FORMATO DE INFORME DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA ESTUDIANTES

CARRERA: COMPUTACIÓN		**ASIGNATURA**: Programación Aplicada
NRO. PRÁCTICA:	3	**TÍTULO PRÁCTICA**: Patrones en Java

OBJETIVO ALCANZADO:

- Identificar los cambios importantes de Java
- Diseñar e Implementar las nuevas técnicas de programación
- Entender los patrones de Java

ACTIVIDADES DESARROLLADAS

1. Revisar la teoría y conceptos de Patrones de Diseño de Java.

2. Diseñar e implementar cada estudiante un patrón de diseño y verificar su funcionamiento. A continuation se detalla el patron a implementar:

NELSON PAUL ORTEGA SEGARRA	Adapter

Pertenece a un tipo de patrón estructural
El patrón Adapter resuelve incompatibilidades entre interfaces adaptando una interfaz hacia la otra utilizando un "adaptador" encargado de la conversión entre interfaces.
Partes del patrón Adapter:
Target: la interfaz que usamos para crear el adapter.
Adapter: es la implementación del target y que se ocupará de realizar la adaptación.
Client: es el que interactúa y usa el adapter.
Adaptee: es la interfaz incompatible que necesitamos adaptar con el adapter.

Función del Adaptador:

1. El adaptador obtiene una interfaz compatible con uno de los objetos existentes.
2. Utilizando esta interfaz, el objeto existente puede invocar con seguridad los métodos del adaptador.
3. Al recibir una llamada, el adaptador pasa la solicitud al segundo objeto, pero en un formato y orden que ese segundo objeto espera.

Cuando usar Adapter:

- Cuando quieras usar una clase existente, pero cuya interfaz no sea compatible con el resto del código.
- Cuando se quiera reutilizar varias subclases existentes que carezcan de alguna funcionalidad común que no pueda añadirse a la superclase.

4. Probar y modificar el patrón de diseño a fin de generar cuales son las ventajas y desventajas.

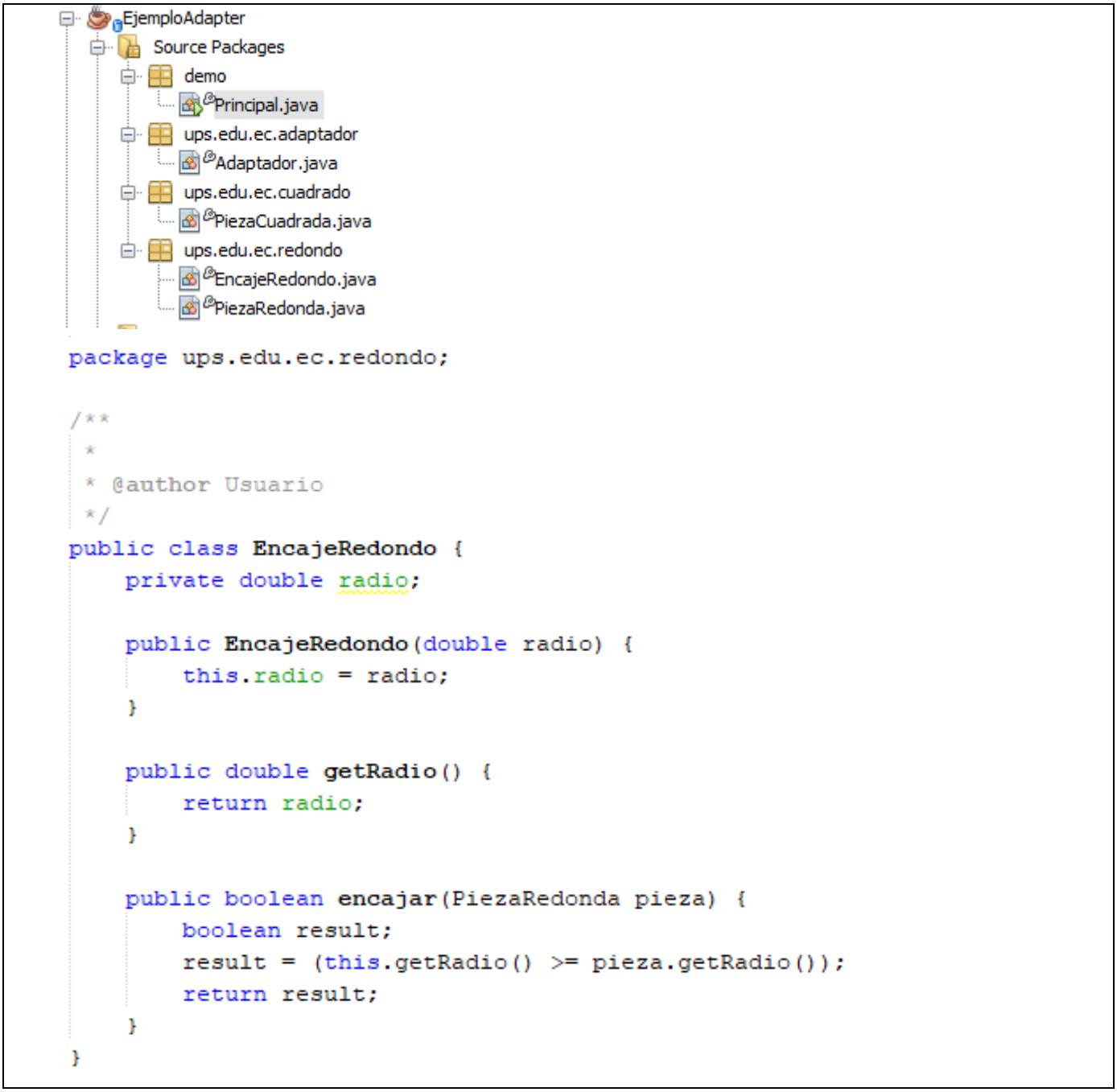
Ventajas:

- Se puede separar la interfaz o el código de conversión de datos de la lógica de negocio primaria del programa.
- Se puede introducir nuevos tipos de adaptadores al programa sin descomponer el código existente.

Desventajas:

- La complejidad del código aumenta, ya que se introducen nuevas clases e interfaces. Por lo que muchas veces se opta por cambiar la clase de servicio para que coincida con el resto del código.

5. Realizar práctica codificando los códigos de los patrones y su estructura.




```java
package ups.edu.ec.redondo;

/**
 *
 * @author Usuario
 */
public class EncajeRedondo {
    private double radio;

    public EncajeRedondo(double radio) {
        this.radio = radio;
    }

    public double getRadio() {
        return radio;
    }

    public boolean encajar(PiezaRedonda pieza) {
        boolean result;
        result = (this.getRadio() >= pieza.getRadio());
        return result;
    }
}
```

```java
package ups.edu.ec.redondo;

/**
 *
 * @author Usuario
 */
public class PiezaRedonda {
    private double radio;

    public PiezaRedonda() {}

    public PiezaRedonda(double radio) {
        this.radio = radio;
    }

    public double getRadio() {
        return radio;
    }
}
```

```java
package ups.edu.ec.cuadrado;

/**
 *
 * @author Usuario
 */
public class PiezaCuadrada {
    private double ancho;

    public PiezaCuadrada(double width) {
        this.ancho = width;
    }

    public double getAncho() {
        return ancho;
    }

    public double getCuadrado() {
        double result;
        result = Math.pow(this.ancho, 2);
        return result;
    }
}
```

```java
package ups.edu.ec.adaptador;

import ups.edu.ec.cuadrado.PiezaCuadrada;
import ups.edu.ec.redondo.PiezaRedonda;

/**
 *
 * @author Usuario
 */
public class Adaptador extends PiezaRedonda{
    private PiezaCuadrada pieza;

    public Adaptador(PiezaCuadrada peg) {
        this.pieza = peg;
    }

    @Override
    public double getRadio() {
        double result;
        // Calcule un radio de círculo mínimo, que pueda ajustarse a esta pieza
        result = (Math.sqrt(Math.pow((pieza.getAncho() / 2), 2) * 2));
        return result;
    }
}
```

```java
import ups.edu.ec.cuadrado.PiezaCuadrada;
import ups.edu.ec.redondo.PiezaRedonda;
import ups.edu.ec.adaptador.Adaptador;
import ups.edu.ec.redondo.EncajeRedondo;

/**
 *
 * @author Usuario
 */
public class Principal {

    public static void main(String[] args) {
        // pieza redonda en agujero redondo EncajeRedondo
        EncajeRedondo hole = new EncajeRedondo(5);
        PiezaRedonda rpeg = new PiezaRedonda(5);
        if (hole.encajar(rpeg)) {
            System.out.println("La pieza redonda de radio 5 encaja en el agujero redondo de radio 5.");
        }

        PiezaCuadrada piezaPeq = new PiezaCuadrada(2);
        PiezaCuadrada piezaGra = new PiezaCuadrada(20);
        // hole.encajar(piezaPeq); // No va a compilar

        // El Adaptador soluciona el problema
        Adaptador AdaptadorPiezaPeq = new Adaptador(piezaPeq);
        Adaptador AdaptadorPiezaGra = new Adaptador(piezaGra);
        if (hole.encajar(AdaptadorPiezaPeq)) {
            System.out.println("La pieza cuadrada de ancho 2 encaja en el agujero redondo de radio 5.");
        }
        if (!hole.encajar(AdaptadorPiezaGra)) {
            System.out.println("La pieza cuadrada de ancho 20 no encaja en el orificio redondo de radio 5.");
        }
    }

}
```

RESULTADO(S) OBTENIDO(S):

- Realizar procesos de investigación sobre los patrones de diseño de Java
- Entender los patrones y su utilización dentro de aplicaciones Java.
- Entender las funcionalidades basadas en patrones.

CONCLUSIONES:

- Aprenden a trabajar dentro de plazos de tiempo establecidos, manejando el lenguaje de programación de Java**.**
- Se encuentra páginas de información confiable para el tema de patrones

RECOMENDACIONES:

- Realizar el trabajo dentro del tiempo establecido.
- Revisar el siguiente link: https://refactoring.guru/es/design-patterns/java

Nombre de estudiante: Nelson Paul Ortega Segarra

Firma de estudiante: