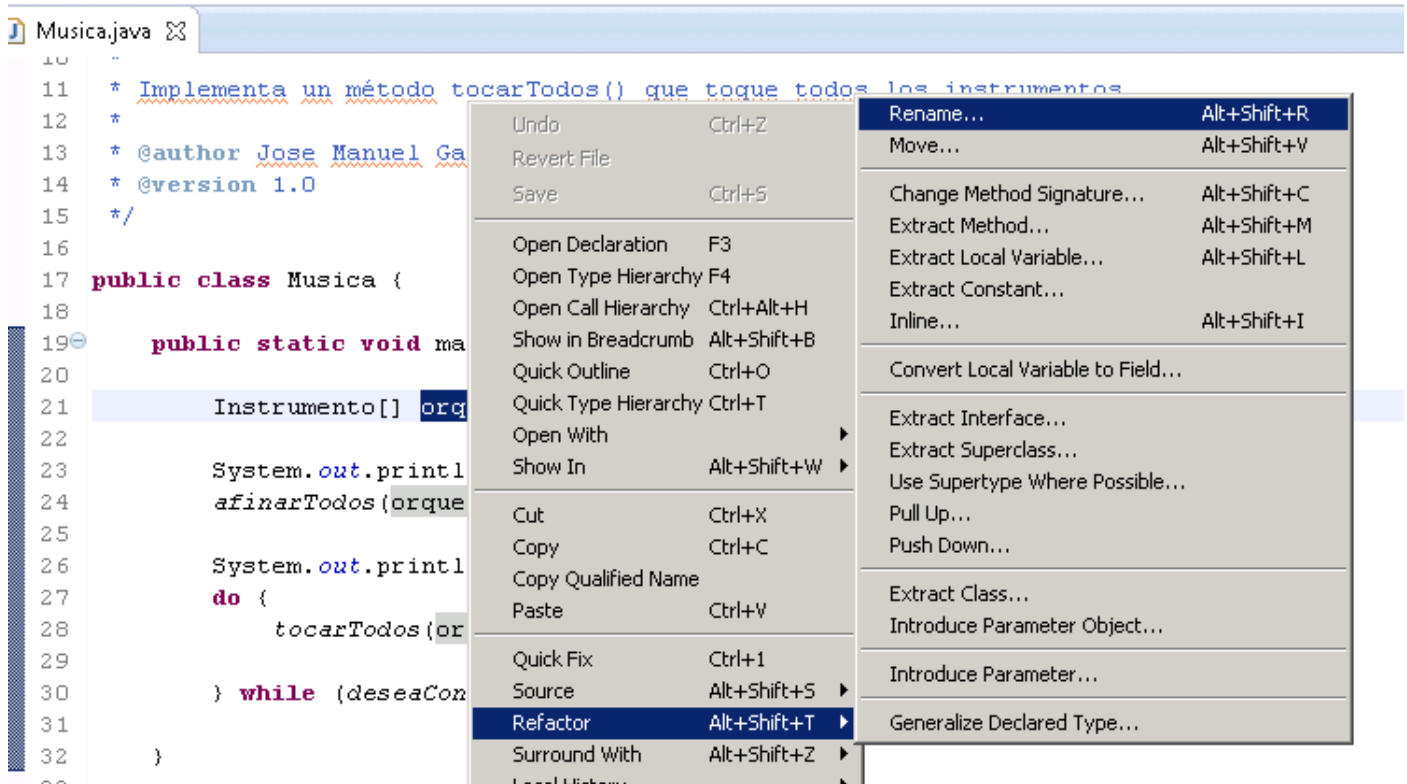


Alumno/a: José Manuel García Valverde

[Refactorizacion-3]

Rename: Consiste en cambiar el nombre a una variable, método, clase, paquete, directorio, etc. En este caso refactorizaremos renombrando el identificador de una variable.

Seleccionamos lo que vamos a renombrar (orquesta)



Shift + Alt + R

Refactor + Rename

Boton derecho + Refactor + Rename

```
public static void main(String[] args) {
    Instrumento[] orquesta = crearOrquesta();

    System.out.println("Instrumentos: ");
    afinarTodos(orquesta);

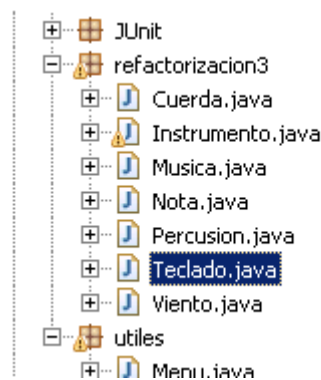
    System.out.println("A tocar: ");
    do {
        tocarTodos(orquesta, obtenerNota());
    } while (deseaContinuar());
}
```

Resultado:

```
public static void main(String[] args) {  
  
    Instrumento[] grupoMusical= crearOrquesta();  
  
    System.out.println("Afinamos todos los instrumentos: ");  
    afinarTodos(grupoMusical);  
  
    System.out.println("A tocar: ");  
    do {  
        tocarTodos(grupoMusical, obtenerNota());  
  
    } while (deseaContinuar());  
}
```

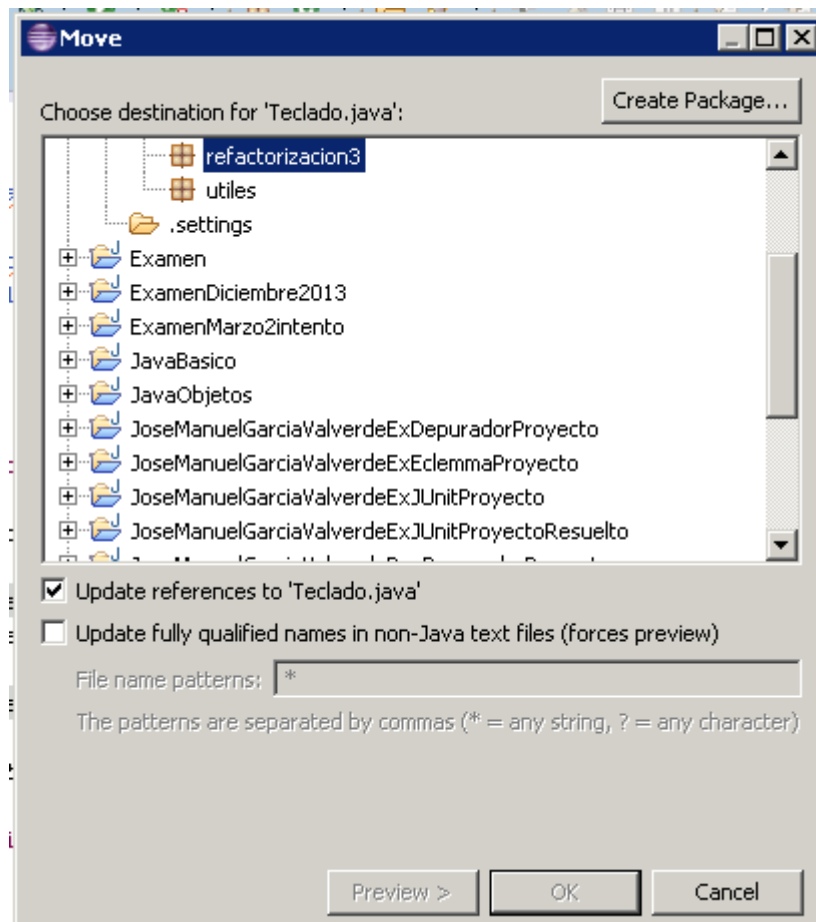
Move: Podemos mover distintos “elementos” como por ejemplo clases a otros paquetes o métodos a otras clases.

Seleccionamos lo que vamos a mover



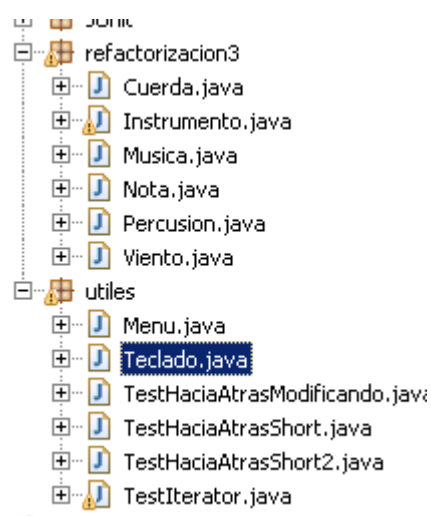
Shift + Alt +L

Refactor + Move



Elegimos donde lo vamos a mover

Resultado:



Extraer variable local: Sustituir una expresión que se repita por una variable local. El resultado es que dicha expresión será sustituida en siempre que aparezca pero de forma local, no sufrirá cambios por ejemplo en otros métodos.

Seleccionamos la expresión a extraer:

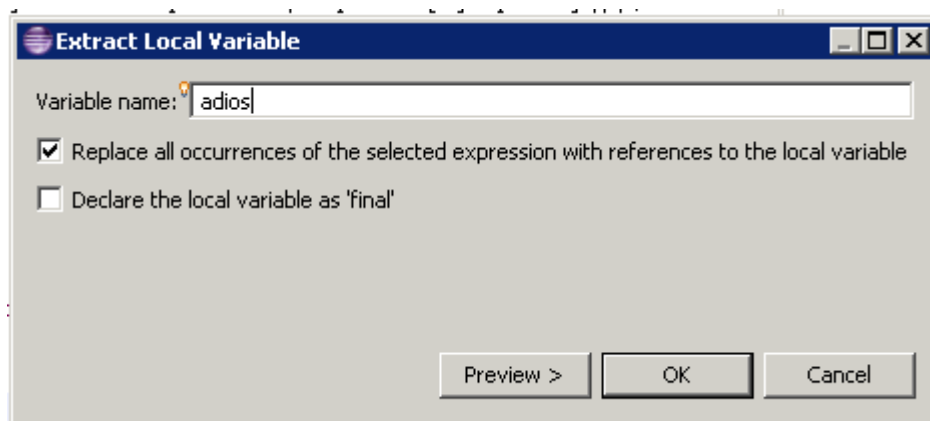
```

    private static boolean deseaContinuar() {
        String cadena = Teclado.leerCadena(
            "{Quieres que siga tocando la orquesta?"
        );
        if (cadena.equals("SI")) {
            return true;
        } else{
            System.out.println("ADIOS");
            return false;
        }
    }

```

Shift + Alt + L

Refactor + Extract Local Variable



Resultado

```

        return true;
    } else{
        String adios = "ADIOS";
        System.out.println(adios);
        return false;
    }

```

Convertir variable local en campo: Convertir una variable local de un método en un campo de la clase.

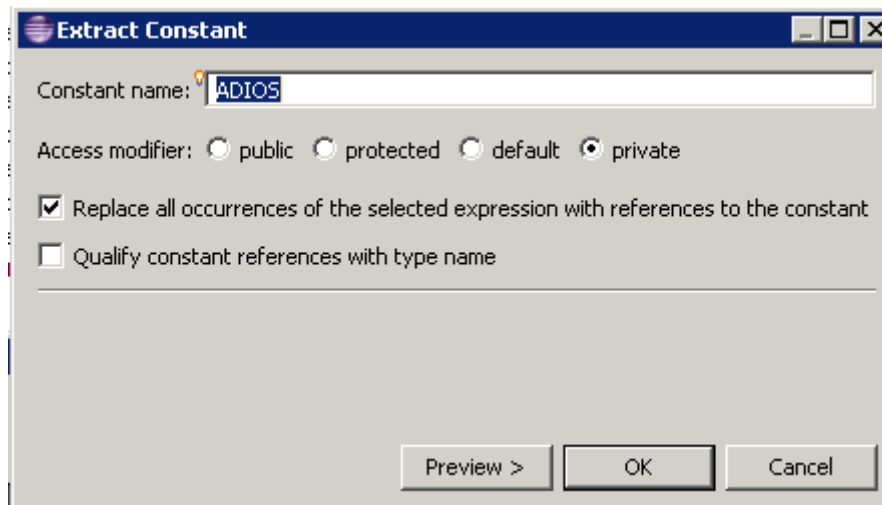
Seleccionamos la variable

```

    if (cadena.equals("SI")) {
        return true;
    } else{
        System.out.println("ADIOS");
        return false;
    }

```

Rename + Convert local variable to field...



Resultado:

```
private static final String ADIOS = "ADIOS";
```

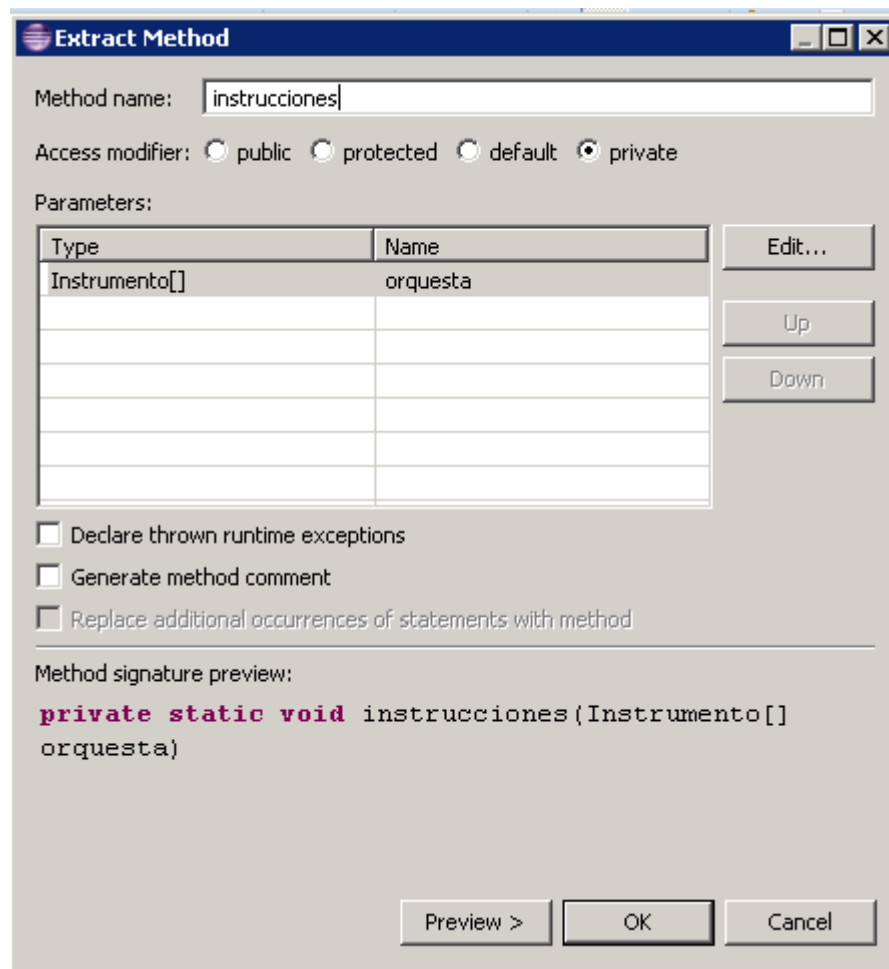
Extraer metodo: Crear de forma automática un método que contenga una serie de instrucciones que se repiten en distintos lugares del código y sustituir las repeticiones por llamadas a dicho método.

Seleccionamos el código que queremos hacer que sea un método.

```
System.out.println("Afinamos todos los instrumentos: ");  
afinarTodos(orquestra);  
  
System.out.println("A tocar: ");  
do {  
    tocarTodos(orquestra, obtenerNota());  
} while (deseaContinuar());
```

Shift + Alt + M

Refactor + Extract Method...



Resultado:

```

instrucciones(orquesta);
do {
    tocarTodos(orquesta, obtenerNota());

} while (deseaContinuar());

}

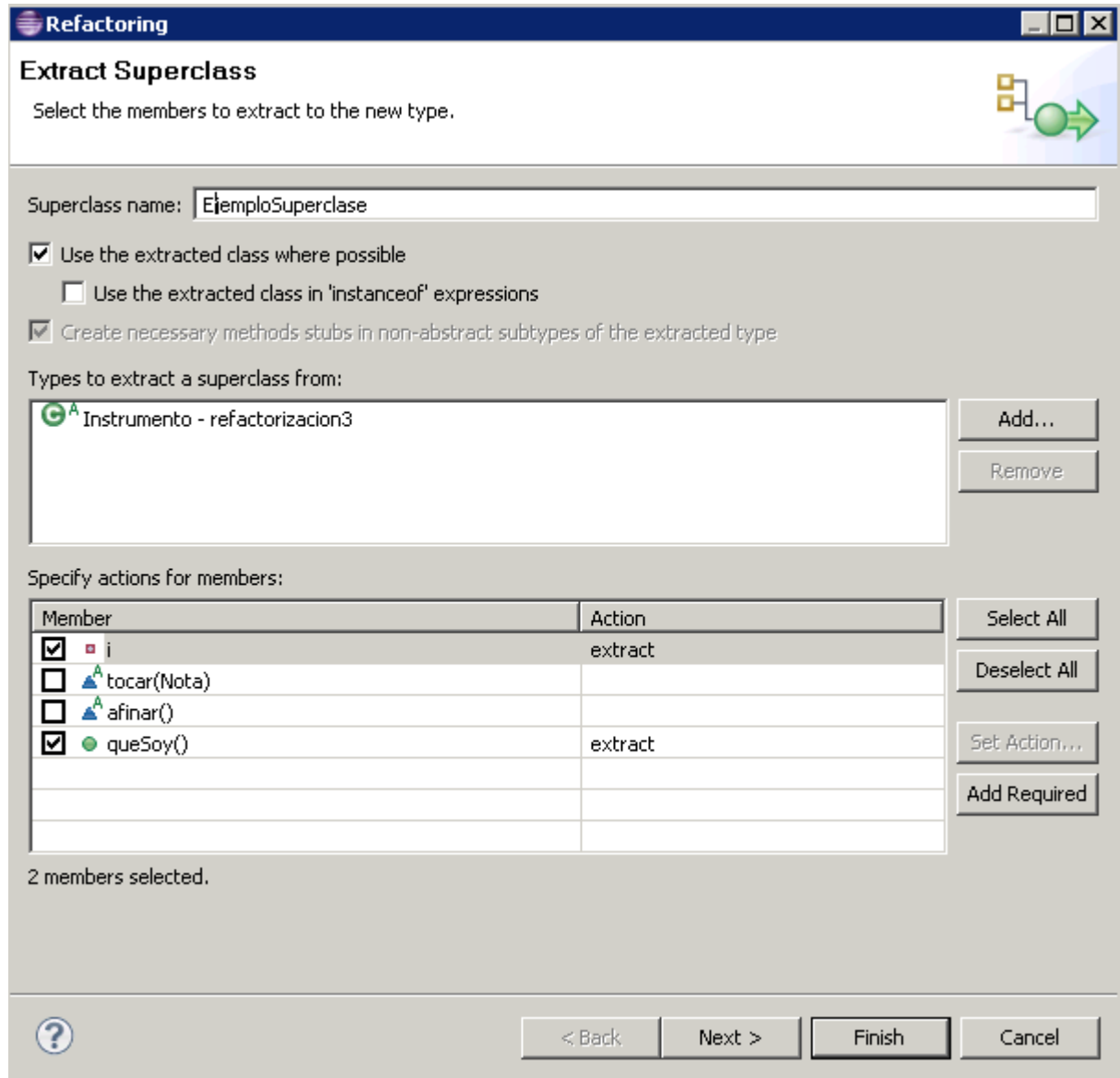
private static void instrucciones(Instrumento[] orquesta) {
    System.out.println("Afinamos todos los instrumentos: ");
    afinarTodos(orquesta);

    System.out.println("A tocar: ");
}

```

Extraer superclase: Consiste en crear una nueva clase con algunos metodos y campos de otra que ya tengamos.

Refactor + extract superclass...



Refactoring

Extract Superclass

Select the members to extract to the new type.

Superclass name:

☒ Use the extracted class where possible
☐ Use the extracted class in 'instanceof' expressions
☒ Create necessary methods stubs in non-abstract subtypes of the extracted type

Types to extract a superclass from:

Instrumento - refactorizacion3

Add...
Remove

Specify actions for members:

| Member | Action |
|--|---------|
| <input checked="" type="checkbox"/> i | extract |
| <input type="checkbox"/> tocar(Nota) | |
| <input type="checkbox"/> afinar() | |
| <input checked="" type="checkbox"/> queSoy() | extract |
| | |
| | |

Select All
Deselect All
Set Action...
Add Required

2 members selected.

? < Back Next > Finish Cancel

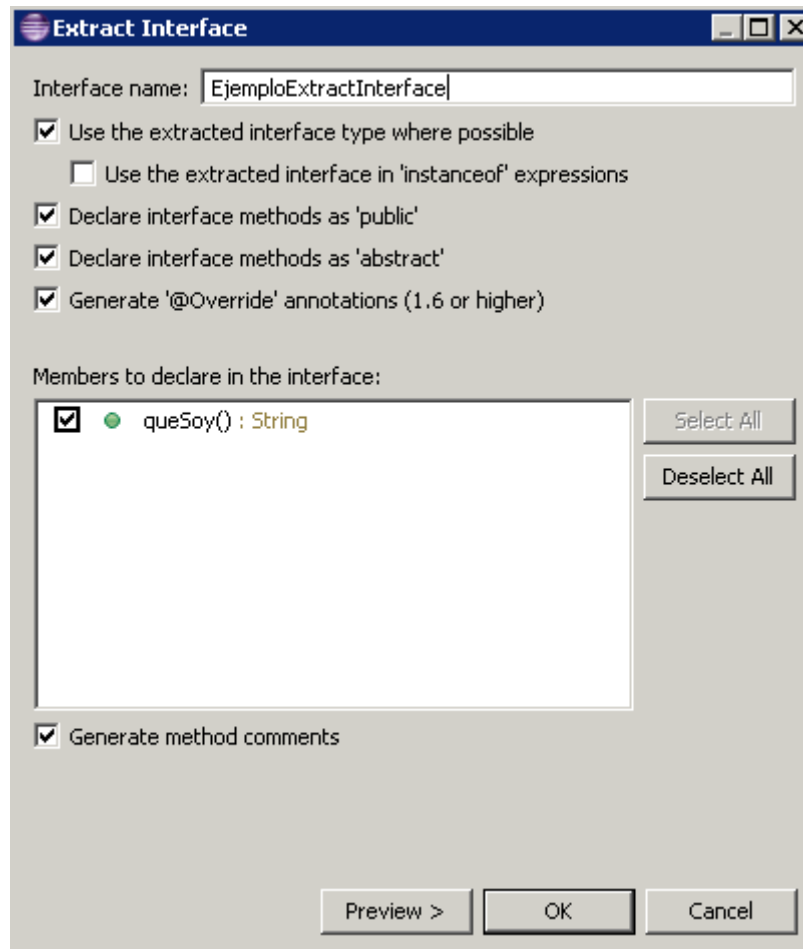
Resultado:

```
public abstract class Instrumento extends EjemploSuperclase {  
    abstract void tocar(Nota nota);  
  
    abstract void afinar();  
}
```

```
public class EjemploSuperclase {  
  
    private String i;  
  
    public EjemploSuperclase() {  
        super();  
    }  
  
    public String queSoy() {  
        return "Instrumento";  
    }  
}
```


Extraer interface: Consiste en crear una nueva interfaz con algunos métodos de otra interfaz.

Refactor + Extract interface...



Resultado:

```
public abstract class Instrumento implements EjemploExtractInterface {  
    private String i;  
  
    abstract void tocar(Nota nota);  
  
    abstract void afinar();  
  
    /* (non-Javadoc)  
     * @see refactorizacion3.EjemploExtractInterface#queSoy()  
     */  
    @Override  
    public String queSoy() {  
        return "Instrumento";  
    }  
}
```

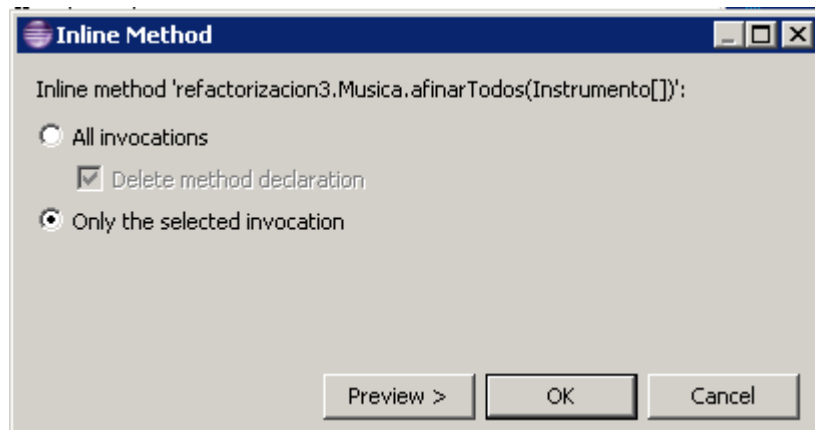
Incorporar: Conlleva sustituir la referencia a un método o variable con la aplicación del método o el valor de la variable. Es útil cuando una expresión queda más clara y limpia en una sola línea o cuando un método es llamado una vez nada más y tiene más sentido como un bloque de código.

Seleccionamos el elemento a incorporar:

```
System.out.println("Afinamos todos los instrumentos: ");  
afinarTodos(orquesta);  
  
System.out.println("A tocar: ");  
do {  
    tocarTodos(orquesta, obtenerNota());  
}
```

Shift + Alt + I

Refactor + Inline...



Resultado:

```
for (int i = 0; i < orquesta.length; i++) {  
    orquesta[i].afinar();  
}
```

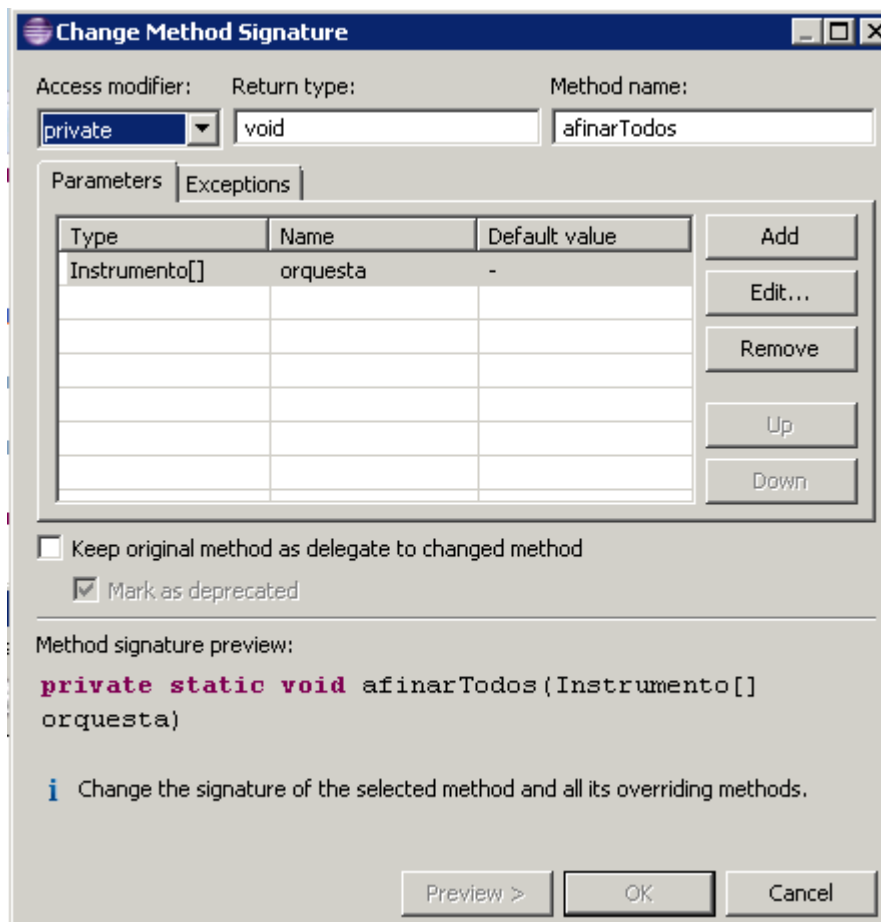
Cambiar signatura de un metodo: Nos permite cambiar los siguientes elementos de un método: modificador de acceso, tipo de retorno, nombre del método, parámetros y excepciones.

Seleccionamos el metodo a modificar:

```
private static void afinarTodos(Instrumento[] orquesta) {  
    for (int i = 0; i < orquesta.length; i++) {  
        orquesta[i].afinar();  
    }  
}
```

Shift + Alt + C

Refactor + Change Method Signature...



Resultado:

```


    */
    public static void ejemploCambiarSignatura(Instrumento[] orquesta) {
        for (int i = 0; i < orquesta.length; i++) {
            orquesta[i].afinar();
        }
    }


```

Inferir argumentos del tipo generico: Nos permite inferir el tipo correcto de los argumentos para una clase. +

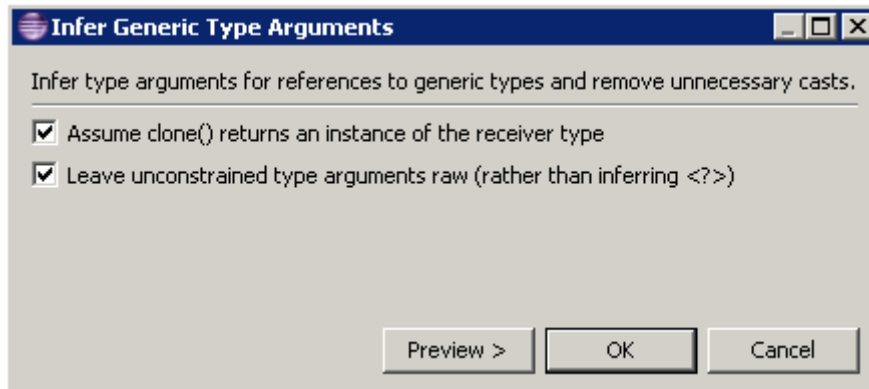
Seleccionamos lo que vamos a modificar:

```


    public static void main(String[] args) {
        ArrayList ejemploInfer = new ArrayList();
        ejemploInfer.add(new Cuerda());
    }


```

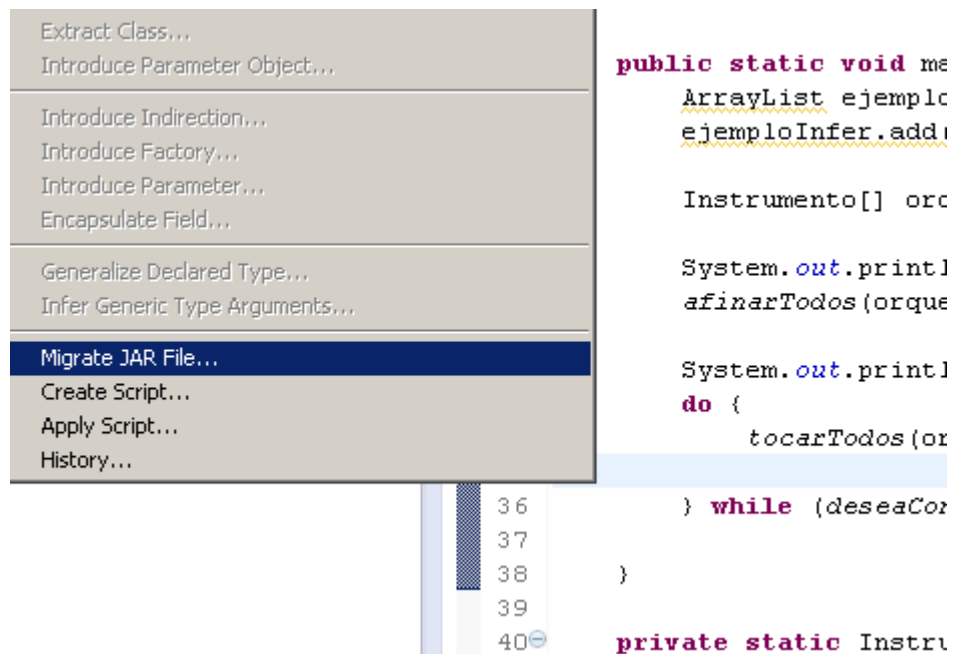
Refactor + Infer Generic Type Arguments...



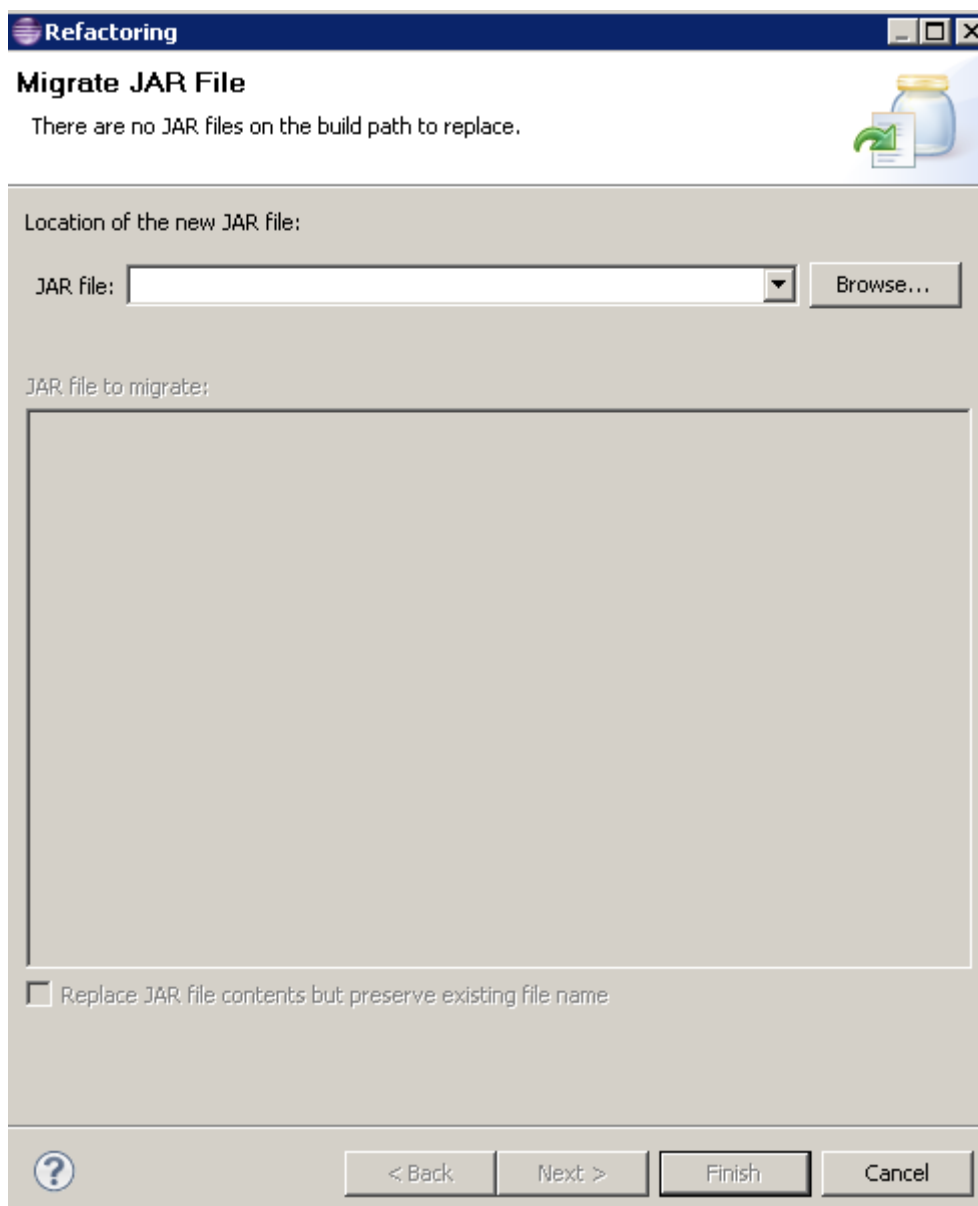
Resultado:

```
ArrayList<Cuerda> ejemploInfer = new ArrayList<Cuerda>();
ejemploInfer.add(new Cuerda());
```

Migrar archivo JAR: Mueve un archivo JAR que se encuentre en la ruta de un proyecto de nuestro workspace a una nueva versión. Esto permite actualizar el archivo en la ruta de un proyecto de una versión mayor. Al refactorizar, el anterior JAR es eliminado y se copia uno nuevo en la ruta del proyecto.

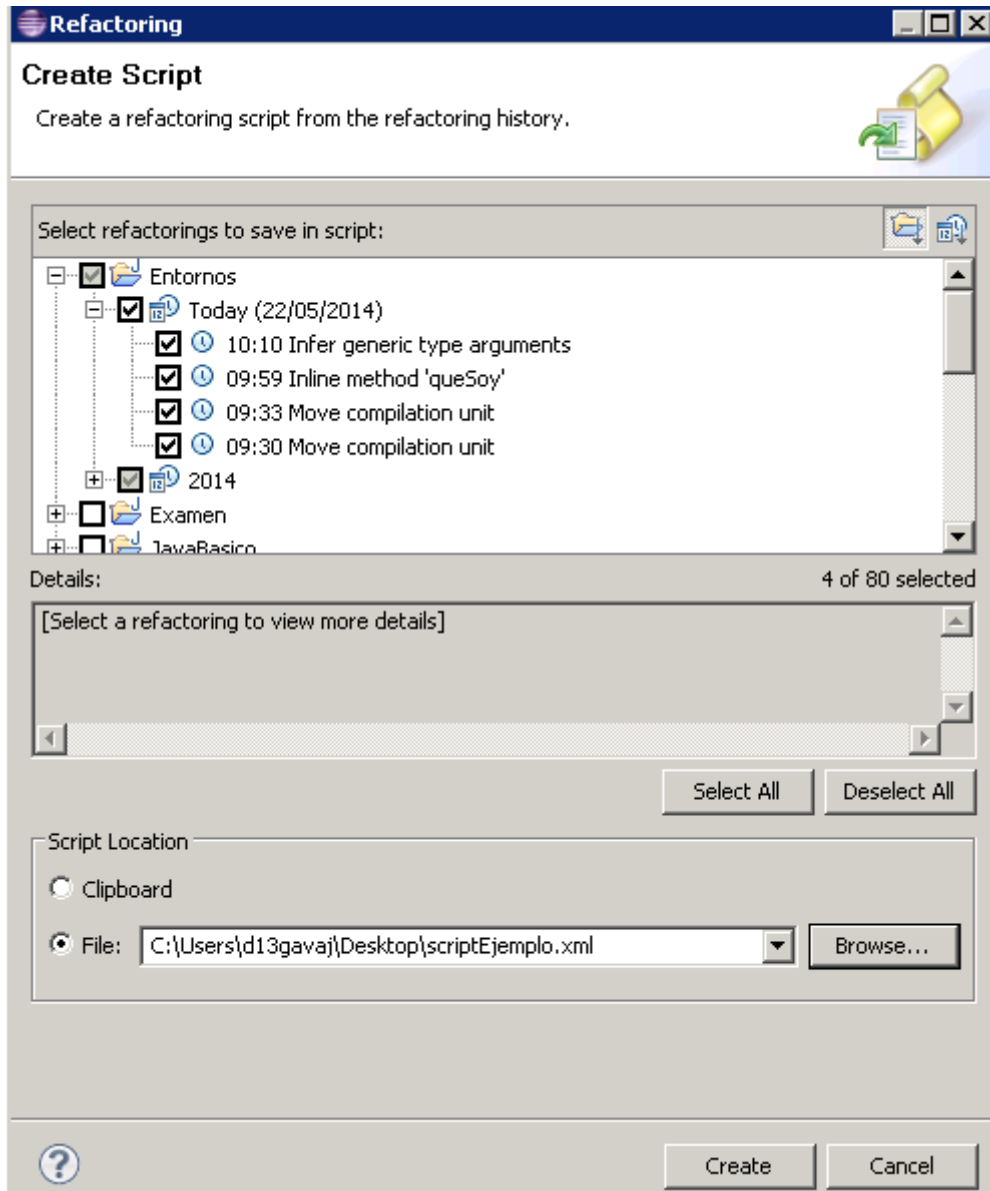


Elegimos el archivo JAR al que queremos migrar y finish:



Crear Script: Nos ofrece una forma de exportar las acciones de refactorización que deseemos para poder utilizarlas de nuevo cuando y donde queramos.

Refactor + Create Script



Resultado:

```
<?xml version="1.0" encoding="UTF-8"?>
<session version="1.0">
  <refactoring comment="Move element &apos;Teclado.java&apos; to &apos;Entornos/src/JUnit&apos; - Original project: &apos;Entornos&apos; - Original element: &apos;Entornos/src/JUnit&apos; - Update references to refactored element" description="Move compilation unit" destination="/src&lt;Unit" element1="/src&lt;utils{Teclado.java" files="0" flags="589830" folders="0" id="org.eclipse.jdt.ui.move" patterns="" policy="org.eclipse.jdt.ui.moveResources" project="Entornos" qualified="false" references="true" units="1" version="1.0"/>
  <refactoring comment="Move element &apos;Teclado.java&apos; to &apos;Entornos/src/utiles&apos; - Original project: &apos;Entornos&apos; - Original element: &apos;Entornos/src/utiles&apos; - Update references to refactored element" description="Move compilation unit" destination="/src&lt;utiles" element1="/src&lt;refactorizacion3{Teclado.java" files="0" flags="589830" folders="0" id="org.eclipse.jdt.ui.move" patterns="" policy="org.eclipse.jdt.ui.moveResources" project="Entornos" qualified="false" references="true" units="1" version="1.0"/>
  <refactoring comment="Inline method &apos;refactorizacion3.Instrumento.queSoy()&apos; in &apos;refactorizacion3.Instrumento&apos; - Original project: &apos;Entornos&apos; - Original element: &apos;refactorizacion3.Instrumento.queSoy()&apos; - Replace all references to method with statements" delete="false" description="Inline method &apos;queSoy&apos;" flags="786438" id="org.eclipse.jdt.ui.inline.method" input="/src&lt;refactorizacion3{Instrumento.java" mode="1" project="Entornos" selection="528 53" version="1.0"/>
  <refactoring clones="true" comment="Infer generic type arguments on &apos;Entornos&apos; - Original project: &apos;Entornos&apos; - Original element: &apos;Entornos&apos; - Assume clone() returns an instance of the receiver type&apos; - Leave unconstrained type arguments raw" description="Infer generic type arguments" element1="/src&lt;refactorizacion3{Musica.java" flags="6" id="org.eclipse.jdt.ui.infer.typearguments" leave="true" project="Entornos" version="1.0"/>
</session>
```

*Para utilizarlo solo tenemos que ir a Refactor + Appy Script y elegir el script.