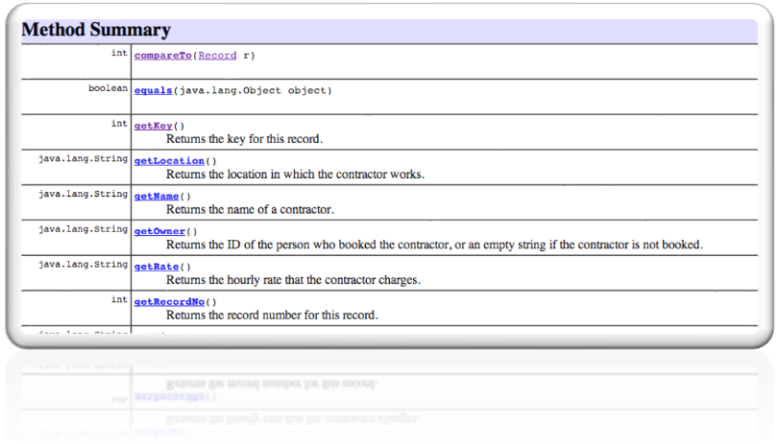
[Estructuras de Datos]



GENERACIÓN AUTOMÁTICA DE DOCUMENTACIÓN EN JAVA: JAVADOC.

TEST UNITARIO EN JAVA: JUNIT.

EXPORTAR E IMPORTAR CÓDIGO JAVA: JAR.

Copyright

Recursos adicionales

Enlaces externos de interés

- Acceda a información útil mediante enlaces externos a:
  - **código fuente** disponible de forma **online**.
  - **otro material** disponible de forma **online**.

GENERACIÓN AUTOMÁTICA DE DOCUMENTACIÓN: JAVADOC

Se recomienda que todo lo documentado en la siguiente sección sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

¿Qué es Javadoc?

- **Javadoc** es una utilidad de Oracle para la **generación** de **documentación** de **APIs** en formato HTML a partir de **código fuente Java**.
- **Javadoc** es el estándar de la industria para documentar clases de Java. La mayoría de los IDEs los generan automáticamente.
- Se deben utilizar los comentarios especiales **/** ... */** con algunas **palabras claves** para determinar la documentación.

En esta materia se busca que los alumnos **aprendan** y **ejerciten** la tarea de **documentar sus soluciones de software**, considerando la siguiente premisa: cualquier persona ajena al código desarrollado y que lo desee utilizar, debe poder con la documentación **interpretar qué hace**, y cerciorarse **cómo es que se usa**.

¿Cómo documentar una API?

- La **correcta** **documentación** de una API es **fundamental** para los clientes que van a utilizarla. En cierto sentido, es una **garantía** de un producto final de la **mayor** **calidad**.
- Se debe **describir** de forma **precisa** y **completa** (sin entrar en detalles de **implementación**), aquellos elementos que forman parte de la API: **clases**, **métodos**, **excepciones**, etc.
- Los **comentarios** están destinados y serán utilizados por los **clientes** de la API documentada. Por esto, es **imprescindible** que las **precondiciones**, **parámetros**, **valores** de **retornos**, **excepciones**, etc., sean **completamente definidas**, para su correcto uso.

Ejemplo: documentando código (no Javadoc)

```
public class Calculadora {
  public static float a(float b, float c) {…} ☹
}
```

*"cualquier persona ajena al código desarrollado y que lo desee utilizar, debe poder con la documentación **interpretar qué hace**, y cerciorarse **cómo es que se usa**."*

Ejemplo: documentando código (no Javadoc)

```
public class Calculadora {
  public static float a(float b, float c) {…} ☹
  public static float dividir(float a, float b){…} ☹
}
```

"cualquier persona ajena al código desarrollado y que lo desee utilizar, debe poder con la documentación interpretar qué hace, y cerciorarse cómo es que se usa."

Ejemplo: documentando código (no Javadoc)

```
public class Calculadora {
  public static float a(float b, float c) {…} ☹
  public static float dividir(float a, float b){…}☹
  // Computa la división real entre dos valores.
  // Parámetro a: valor del dividendo de la división.
  // Parámetro b: valor del divisor de la división.
  // Retorna: resultado de aplicar la división.
  // Lanza: ErrorException en caso de que el divisor sea cero.
  public static float dividir(float a, float b) throws ErrorException{…}😀
}
```

*"cualquier persona ajena al código desarrollado y que lo desee utilizar, debe poder con la documentación **interpretar qué hace**, y cerciorarse **cómo es que se usa**."*

Ejemplo: documentando código (no Javadoc)

```
public class Calculadora {
  public static float a(float b, float c) {…} ☹
  public static float dividir(float a, float b){…}☹
  // Computa la división real entre dos valores.
  // Parámetro a: valor del dividendo de la división.
  // Parámetro b: valor del divisor de la división.
  // Retorna: resultado de aplicar la división.
  // Lanza: ErrorException en caso de que el divisor sea cero.
  public static float dividir(float a, float b) throws ErrorException{…}😀
  // Computa la división real entre dos valores.
  // Parámetro dividendo: valor del dividendo de la división.
  // Parámetro divisor: valor del divisor de la división.
  // Retorna: resultado de aplicar la división.
  // Lanza: DivisionPorCero en caso de que el divisor sea cero.
  public static float dividir(float dividendo, float divisor) throws DivisionPorCero{…}😀😀😀
}
```

*"cualquier persona ajena al código desarrollado y que lo desee utilizar, debe poder con la documentación **interpretar qué hace**, y cerciorarse **cómo es que se usa**."*

Generando Javadoc: palabras claves

- Javadoc define **etiquetas** para indicar **qué** tipo de elemento de la **documentación** se está **describiendo**. Algunas de ellas:

Tag	Descripción	Uso
@author	Nombre del desarrollador.	nombre_autor
@deprecated	Indica que el método o clase es antigua y que no se recomienda su uso porque posiblemente desaparecerá en versiones posteriores.	descripción
@param	Definición de un parámetro de un método, es requerido para todos los parámetros del método.	nombre_parametro descripción
@return	Informa de lo que devuelve el método, no se puede usar en constructores o métodos "void".	descripción
@see	Asocia con otro método o clase.	referencia (#método(); clase#método(); paquete.clase; paquete.clase#método()).
@throws	Excepción lanzada por el método	nombre_clase descripción
@version	Versión del método o clase.	versión

Ejemplo: documentando código (Javadoc)

```java
/**
 * Modela las operaciones de una calculadora.
 * @author Federico Joaquín (federico.joaquin@cs.uns.edu.ar)
 */
public class Calculadora {
/**
 * Computa la división real entre dos valores.
 * @param dividendo Valor del dividendo de la división.
 * @param divisor Valor del divisor de la división.
 * @return Resultado de aplicar la división.
 * @throws DivisionPorCero en caso que divisor sea cero.
 */
public static float dividir(float dividendo, float divisor) throws DivisionPorCero{…}
}
```

"cualquier persona ajena al código desarrollado y que lo desee utilizar, debe poder con la documentación interpretar qué hace, y cerciorarse cómo es que se usa."

Ejemplo: documentando código (Javadoc)

- La documentación se exporta fuera de la IDE haciendo uso del ejecutable ***javadoc*** que viene incluído en el **JDK** instalado. ***Ej: javadoc.exe Pila.java***
- Desde la IDE Eclipse, se selecciona la clase, paquete o proyecto para el que se desea generar la documentación, y luego: *File* → *Exportar* → *Javadoc*.
- Siguiendo los pasos del asistente hay que indicar el **destino** de la carpeta de documentación, así como también se debe indicar **la ubicación del ejecutable javadoc** (carpeta de instalación de la jdk***../bin/javadoc.exe***)

Para tener en cuenta…

Interfaces de los TDAs usados en la materia

- En esta materia, **todas** las interfaces que se usarán **ya se encuentran documentadas**. ☺
- Las interfaces están **disponibles** en la página de la materia y son las que **deben utilizar para programar en esta materia**.
- Javadoc permite "**heredar**" la documentación de una **interfaz** en una clase que la **implementa** mediante la **palabra clave** `@Override`.
- Entonces, ¿**qué** es lo que **deben documentar** los **alumnos** en sus **implementaciones**?

1) Encabezado de las clases.
2) Constructores.
3) Métodos privados y auxiliares.
4) Encabezado y constructores de excepciones.
5) Las clases auxiliares (ej. GUIs) que no implementan interfaces.

TEST UNITARIO EN JAVA: JUNIT

Se recomienda que todo lo documentado en la siguiente sección sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

¿Qué es JUnit?

- **JUnit** es un **conjunto de bibliotecas** que son utilizadas en **programación** para hacer **pruebas unitarias** de aplicaciones **Java**.
  - Una **prueba unitaria** es una forma de comprobar el **correcto** funcionamiento de una **unidad de código**.
  - En nuestro caso, las unidades de código serán los **TDAs** implementados.
- **JUnit** es un **framework** que permite **evaluar el funcionamiento** de los **métodos** de una **clase**.
- En función de algún **valor de entrada** permite evaluar el valor de **retorno** esperado; si la clase **cumple** con la **especificación**, entonces **JUnit** indicará que el método pasó **exitosamente** la prueba; en caso de que el valor esperado sea **diferente** al esperado, **JUnit** devolverá un **fallo** en el método correspondiente.

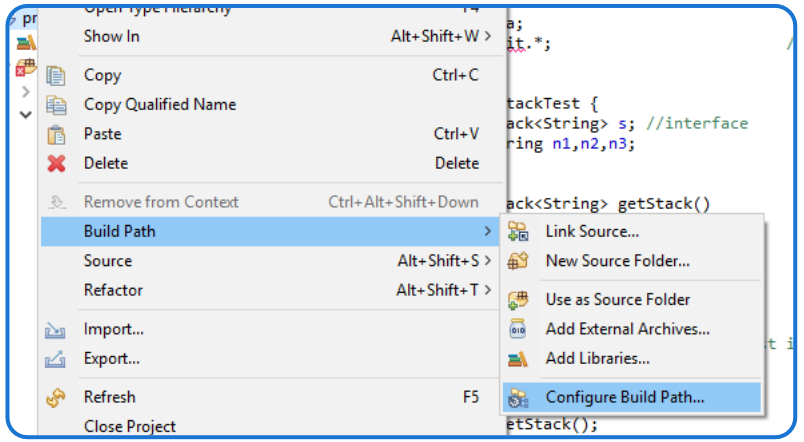
¿Qué es JUnit?

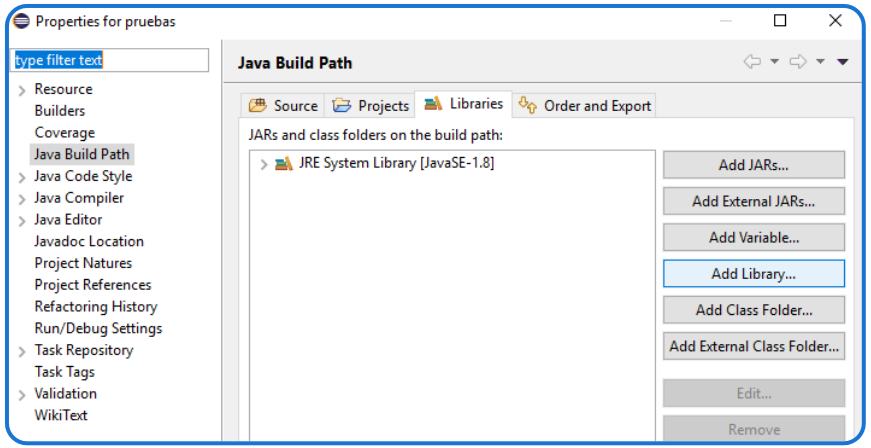
- Esta materia no tiene por **objetivo** el **aprendizaje** de **cómo** se definen los casos de pruebas en **JUnit**, sino que se usarán **casos de test definidos por la cátedra** para **validar** la **correctitud** de las operaciones de los **TDAs**.
- Quedan los alumnos en **total libertad** de indagar, consultar e interiorizarse sobre cómo los tests se definen, qué herramientas presenta **JUnit** para tal fin, y aun más, se pueden definir nuevos casos de prueba.

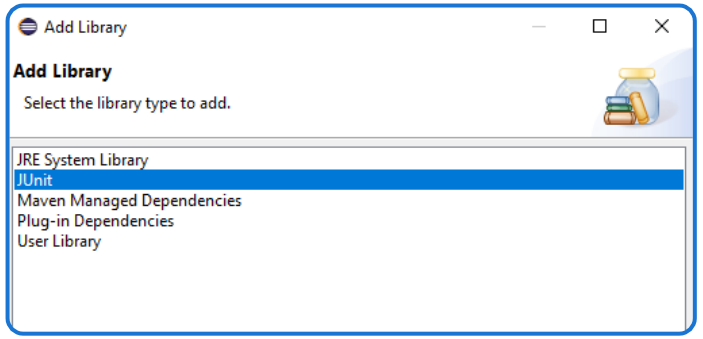
En la materia es **obligatorio** para la aprobación del proyecto, por caso, que los **TDAs** solicitados **reporten** que **todas las operaciones** definidas en estos retornan **resultados exitosos** cuando son analizadas por los **tests definidos** para tales **TDAs**. Es importante en este sentido, que las implementaciones que los alumnos realicen sean **validadas** de forma permanente por estos tests.

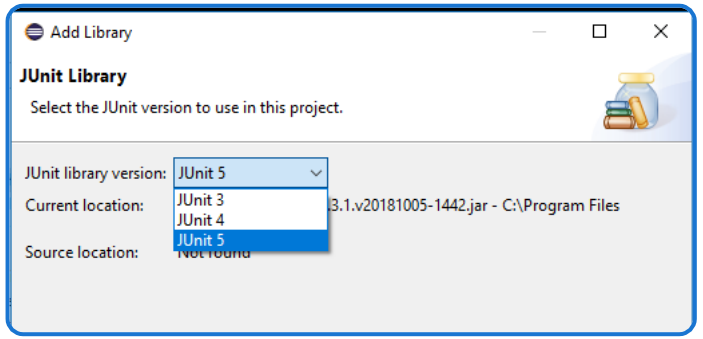
¿Cómo utilizar los test definidos en JUnit?

- Primero es necesario que el **proyecto Java** contenga el **framework JUnit** (librería).
- Click derecho en el proyecto → *Build Path* → *Configure Build Path..*
- En la pestaña *Libraries* → *Add Library…*
- Seleccionar la librería *JUnit* → *Next* → *JUnit X* (X es la versión que se disponga).



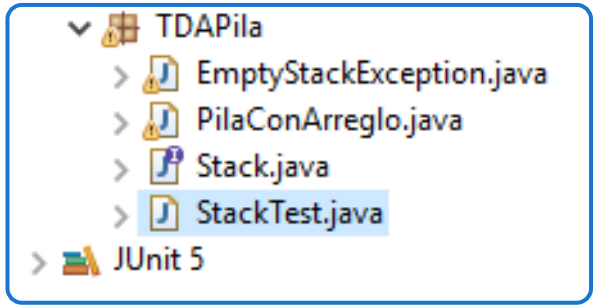


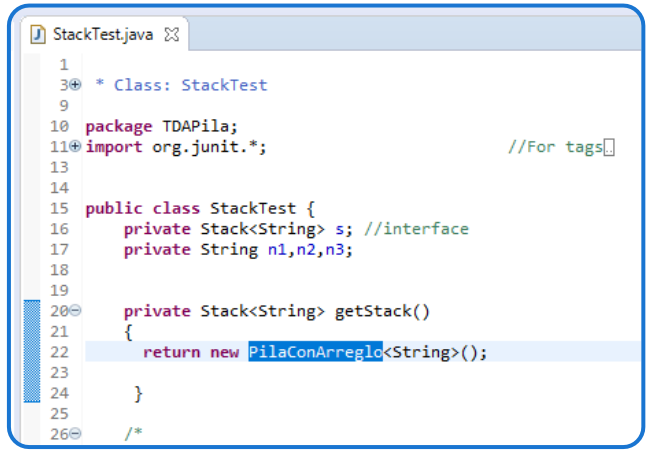


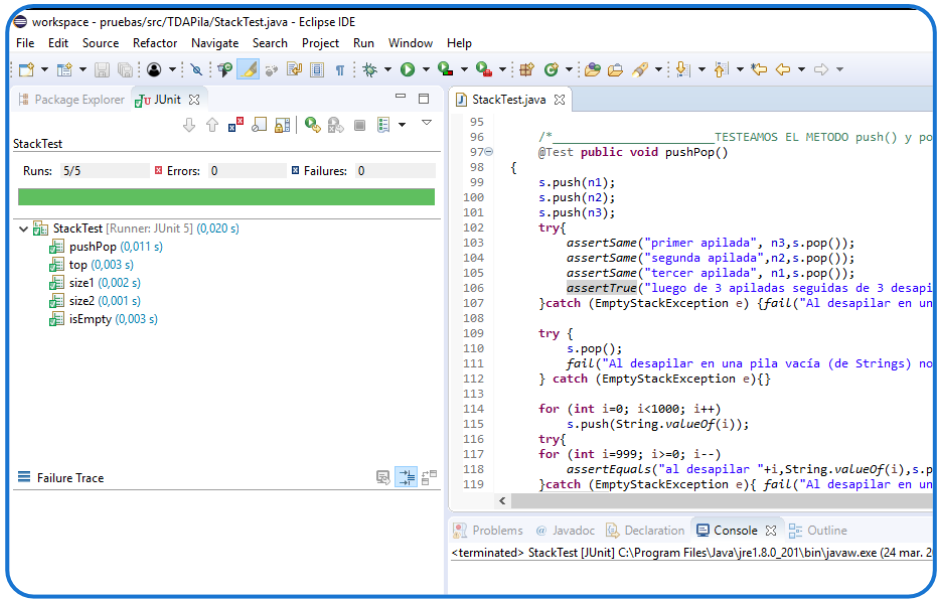


¿Cómo utilizar los test definidos en JUnit?

- Luego, la clase que define el test debe **colocarse** dentro del **proyecto**. Considerar por ejemplo cómo utilizar la clase ***StackTest.java*** definida por la cátedra:
- Copiar y pegar ***StackTest.java*** en el paquete **TDAPila**.
- Modificar en el método ***getStack()*** el **nombre** de la **clase a testear** (colocar el nombre de la clase que se desarrolló e implementa a **Stack**).
- Ejecutar el test con ***Run***.







EXPORTAR E IMPORTAR CÓDIGO JAVA: JAR

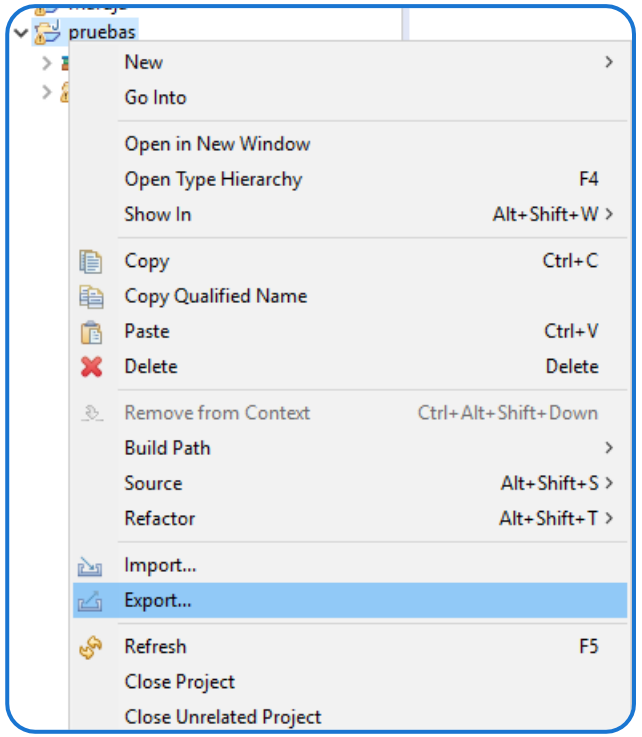
Se recomienda que todo lo documentado en la siguiente sección sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

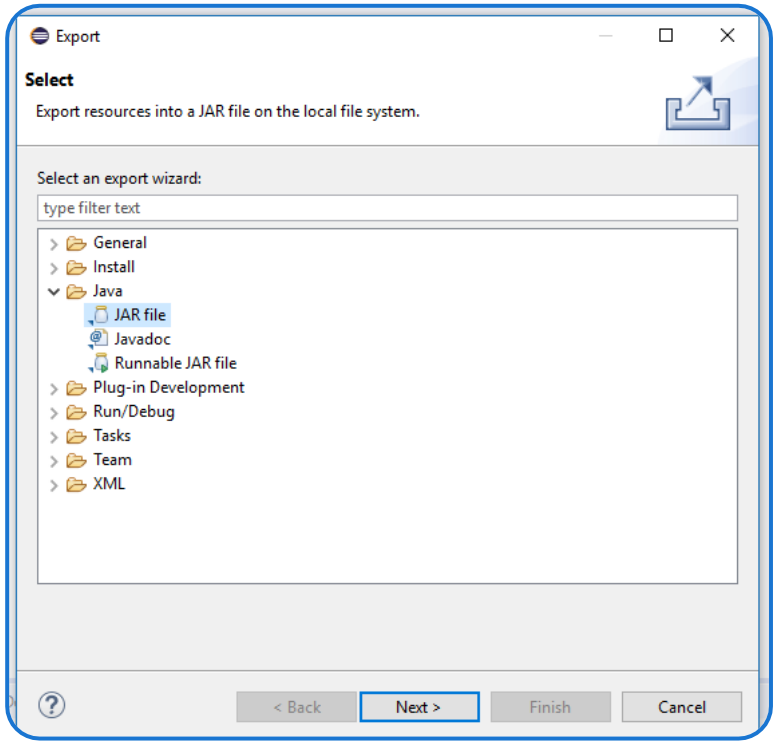
Introducción

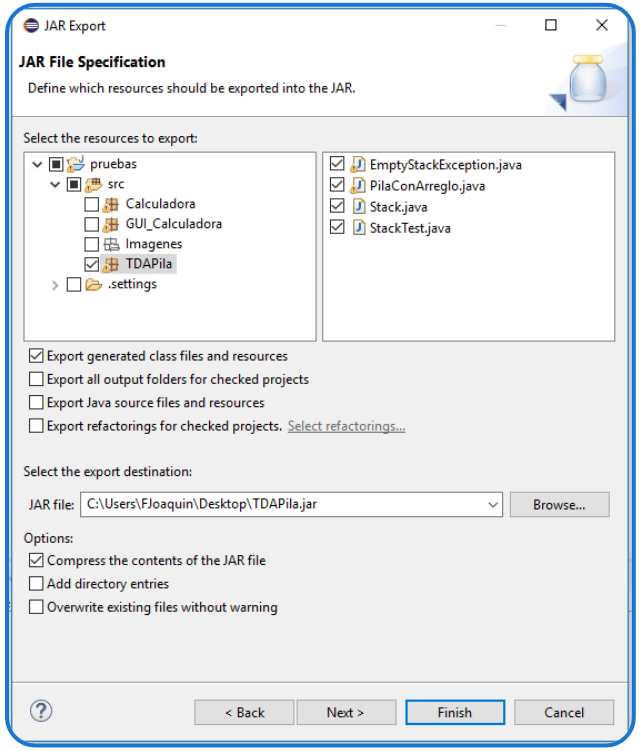
- La **reutilización** del código es una parte **crucial** del **desarrollo** de **software**.
- Muchas veces **no se dispone** del **código fuente** de determinadas funcionalidades, pero por el contrario, se dispone de **librerías** que **encapsulan** y **permiten** utilizar dicha funcionalidad.
- Para **permitir** el uso de funcionalidad desarrollada en Java de la que no se dispone del código fuente, se utilizan las **librerías externas**.
  - A veces **se usarán** desarrollos en Java empaquetados en un librería externa que hay que **importar**.
  - A veces **se desarrollará** código Java que, una vez finalizado, se ofrecerá de forma empaquetada como una librería que debe ser **exportada**.

Exportando funcionalidad

- Para exportar funcionalidad, se deben seguir los siguientes pasos:
  - Click derecho sobre el *proyecto/paquete/clase* a exportar → *Exportar*.
  - Seleccionar *JAR File* → *Next* → Seleccionar *destino* → *Finish.*

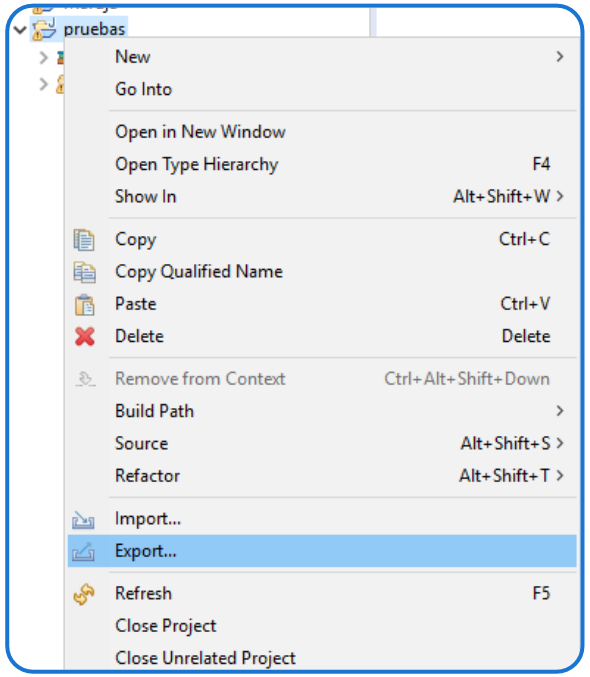


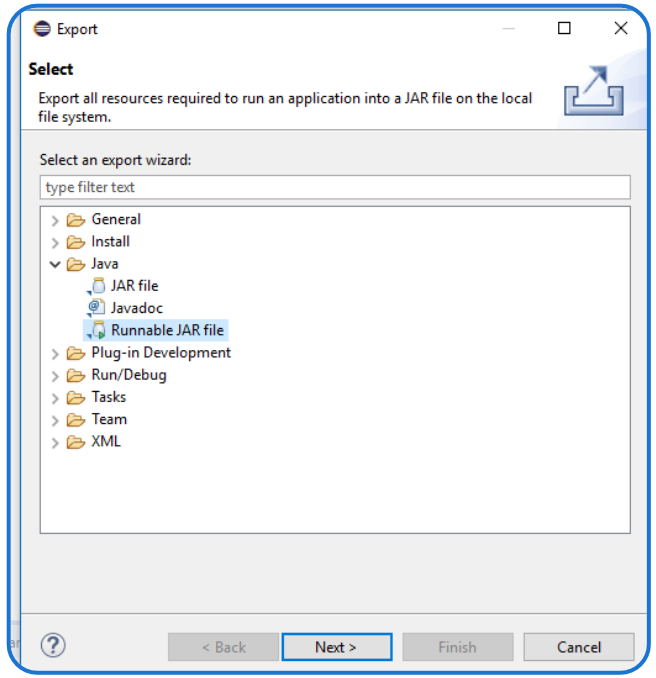


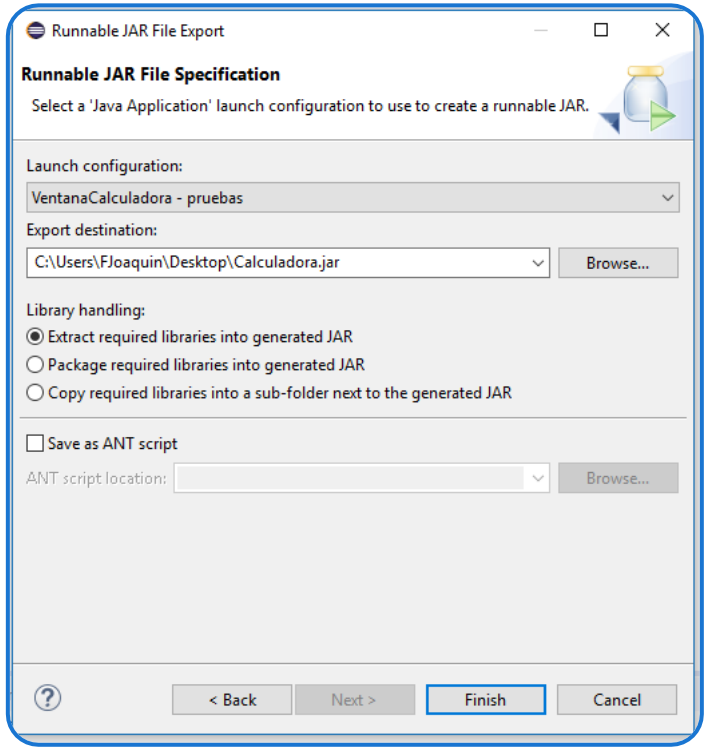


Exportando funcionalidad como JAR ejecutable

- Para exportar funcionalidad como JAR ejecutable, hacer:
  - Click derecho sobre el *proyecto/paquete/clase* a exportar → *Exportar*.
  - Seleccionar *Runnable JAR File* → *Next* → Seleccionar *destino* e indicar la clase *Launcher* (*lanzador/main*) → *Finish.*

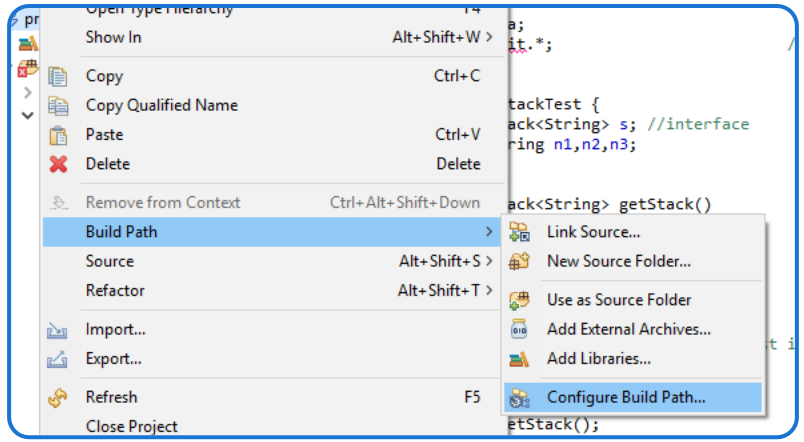


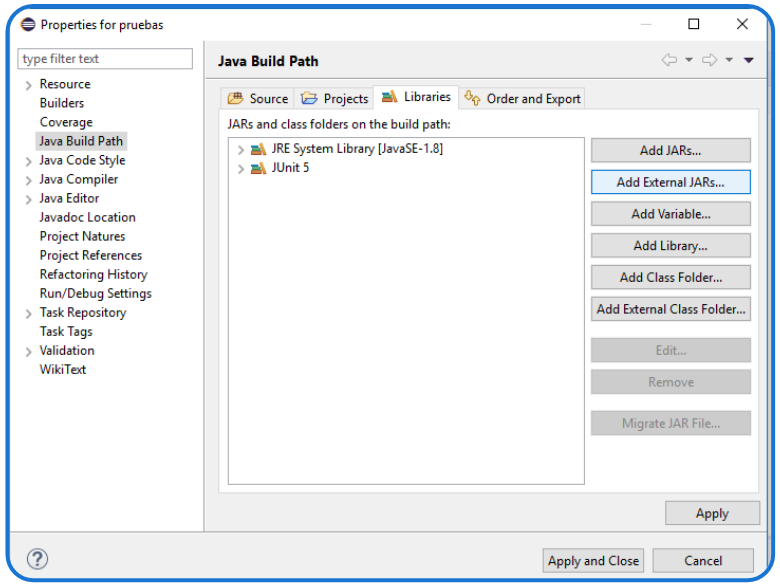


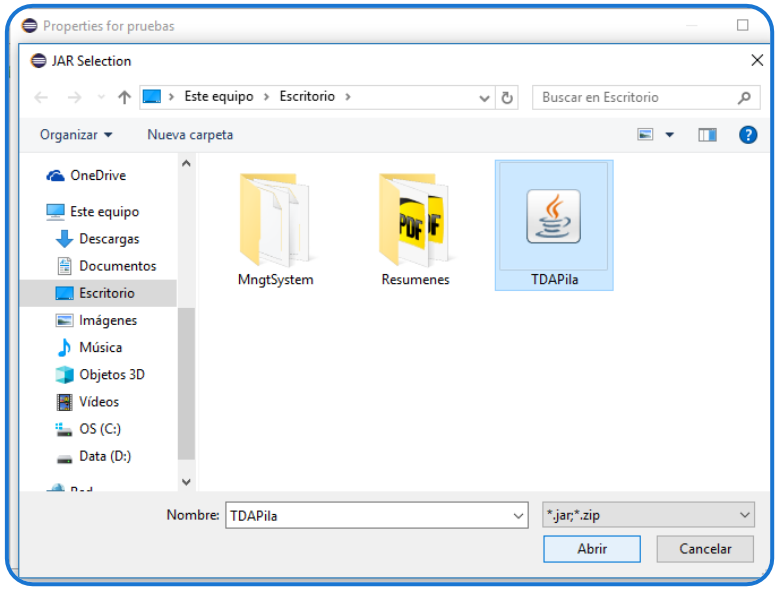


Importando funcionalidad

- Para importar funcionalidad, se deben seguir los siguientes pasos:
  - Click derecho en el *proyecto* → *Build Path* → *Configure Build Path..*
  - En la pestaña *Libraries* → *Add External JARs…* → Seleccionar *JAR* → *Apply.*







Fin de la presentación.