CONCURRENCIA EN JAVA

CATALINA MULFORD MONROY

P1

PEDRO FELIPE GÓMEZ BONILLA

CAMPUSLANDS
SPUTNIK
RUTA JAVA
FLORIDABLANCA
13 de agosto del 2024

Tabla de Contenidos

Introducción

En el siguiente informe se encontrará la investigación realizada acerca de “abstract factory” y “concurrencia”, enfocándose en esta última. Se encontrarán una corta descripción de cada uno junto a sus respectivos ejemplos.

Concurrencia

Cuando se habla de concurrencia se hace referencia a la posibilidad de desarrollar múltiples tareas en el mismo tiempo, permitiendo la eficiencia de los programas. La concurrencia posee dos conceptos básicos se utilizan para poder aprovechar todos los recursos posibles, estos conceptos son:

- **Procesos:** Son programas que se ejecutan en su propio espacio de memoria y se encuentran separados de otros procesos.

- **Hilos (thread):** Es el paso a paso de cómo se ejecutará el proceso, cada proceso tiene al menos un hilo, llamado hilo principal. A diferencia de los procesos, los hilos están conectados entre sí, ya que comparten los recursos.

Tanto los procesos como los hilos son ejecutados por los procesadores CPU, siendo así esencial para el funcionamiento de la concurrencia. A través de está es posible que Java permitá ejecutar operaciones sean asincrónicas y paralelas, lo que puede resultar en una mejora significativa del rendimiento.

Lo primero al momento de crear una concurrencia es la creación del hilo donde se pondrá la tarea, o proceso, que se va a conectar. Esto se puede ver en el siguiente ejemplo:

```
public class ThreadRunnable {
	private static final Instant INICIO = Instant.now();
	public static void main(String[] args) {
		Runnable tarea = () -> {
			try {
				Log("Empieza la tarea");
				Thread.sleep(5000);
				Log("Termina la tarea");
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		};
		Thread hilo = new Thread(tarea);
		hilo.start();
		try {
			Log("Se empieza a esperar al hilo");
			hilo.join(3000);
```

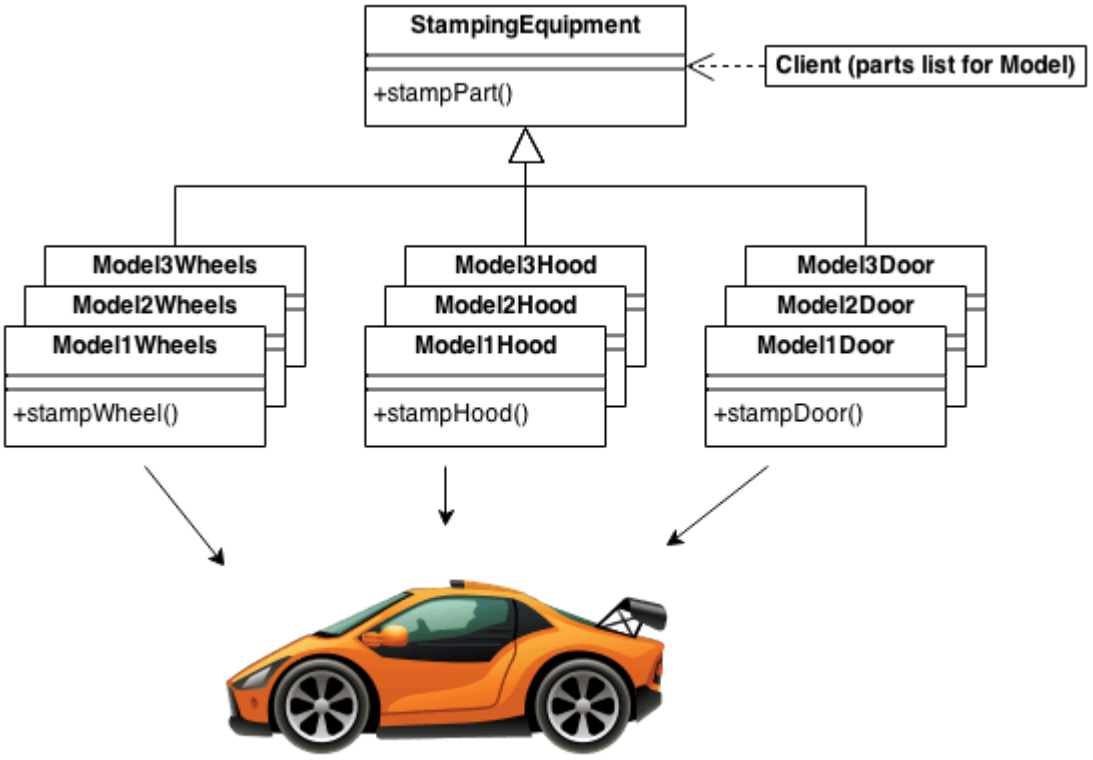
```
                Log("Se termina de esperar al hilo");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
      }
      private static void Log(Object mensaje) {
            System.out.println(String.format("%s [%s] %s",
                Duration.between(INICIO, Instant.now()),
Thread.currentThread().getName(), mensaje.toString()));
      }
}
```

Instalación General

Nos permite producir familias de objetos relacionados sin especificar sus clases concretas. Adicionalmente, se tiene la posibilidad de poder expandir el código sin afectar el ya existente.

Lo que se hará es declarar de forma explícita interfaces para cada producto diferente de la familia de productos. Después podemos hacer que todas las variantes de los productos sigan esas interfaces.

La interfaz fábrica abstracta declara un grupo de métodos de creación que el código cliente puede utilizar para producir distintos tipos de elementos. Las fábricas concretas coinciden con sistemas operativos específicos y crean los elementos UI correspondientes.



Este patrón ofrece una interfaz para crear un objeto a partir de una clase general. También funciona con varias clases de productos que estén relacionados. Un ejemplo del uso de este patrón es el siguiente:

```
interface GUIFactory is
    method createButton():Button
    method createCheckbox():Checkbox
class WinFactory implements GUIFactory is
    method createButton():Button is
        return new WinButton()
    method createCheckbox():Checkbox is
        return new WinCheckbox()
class MacFactory implements GUIFactory is
    method createButton():Button is
        return new MacButton()
    method createCheckbox():Checkbox is
        return new MacCheckbox()
interface Button is
    method paint()
class WinButton implements Button is
    method paint() is
class MacButton implements Button is
```

```
    method paint() is
interface Checkbox is
    method paint()
class WinCheckbox implements Checkbox is
    method paint() is
class MacCheckbox implements Checkbox is
    method paint() is
class Application is
    private field factory: GUIFactory
    private field button: Button
    constructor Application(factory: GUIFactory) is
        this.factory = factory
    method createUI() is
        this.button = factory.createButton()
    method paint() is
        button.paint()
class ApplicationConfigurator is
    method main() is
        config = readApplicationConfigFile()
        if (config.OS == "Windows") then
            factory = new WinFactory()
        else if (config.OS == "Mac") then
            factory = new MacFactory()
        else
            throw new Exception("Error! Unknown operating system.")
        Application app = new Application(factory)
```

Referencias

Diego Díaz Rodríguez. (2019). *Introducción a la Concurrencia en Java (I)*. The Softtek Blog. Obtenido el 13 de agosto del 2024 de https://blog.softtek.com/es/java-concurrency

(2023). *Explorando la Concurrencia en Java: Maximizar Rendimiento 2023.* BacaSoftware. Obtenido el 13 de agosto del 2024 de https://www.bacasoftware.com/explorando-la-concurrency-en-java-maximizar-rendimiento-2023/#:~:text=La%20concurrencia%20en%20Java%20permite,los%20recursos%20de%20la%20CPU.

Abstract Factory. Refactoring.Guru. Obtenido el 13 de agosto del 2024 de https://refactoring.guru/es/design-patterns/abstract-factory

Abstract Factory Design Pattern. SourceMaking. Obtenido el 13 de agosto del 2024 de https://sourcemaking.com/design_patterns/abstract_factory