incompatibles.

Bridge

Permite dividir una clase grande o un grupo de clases estrechamente relacionadas, en dos jerarquías separadas (abstracción e implementación) que pueden desarrollarse independientemente la una de la otra.

Composite

Permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.

Proxy

Permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.

Patrones estructurales

Decorator

Permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.

Facade

Proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.

Flyweight

Permite mantener más objetos dentro de la cantidad disponible de memoria RAM compartiendo las partes comunes del estado entre varios objetos en lugar de mantener toda la información en cada objeto.

Patrones de comportamiento

Chain of Responsibility

Permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.

Command

Convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación te permite parametrizar los métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y soportar operaciones que no se pueden realizar.

Iterator

Permite recorrer elementos de una colección sin exponer su representación subyacente (lista, pila, árbol, etc.).

Patrones de comportamiento

Mediator

Permite reducir las dependencias caóticas entre objetos. El patrón restringe las comunicaciones directas entre los objetos, forzándolos a colaborar únicamente a través de un objeto mediador.

Memento

Permite guardar y restaurar el estado previo de un objeto sin revelar los detalles de su implementación.

Observer

Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.

Patrones de comportamiento

State

Permite a un objeto alterar su comportamiento cuando su estado interno cambia. Parece como si el objeto cambiara su clase.

Strategy

Permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables.

Template Method

Define el esqueleto de un algoritmo en la superclase pero permite que las subclases sobre-escriban pasos del algoritmo sin cambiar su estructura.

Visitor

Duplica el código de recorrido en cada objeto de tipo compuesto. Sólo se utiliza para implementar recorridos complejos que dependen de los resultados de las operaciones.

Almuerzo

45 min

Testing

ACADEMIA CONSULTEC

Testing

El testing o prueba de código hace referencia a los procesos de validación del funcionamiento del software de un determinado programa o aplicación. Este mecanismo se caracteriza por proporcionar una garantía de calidad del sistema, para lo que utiliza recursos como las pruebas unitarias de software, que se encargan de comprobar que un fragmento del código fuente esté funcionando de forma adecuada.

Por tanto, las pruebas unitarias de software pueden resultar de gran ayuda para asegurar el funcionamiento de tu código, por lo que es necesario que se aprenda todos los detalles al respecto, incluyendo sus principales características, utilidades y propiedades

Pruebas Unitarias

Las pruebas unitarias de software, conocidas también como unit testing o test unitarios, pueden definirse como un mecanismo de comprobación del funcionamiento de las unidades de menor tamaño de un programa o aplicación en específico.

Estas pruebas unitarias de software forman parte de la estrategia de metodología ágil del trabajo del desarrollo, donde se busca ofrecer piezas pequeñas de software en funcionamiento en un corto periodo de tiempo, con el objetivo de aumentar la satisfacción del cliente.

Características del UnitTesting

Las pruebas unitarias de software, también conocidas como unit testing, incluyen un conjunto de características y propiedades que permiten su funcionamiento, como. por ejemplo, que se realiza a través de la escritura de fragmentos del código fuente de una aplicación o programa para que se prueben las unidades de este código.

- Se encuentran funcionando de la forma que deberían e independientemente
- Funcionan mediante el mecanismo de aislar una parte determinada del código fuente
- Usualmente se llevan a cabo como primera evaluación en la fase del desarrollo
- Demuestran que no existe ningún tipo de error en el código fuente

Ventajas

- Demostrar que la lógica del código fuente de un programa o aplicación se encuentre en buen estado.
- permite el aumento de la legibilidad del código.
- Su ejecución se debe de llevar solo algunos milisegundos.

ACADEMIA CONSULTEC

Tipos de UnitTesting

Si hablamos solo de pruebas de software, hay muchos tipos, y las pruebas unitarias son una de ellas. Las pruebas unitarias se dividen además en dos tipos.

Prueba manual

En las pruebas unitarias manuales, un desarrollador escribe código para probar una sección en particular interactuando con las API o el software en persona para detectar los errores. Esta es una tarea un poco costosa y que requiere mucho tiempo, ya que requiere que alguien trabaje en dicho entorno para probar los componentes individuales del software. Esto puede conducir a un error humano, como errores tipográficos, omisión de pasos y más.

Pruebas automatizadas

La máquina realiza la misma tarea de prueba unitaria y ejecuta el script de prueba previamente escrito. Con las pruebas unitarias automatizadas, puede probar una sola secuencia o una secuencia compleja que produzca el mismo resultado.

Evaluaciones en UnitTesting

ACADEMIA CONSULTEC

Herramientas para UnitTest

- **JUnit**: es un marco de prueba de código abierto para pruebas unitarias que ayuda a los desarrolladores de Java a escribir y ejecutar pruebas repetibles. Funciona de la misma manera que NUnit.

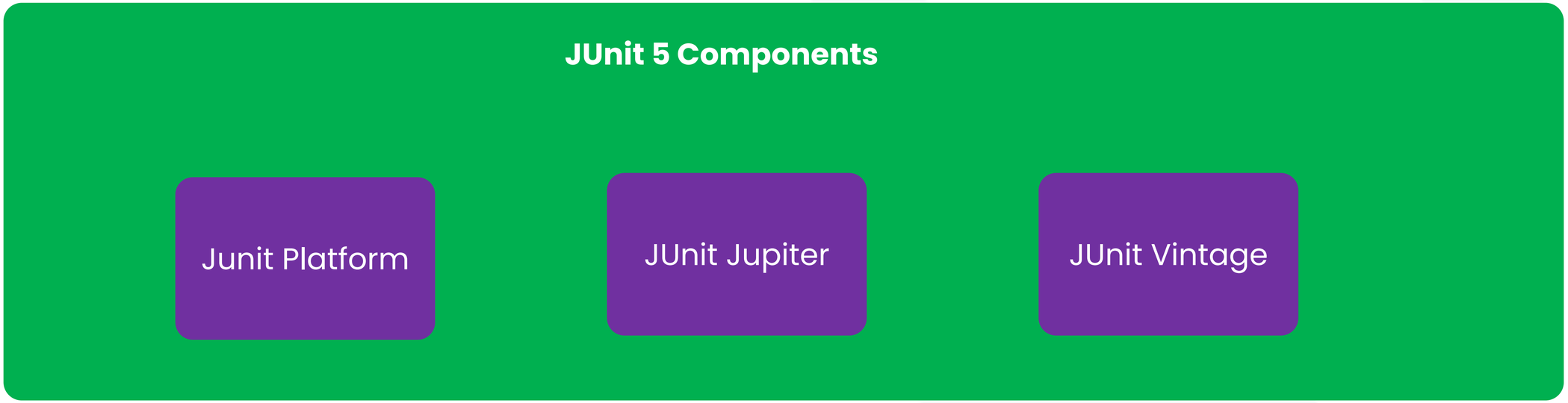
- **PruebaNG**: Es nuevamente un marco de prueba especialmente inspirado en NUnit y JUnit. Encontrarás algunas funcionalidades añadidas. Además, admite pruebas parametrizadas y basadas en datos.

- **jprueba**: Jtest es desarrollado por Parasoft y se usa especialmente para probar aplicaciones de software Java. Además, admite el análisis de código estático y asegura una codificación sin defectos durante todo el proceso de desarrollo de software.

- **EMMA**: Es un conjunto de herramientas gratuito y de código abierto para medir y analizar la cobertura del código Java. Obtendrá soporte para el desarrollo de software a gran escala mientras maneja el trabajo individual de forma iterativa y rápida.

Testeando con JUnit

En el mundo de Java, JUnit es uno de los marcos populares utilizados para implementar pruebas unitarias contra el código de Java. JUnit ayuda principalmente a los desarrolladores a probar su código en la JVM por sí mismos.



ACADEMIA CONSULTEC

Arquitectura JUnit

Plataforma JUnit

- Lanza marcos de prueba en la JVM.
- Se ha utilizado la API de TestEngine para crear un marco de prueba que se ejecuta en la plataforma Junit.

JUnit Júpiter

- Combinación de un nuevo modelo de programación para escribir pruebas y un modelo de extensión para extensions.
- Adición de nuevas anotaciones como @BeforeEach, @AfterEach, @AfterAll, @BeforeAll etc.< /li>

JUnit Vintage

- Proporciona soporte para ejecutar pruebas anteriores de JUnit versión 3 y 4 en esta nueva Plataforma.

ACADEMIA CONSULTEC

Dependencias de JUnit Maven

Para implementar

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-runner</artifactId>
    <version>1.1.1</version>
    <scope>test</scope>
  </dependency>
```

Ejecutar las pruebas unitarias donde IDE no tiene soporte JUnit5

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.19.1</version>
  <dependencies>
    <dependency>
      <groupId>org.junit.platform</groupId>
      <artifactId>junit-platform-surefire-provider</artifactId>
      <version>1.0.2</version>
    </dependency>
  </dependencies>
</plugin>
```

ACADEMIA CONSULTEC

Anotaciones JUnit

Annotation	Description
@Test	Indica un método de prueba
@DisplayName	Declara un nombre de visualización personalizado para la clase de prueba o el método de prueba
@BeforeEach	Indica que el método anotado debe ejecutarse antes de cada método de prueba
@AfterEach	Indica que el método anotado debe ejecutarse después de cada método de prueba
@BeforeAll	Indica que el método anotado debe ejecutarse antes que todos los métodos de prueba
@AfterAll	Indica que el método anotado debe ejecutarse después de todos los métodos de prueba
@Disable	Se utiliza para desactivar una clase o método de prueba
@Nested	Indica que la clase anotada es una clase de prueba anidada y no estática
@Tag	Declarar etiquetas para filtrar las pruebas
@ExtendWith	Registrar extensiones personalizadas

Afirmaciones JUnit

Assertion	Description
assertEquals(expected, actual)	Falla cuando lo esperado no es igual a lo real
assertFalse(expression)	Falla cuando la expresión no es falsa
assertNull(actual)	Falla cuando actual no es null
assertNotNull(actual)	Falla cuando actual es null
assertAll()	Agrupa muchas aserciones y cada aserción se ejecuta incluso si una o más de ellas fallan
assertTrue(expression)	Falla si la expresión no es verdadera
assertThrows()	Se espera que la clase a probar lance una excepción
Assertion	Descripcion
assertEquals(expected, actual)	Falla cuando lo esperado no es igual a lo real
assertFalse(expression)	Falla cuando la expresión no es falsa

ACADEMIA CONSULTEC

Gracias

¡Nos vemos pronto!