UT2.2. Creación de componentes visuales

24 horas

Contenidos

1. Desarrollo de software basado en componentes. Reutilización del software. Beneficios.
2. Componentes visuales: concepto de componente; propiedades y atributos.
3. Eventos:
   a. Componentes y eventos
   b. Listeners
   c. Métodos y eventos
4. Persistencia del componente.
5. Herramientas para desarrollo de componentes visuales.
6. Empaquetado de componentes.

Resultados de Aprendizaje

RA2: Genera interfaces naturales de usuario utilizando herramientas visuales	5%
a) Se han identificado las herramientas disponibles para el aprendizaje automático relacionadas con las interfaces de usuario.	40% .
b) Se ha creado una interfaz natural de usuario utilizando las herramientas disponibles.	25% .
c) Se ha utilizado el reconocimiento de voz para implementar acciones en las interfaces naturales de usuario.	25% .
d) Se ha incorporado la detección del movimiento del cuerpo para implementar acciones en las interfaces naturales de usuario.	5% .
e) Se han integrado elementos de detección de partes del cuerpo para implementar acciones en las interfaces naturales de usuario.	5% .
f) Se ha integrado la realidad aumentada en los interfaces de usuario.	5%

Resultados de Aprendizaje

RA3: Crea componentes visuales valorando y empleando herramientas específicas	15%
a) Se han identificado las herramientas para diseño y prueba de componentes.	10%
b) Se han creado componentes visuales.	10%
c) Se han definido sus métodos y propiedades con asignación de valores por defecto.	10% .
d) Se han determinado los eventos a los que debe responder el componente y se les han asociado las acciones correspondientes.	20% .
e) Se han realizado pruebas unitarias sobre los componentes desarrollados.	10%
f) Se han documentado los componentes creados.	10%
g) Se han empaquetado componentes.	10%
h) Se han programado aplicaciones cuyo interfaz gráfico utiliza los componentes creados.	20% .

Creación de componentes personalizados

Extender una clase existente de Swing:

Puedes crear un componente personalizado extendiendo cualquier componente de Swing existente (como JPanel, JButton, etc.) o JComponent si deseas empezar desde cero.

- **Personalizar el comportamiento:** Sobrescribir métodos clave como paintComponent() para personalizar el dibujo o agregar funcionalidad específica.
- **Agregar propiedades y métodos:** Definir nuevas propiedades y métodos para el componente, como colores, acciones especiales, etc.
- **Agregar eventos personalizados:** Añadir oyentes de eventos (EventListener) si el componente necesita responder a interacciones (clics, teclado, etc.).

Ejemplo

Descarga *RoundButton.java* y *CustomButtonTest.java* y estudíalos en clase.

La clase *CustomButtonTest* es la que debes ejecutar para probar el nuevo botón que hemos tuneado.

Empaquetado de componentes en .jar

Después de crear tu componente personalizado, puedes empaquetarlo en un archivo .jar para distribuirlo o reutilizarlo.

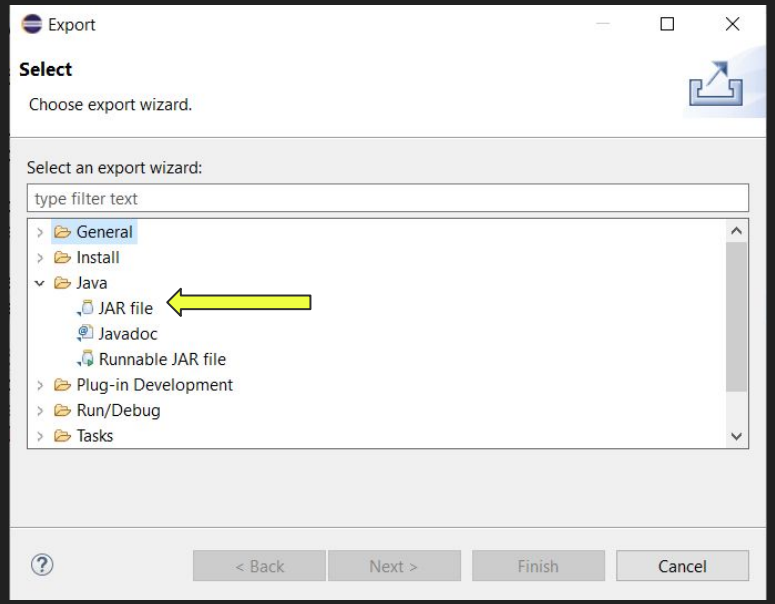
Pasos para empaquetar el archivo .jar:

1. **Compila tu clase:** Asegúrate de que tu componente (y cualquier otra clase relacionada) esté compilado. Esto generará archivos .class en el directorio bin o out (según tu IDE).

2. **Crea el archivo .jar:** Los IDEs ofrecen opciones de exportación.

1. **Si usas Eclipse:**

Haz clic derecho en el proyecto y selecciona Exportar.
Elige JAR file y sigue los pasos para empaquetar las clases y recursos necesarios.
Indica dónde quieres guardar el archivo .jar y selecciona qué clases deseas empaquetar.



Empaquetado de componentes en .jar

2. Si usas IntelliJ IDEA:

Ve a File > Project Structure y selecciona Artifacts.
Crea un nuevo JAR Artifact y selecciona las clases que quieres empaquetar.
Luego, haz clic en Build > Build Artifacts > Build para generar el .jar.

3. Si usas la línea de comandos:

Coloca todos los archivos .class en un directorio (por ejemplo, bin/).
Usa el siguiente comando en la terminal:

bash

Copiar código

jar cfmi Boton.jar MANIFEST.MF -C bin/ .

Aquí, Boton.jar será el archivo empaquetado, y MANIFEST.MF es el archivo de manifiesto que puede contener información opcional como el punto de entrada (Main-Class).
Agregar el manifiesto (opcional): Si tu componente tiene una clase principal o necesitas definir dependencias, crea un archivo MANIFEST.MF con algo como esto:

Manifest-Version: 1.0

Main-Class: CustomButtonTest

Importado de componentes en .jar

Usar el .jar: Puedes añadir el archivo .jar a cualquier proyecto de Java que quiera usar tu componente. Solo tienes que importar el .jar en las dependencias del proyecto.

Usar el .jar en un Nuevo Proyecto

Una vez que tienes tu .jar, puedes usarlo en otros proyectos de Java.

- **En Eclipse o IntelliJ:**

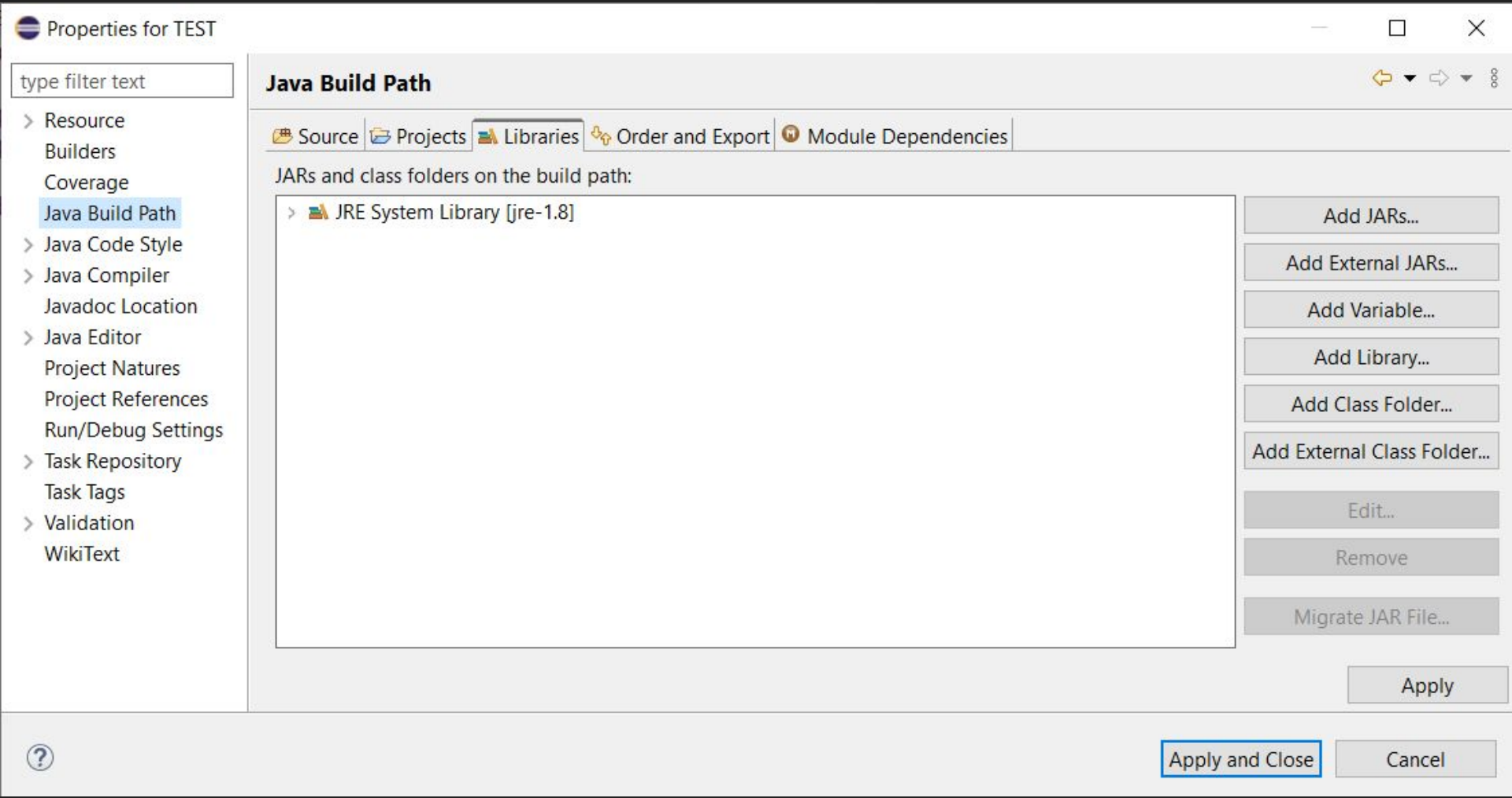
  Importa el archivo .jar en el proyecto (haz clic derecho en el proyecto > Build Path > Add External JARs).

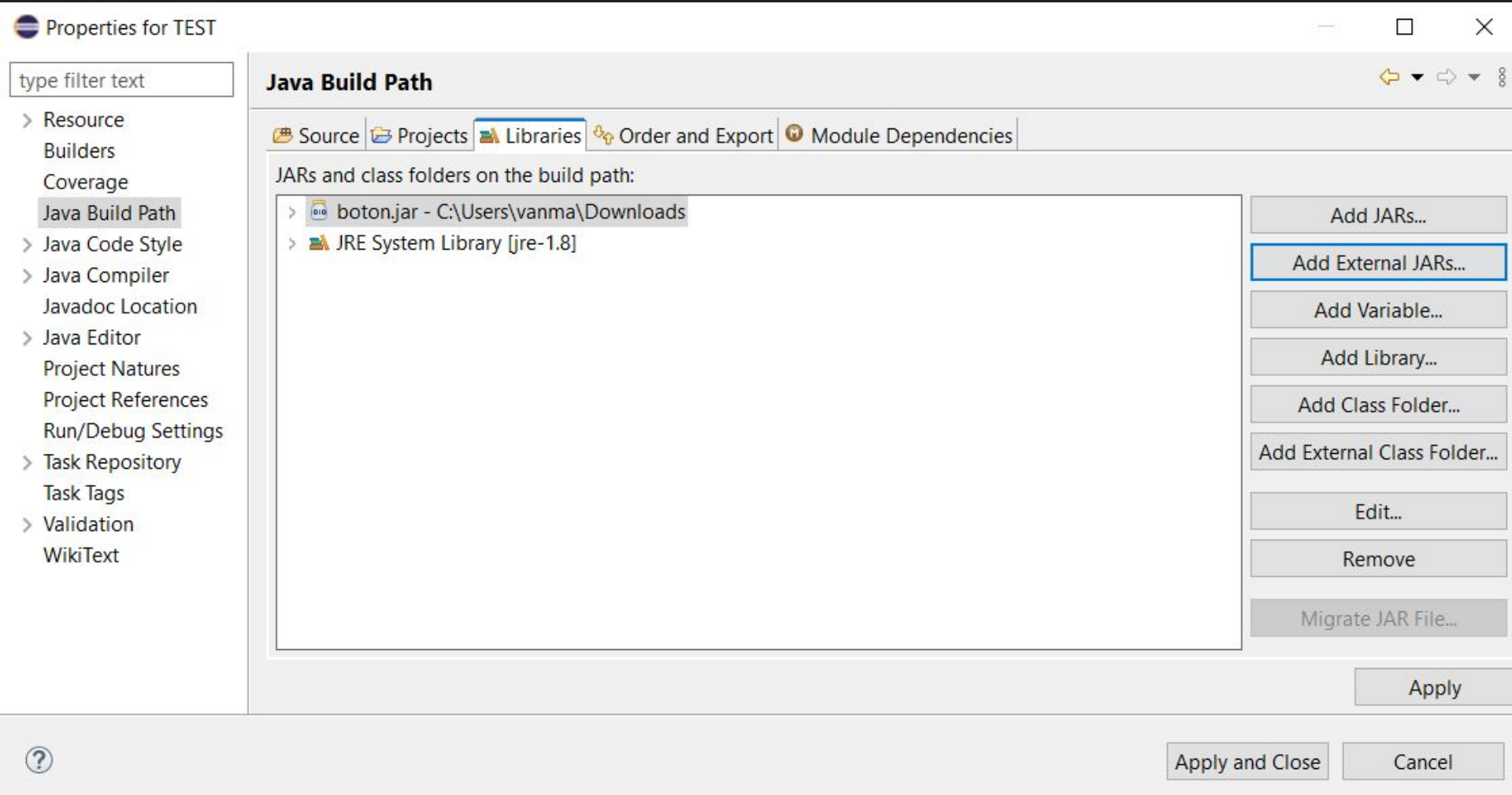
  Usa el componente normalmente importando las clases desde el .jar.

Importado de componentes en .jar

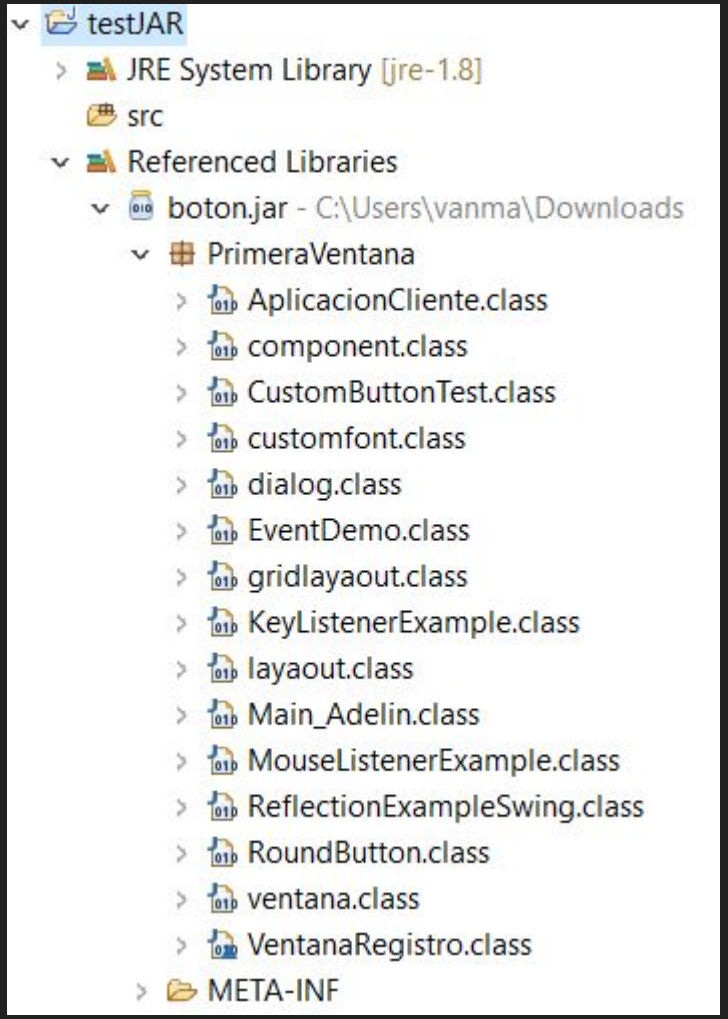
Vamos a incluir nuestro jar exportado en un nuevo proyecto y a usarlo con una clase de prueba TestJar.

1. Crea un nuevo proyecto que llame testJAR.
2. Haz clic derecho en el nombre del proyecto y selecciona Properties (Propiedades).
3. En la ventana de propiedades, ve a la opción Java Build Path y selecciona la pestaña Libraries.
4. Haz clic en Add External JARs y selecciona el archivo .jar que contiene la clase RoundButton.


Properties for TEST
type filter text
Resource
Builders
Coverage
Java Build Path
Java Code Style
Java Compiler
Javadoc Location
Java Editor
Project Natures
Project References
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
Module Dependencies
JARs and class folders on the build path:
JRE System Library [jre-1.8]
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Apply
Apply and Close
Cancel


Properties for TEST
type filter text
Resource
Builders
Coverage
Java Build Path
Java Code Style
Java Compiler
Javadoc Location
Java Editor
Project Natures
Project References
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
Module Dependencies
JARs and class folders on the build path:
boton.jar - C:\Users\vanma\Downloads
JRE System Library [jre-1.8]
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Apply
Apply and Close
Cancel

El nuevo proyecto tiene todas las clases del boton.jar que exportamos.



Probando nuestro .jar

Crea una nueva clase TestJar.

```
import javax.swing.*;
import PrimeraVentana.RoundButton;  // Importar desde el .jar

public class TestJar {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Prueba de Componente JAR");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);

        // Usar el botón redondo desde el .jar
        RoundButton roundButton = new RoundButton("Componente JAR");

        frame.add(roundButton);
        frame.setVisible(true);
    }
}
```

RoundButton.java | MouseListenerExample.java | CustomButtonTest.java | TestJar.java

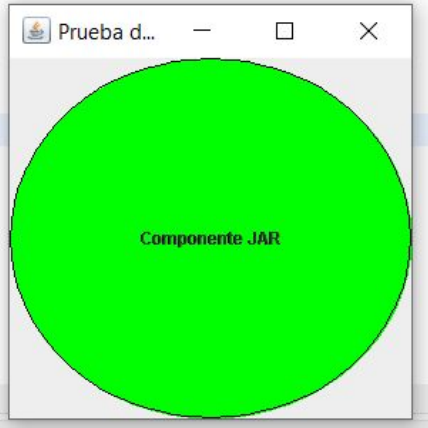
```java
package testJAR;

import javax.swing.*;
import PrimeraVentana.RoundButton;  // Importar desde el .jar

public class TestJar {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Prueba de Componente JAR");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);

        // Usar el botón redondo desde el .jar
        RoundButton roundButton = new RoundButton("Componente JAR");

        frame.add(roundButton);
        frame.setVisible(true);
    }
}
```



Console

```
TestJar (1) [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe  (12 oct 2024 20:25:29) [pid: 14244]
```

Actividad 2

1. Personaliza otro componente que no sea un botón.
2. Pruébalo desde una clase de prueba como hicimos con CustomButtonTest.
3. Empaquétalo en un .jar.
4. Impórtalo en el proyecto de la Práctica 2 o el de la Práctica 2 EXTRA y prueba que funciona.

Introspección en Java

La introspección en Java es la capacidad de inspeccionar las características de una clase o un objeto en **tiempo de ejecución**. A través de la introspección, podemos acceder a información sobre las propiedades, métodos y eventos de los objetos.

Java ofrece introspección a través de su **API de reflexión**, la cual nos permite:

- Obtener los métodos y atributos de una clase.
- Invocar métodos de una clase en tiempo de ejecución.
- Crear nuevas instancias de objetos dinámicamente.

Reflexión en Java

La reflexión es el mecanismo que permite a los programas de Java examinar y modificar la estructura interna de los objetos y clases en tiempo de ejecución. A través de la reflexión podemos manipular objetos o invocar métodos sin conocer la clase exacta en tiempo de compilación.

Se utilizan las clases del **paquete *java.lang.reflect*** (como Class, Method, Field, etc.).

En el desarrollo de interfaces gráficas con Swing, la reflexión puede ser muy útil para:

- Crear componentes dinámicamente a partir de clases cuyo tipo no conoces en tiempo de compilación.
- Asignar eventos o cambiar las propiedades de un componente Swing a través de reflexión, como configurar botones, etiquetas u otros componentes.

Reflexión en Java

Esto es útil cuando queremos modificar componentes dinámicamente o basados en algún input externo.

Descarga el ejemplo R*eflectionExampleSwing.java* del Aula Virtual y estudia cómo el texto del botón cambia en tiempo de ejecución usando reflexión, un concepto avanzado y muy potente.

Implicaciones en el desarrollo:

- **Flexibilidad:** La reflexión permite escribir código más flexible, pero hay que tener cuidado porque el uso excesivo de reflexión puede afectar el rendimiento.
- **Seguridad:** El uso de reflexión puede romper la encapsulación de las clases y acceder a métodos privados o campos. Esto debe hacerse con responsabilidad.

La reflexión es fundamental cuando se trabaja con bibliotecas que pueden ser desconocidas o cuando se desea modificar clases de manera genérica (por ejemplo, frameworks como **Spring** o **Hibernate** usan reflexión ampliamente).

Práctica 3

Realiza los ejercicios del pdf que encontrarás en el Aula Virtual.