Código con Comentarios Explicativos

Clase `InformeFinancieroTriemestre2`

java

```
package es.pildorasIoC.AnnotationsConfiguration;

import org.springframework.stereotype.Component;

// Marca esta clase como un bean de Spring
@Component
public class InformeFinancieroTriemestre2 implements CreacionInformeFinanciero {

    // Implementa el método de la interfaz para devolver un informe
    @Override
    public String getInformeFinaciero() {
        return "Presentación de Informe Extremadamente exitos del trimestre 2
(AnnotationsQualifier)";
    }
}
```

Clase `ConfiguracionBeans`

java

```
package es.pildorasIoC.AnnotationsConfiguration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

// Indica que esta es una clase de configuración de Spring
@Configuration
// Escanea el paquete especificado en busca de componentes
@ComponentScan("es.pildorasIoC.AnnotationsConfiguration")
public class ConfiguracionBeans {

    // Define un bean de tipo CreacionInformeFinanciero con ID "informeDtoCompras"
    @Bean
    public CreacionInformeFinanciero informeDtoCompras() {
        return new InformeFinancieroDtoCompras();
    }

    // Define un bean de tipo Empleados con ID "directorFinanciero" e inyecta el bean
"informeDtoCompras"
    @Bean
    public Empleados directorFinanciero() {
        return new DirectorFinanciero(informeDtoCompras());
    }
}
```

Interfaz CreacionInformeFinanciero

java

```java
package es.pildorasIoC.AnnotationsConfiguration;

import org.springframework.stereotype.Component;

// Marca esta interfaz como un componente de Spring
@Component
public interface CreacionInformeFinanciero {

    // Método para obtener el informe financiero
    public String getInformeFinaciero();
}
```

Interfaz Empleados

java

```java
package es.pildorasIoC.AnnotationsConfiguration;

public interface Empleados {

    // Método para obtener las tareas de un empleado
    public String getTareas();

    // Método para obtener los informes de un empleado
    public String getInformes();
}
```

Clase InformeFinacieroTrimestre1

java

```java
package es.pildorasIoC.AnnotationsConfiguration;

import org.springframework.stereotype.Component;

// Marca esta clase como un bean de Spring
@Component
public class InformeFinacieroTrimestre1 implements CreacionInformeFinanciero {

    // Implementa el método de la interfaz para devolver un informe
    @Override
    public String getInformeFinaciero() {
        return "Presentación del informe financiero trimeste 1";
    }
}
```

Clase InformeFinamcieroTrimentre3

java

```java
package es.pildorasIoC.AnnotationsConfiguration;

import org.springframework.stereotype.Component;

// Marca esta clase como un bean de Spring
@Component
public class InformeFinamcieroTrimentre3 implements CreacionInformeFinanciero {

    // Implementa el método de la interfaz para devolver un informe
    @Override
    public String getInformeFinaciero() {
        return "Presentacion de Iforme de Gastos operativos trimestre 3";
    }
}
```

Clase *InformeFinancieroDtoCompras*

java

```java
package es.pildorasIoC.AnnotationsConfiguration;

public class InformeFinancieroDtoCompras implements CreacionInformeFinanciero {

    // Implementa el método de la interfaz para devolver un informe
    @Override
    public String getInformeFinaciero() {
        return "Informe financiero del departamento de compras 2024";
    }
}
```

Clase *DirectorFinanciero*

java

```java
package es.pildorasIoC.AnnotationsConfiguration;

public class DirectorFinanciero implements Empleados {

    // Dependencia inyectada
    private CreacionInformeFinanciero informeFinancieroCompras;

    // Constructor que recibe la dependencia
    public DirectorFinanciero(CreacionInformeFinanciero informeFinanciero) {
        this.informeFinancieroCompras = informeFinanciero;
    }

    // Implementa el método de la interfaz para devolver las tareas
    @Override
    public String getTareas() {
        return "Gestion de las operaciónes financieras de la empresa";
    }

    // Implementa el método de la interfaz para devolver los informes
    @Override
    public String getInformes() {
        return this.informeFinancieroCompras.getInformeFinaciero();
    }
}
```

Clase Principal *MainUsoAnnotationAnotacionConfiguration*

java

```
package es.pildorasIoC.AnnotationsConfiguration;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainUsoAnnotationAnotacionConfiguration {

    public static void main(String[] args) {
        // Crea el contexto de aplicación basado en la configuración de Java
        AnnotationConfigApplicationContext contextoAnnotation = new
AnnotationConfigApplicationContext(ConfiguracionBeans.class);

        // Solicita el bean "comercialExperimentadoAnotacionConfiguration" del contexto
        Empleados antonio =
contextoAnnotation.getBean("comercialExperimentadoAnotacionConfiguration", Empleados.class);
        Empleados lucia =
contextoAnnotation.getBean("comercialExperimentadoAnotacionConfiguration", Empleados.class);

        // Solicita el bean "directorFinanciero" del contexto
        Empleados pedro = contextoAnnotation.getBean("directorFinanciero", Empleados.class);

        // Verifica si los beans son singleton (apuntan a la misma dirección de memoria)
        if (antonio.hashCode() == lucia.hashCode()) {
            System.out.println("*********** Ejemplo de Patron Singleton  *************** :");
            System.out.println("Apuntan al la misma dirección de memoria");
            System.out.println();
            System.out.println("Antonio: Dirección de memoria ==> " + antonio);
            System.out.println("Lucia: Dirección de memoria ==> " + lucia);
            System.out.println();
            System.out.println("Antonio: hashCode ==> " + antonio.hashCode());
            System.out.println("Lucia   hashCode ==> " + lucia.hashCode());
            System.out.println();
            System.out.println("Pedro: Informe Compras ==> " + pedro.getInformes());
        }

        // Verifica si los beans son prototype (no apuntan a la misma dirección de memoria)
        if (antonio.hashCode() != lucia.hashCode()) {
            System.out.println("*********** Ejemplo de Patron Prototype ************** :");
            System.out.println("No apuntan al la misma dirección de memoria");
            System.out.println();
            System.out.println("Antonio Dirección de memoria ==> " + antonio);
            System.out.println("Lucia Dirección de memoria ==> " + lucia);
            System.out.println();
            System.out.println("Antonio hashCode ==> " + antonio.hashCode());
            System.out.println("Lucia hashCode  ==> " + lucia.hashCode());
        }

        // Cierra el contexto de Spring
        contextoAnnotation.close();
    }
}
```

Explicación Sencilla del Código

1. **Componentes y Beans**:
   - Clases como `InformeFinancieroTriemestre2`, `InformeFinacieroTrimestre1`, `InformeFinamcieroTrimentre3` están anotadas con `@Component`, indicando a Spring que las registre como beans.
   - La clase `DirectorFinanciero` y `InformeFinancieroDtoCompras` implementan la lógica de negocio y dependen de la interfaz `CreacionInformeFinanciero`.
2. **Configuración de Spring**:
   - La clase `ConfiguracionBeans` está anotada con `@Configuration` y `@ComponentScan`, permitiendo que Spring busque y registre automáticamente los componentes en el paquete `es.pildorasIoC.AnnotationsConfiguration`.
   - Los métodos anotados con `@Bean` en `ConfiguracionBeans` definen beans adicionales y sus dependencias.
3. **Contexto de Aplicación**:
   - En `MainUsoAnnotationAnotacionConfiguration`, el contexto de Spring se inicializa utilizando `AnnotationConfigApplicationContext`.
   - Se solicitan varios beans desde el contexto y se comprueba si son singleton o prototype, mostrando información al respecto.

Flujo de Ejecución Detallado

1. **Inicio de la Aplicación**:
   - La ejecución comienza en el método `main` de `MainUsoAnnotationAnotacionConfiguration`.
2. **Creación del Contexto de Aplicación**:
   - Se crea `contextoAnnotation` como una instancia de `AnnotationConfigApplicationContext`, pasando `ConfiguracionBeans.class` como argumento. Esto carga la configuración de Spring y escanea los componentes.
3. **Registro de Beans**:
   - Spring registra los beans definidos en la configuración (`informeDtoCompras`, `directorFinanciero`) y los componentes escaneados (`InformeFinancieroTriemestre2`, etc.).
4. **Obtención de Beans**:
   - Se obtienen los beans `comercialExperimentadoAnotacionConfiguration` y `directorFinanciero` del contexto de Spring.
5. **Verificación de Singleton/Prototype**:
   - Se comparan las instancias de los beans `comercialExperimentadoAnotacionConfiguration` (`antonio` y `lucia`) para determinar si son singleton o prototype.
6. **Salida de Información**:
   - Se imprime la información sobre la dirección de memoria y los hashCodes de los beans, además del informe generado por el bean `directorFinanciero`.
7. **Cierre del Contexto**:
   - Finalmente,