IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

UD 3.- Creación de componentes visuales.

Enlace para exportar un proyecto javafx : https://javautodidacta.es/como-exportar-proyecto-javafx-intellij/

1 Desarrollo de componentes

1.1 Concepto de componente

Un **componente** es un elemento software reutilizable, con una especificación y funcionalidad clara, usando en la ingeniería de software basada en componentes.

Por definición, un **componente** puede ser reutilizado en diferentes sistemas, así como también puede ser reemplazado por otro que sea capaz de proveer la misma funcionalidad.

El desarrollo de software basado en componentes permite reutilizar piezas de código para diversas tareas, que conlleva diversos beneficios como las mejoras a la calidad, reducción del ciclo de desarrollo y el mayor retorno sobre la inversión.

Un **componente** puede ser tan simple como un conversor Dolar-Euro o tan complejo como para manejar un sistema completo. Los componentes más complejos pueden ser divididos en partes más simples, subcomponentes, igualmente reutilizables y reemplazables.

Cuando se habla de **componentes gráficos**, se habla de un grupo de elementos gráficos que funcionan de manera conjunta. Estos pueden ser paneles completos, grupos de botones, agrupación de varios componentes o incluso vistas completas de la interfaz o ventanas personalizadas, como son los cuadros de dialogo, normalmente ya implementados en la biblioteca grafica.

1.2 Características:

Para poder catalogar un componente como tal ha de cumplir una serie de características. Un componente puede ser:

- **Independiente de la plataforma:** hardware, software o sistema operativo.
- **Identificable:** para poder clasificarlo y acceder a sus servicios.
- **Autocontenido**, es decir, no debe requerir la utilización de otros componentes para su correcto funcionamiento.
- **Reemplazable**, por nuevas versiones u otro componente que realice lo mismo.
- **Accesible solamente a través de su interfaz**. Una interfaz define las operaciones (servicios o responsabilidades) que el componente puede realizar.
- **Invariante.** Sus servicios no deben variar, su implementación puede.
- **Documentado apropiadamente**, para facilitar su utilización.
- **Genérico**, sus servicios han de ser lo más reutilizables posible.

- **Reutilizable dinámicamente**, cargado en tiempo de ejecución en una aplicación.
- **Distribuido como paquete,** que almacena todos los elementos que los componen.

1.3 Puntos fuertes y débiles:

Como todo, los componentes tienen ciertas características que lo hacen útiles, pero el desarrollo de estos implica complicación extra, a veces innecesaria.

Ventajas:

- Reutilizacion del software. Nos lleva a alcanzar un mayor nivel de reutilización de software.
- Agiliza las pruebas. Permite hacer pruebas a cada componente antes de probar el conjunto completo de componentes ensamblados.
- Simplifica el mantenimiento del sistema. Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- Mayor cálida. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorara con el paso del tiempo.
- Funcionalidad mejorada. Para usar un que contenga una pieza de funcionalidad, solo se necesita entender su naturaleza, mas no sus detalles internos.

Desventajas

Las limitaciones en el desarrollo de componentes son las siguientes:

- Solo es habitual su aplicación en algunos campos como las interfaces graficas.
- No siempre es posible encontrar componentes adecuados para cada aplicación.
- No todas las plataformas están preparadas para ejecutar todos los componentes.
- No hay procesos de certificación para garantizar la calidad de los componentes.
- La falta de estándares hacen difícil la localización, adaptación y reusabilidad.

2 ¿Qué es la serialización y para qué sirve?

La serialización es el proceso de convertir el estado de un objeto en un formato que se pueda almacenar o transportar. El complemento de serialización es deserialización, que convierte una secuencia en un objeto. Juntos, estos procesos permiten almacenar y transferir datos.

Para que un programa java pueda convertir un objeto en un montón de bytes y pueda luego recuperarlo, el objeto necesita ser **Serializable**. Al poder convertir el objeto a bytes, ese objeto se puede enviar a través de red, guardarlo en un fichero, y después reconstruirlo al otra lado de la red, leerlo del fichero

Para que un objeto sea serializable basta con que implemente la interfaz **Serializable**. Como la interfaz Serializable no tiene métodos, es muy sencillo implementarla, basta con un implements Serializable y nada más.

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

Por ejemplo, la clase Datos siguiente es Serializable y java sabe perfectamente enviarla o recibirla por red, a través de socket o de rmi. También java sabe escribirla en un fichero o reconstruirla a partir del fichero.

```
public class Datos implements Serializable
{
  public int a;
  public String b;
  public char c;
}
```

Si dentro de la clase hay atributos que son otras clases, éstos a su vez también deben ser Serializable. Con los tipos de java (String, Integer, etc.) no hay problema porque lo son. Si ponemos como atributos nuestras propias clases, éstas a su vez deben implementar Serializable. Por ejemplo

```
/* Esta clase es Serializable porque implementa Serializable y todos sus
 *  campos son Serializable, incluido "Datos f;"
 */
public class DatoGordo implements Serializable
{
  public int d;
  public Integer e;
  Datos f;
}
```

Además vamos a necesitar dos clases ObjectOutputStream y ObjectInputStream, para distribuir y recuperar un objeto. (Ir a la API y echar un vistazo de estas clases y de la interfaz serializable)

Pasos: Almacenar el objeto empleado

```
class Empleado implements Serializable{
public Empleado(String n, double s, int agno, int mes, int dia){

        nombre=n;

        sueldo=s;

        GregorianCalendar calendario=new GregorianCalendar(agno, mes-1,dia);

        fechaContrato=calendario.getTime();

}


public String getNombre() {
        return nombre;
}

public double getSueldo() {
        return sueldo;
}

public Date getFechaContrato() {
        return fechaContrato;
}
```

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es


```
        public void subirSueldo(double porcentaje){
                double aumento=sueldo*porcentaje/100;
                sueldo+=aumento;
        }
        public String toString(){
                return "El Nombre es =" + nombre + ",y su sueldo es =" + sueldo + ", fecha de contrato="
+ fechaContrato;
        }
        private String nombre;
        private double sueldo;
        private Date fechaContrato;
}
}
```

1. **Crear un flujo de salida:**

   ObjectOutputStream flujoSalida=new ObjectOutputStream(new FileOutputStream(C:\salida\objeto.dat);

2. **Escribir ese flujo de salida:**

   flujoSalida.writeObject(persona)

3. **Crear una nueva clase Recuperando**

```
public class Recuperando
 {
        public static void main(String[] args){
                ObjectInputStream flujoEntrada=new ObjectInputStream(new FileInputS-
tream(C:\salida\objeto.dat)
                Empleado[] entradaObjeto=(Empleado[])flujoEntrada.readObject();
                flujoEntrada.close();
        }
        }
```