

Más allá del IF y del WHILE

Aprender a programar software
a base de ejercicios y ejemplos.

Juan Murua Olalde

19/Julio/2020
8 de septiembre de 2022

Nota: Una copia .pdf de este manual se puede descargar desde www.susosise.es

Nota: El código fuente y el historial de cambios de este documento se puede obtener en

https://bitbucket.org/susosise/aprender_a_programar_software/commits/



<https://creativecommons.org/licenses/by-sa/4.0>

Índice general

1. Técnicas de trabajo	6
1.1. Control de versiones	6
1.2. Test unitarios	12
1.3. Refactorizar	19
2. Interfaz gráfico de usuario: GUI (Graphic User Interface)	21
2.1. Formas de construirlos	21
2.1.1. Indicando ubicaciones y dimensiones fijas	21
2.1.2. Indicando relaciones y recomendando dimensiones	21
2.2. Event Loop , Event Dispatcher ::: Reaccionar a lo que sucede en la pantalla, en el ratón o en el teclado	22
2.3. Responsive GUI ::: Reaccionar a lo que sucede en el interior de la aplicación	22
2.4. Elementos habituales en los GUIs	22
2.4.1. Controles con los que el usuario puede interactuar	22
2.4.2. Distribuciones generales con las que organizar la pantalla	23
3. Ejercicios	25
3.1. Hello, Benzirpi.	25
3.2. Bocatas	26
3.3. Selección plana	26
3.4. Diálogos estándares	26
3.5. Scroll	27
3.6. Split	28
3.7. Calendario	29
3.8. Selección arbórea	29
3.9. Barra de progreso	29

3.10. Internacionalización (i18n) y localización (L10n)	30
3.11. Ayudas y documentación	30
4. Objetos: clases e instancias	32
4.1. Encapsulación	32
4.2. Polimorfismo	32
4.3. Herencia	32
4.4. Sobrecarga de funciones (overwrite)	32
4.5. Interface (clase abstracta)	33
4.6. Cuándo usar herencia y cuándo interface	33
4.7. Factorias para “fabricar” objetos (inyección de dependencias)	34
5. Ejercicios	36
5.1. Áreas y perímetros de figuras geométricas	36
5.1.1. Utilizando herencia	36
5.1.2. Utilizando interface	36
5.2. Ascensores	37
6. Datos en memoria de trabajo	38
6.1. Estructuras lineales (arrays, colas,...)	38
6.2. Listas (list, vector,...)	40
6.3. Diccionarios (dictionary, map,...)	43
6.4. Estructuras arbóreas (trees) y grafos (graphs)	44
6.5. Bases de datos	44
6.6. La importancia de especificar el tipo de los datos: “generics”	45
6.7. Las propias funciones como un tipo de dato más: “expresiones Lambda”	46
7. Datos almacenados (persistencia)	47
7.1. Archivos de texto “plano”	47
7.2. Expresiones regulares	48
7.3. Archivos XML	49
7.4. Archivos JSON	50
7.5. Archivos CBOR u otras representaciones binarias	50
7.6. Bases de datos	51
8. Ejercicios	52

8.1. Cuentas bancarias	53
8.2. Gestión de accesos	54
8.3. Gestión de pedidos	55
9. Tratamiento de incidencias y errores	56
9.1. Excepciones	56
9.2. Registro (log)	57
9.3. Ejercicios	58
10. Comunicaciones	59
10.1. Usando protocolo HTTP	59
10.2. Usando protocolos tcp/ip	60
10.3. Usando protocolos especializados para un tipo de tarea concreta . . .	60
11. Ejercicios	61
12. Procesamiento paralelo	62
12.1. Estrategias para repartir tareas	63
12.2. Salvaguardas para evitar problemas	63
12.2.1. operaciones atómicas	63
12.2.2. transacciones	63
12.2.3. semáforos	64
12.2.4. bloqueos	64
12.2.5. Programación funcional	65
12.3. Algunos usos del procesamiento paralelo	65
12.3.1. Multitarea	65
12.3.2. Gráficos	66
12.3.3. Cálculos	66
13. Ejercicios	67
14. Organización interna del código	68
14.1. Orientación a eventos (event-driven)	68
14.2. Orientación a objetos (object-oriented)	69
14.3. Arquitecturas multi-capa (n-tier)	70
14.4. Aplicaciones distribuidas	76

14.5. Orientación a servicios (SOA, SaaS, microservices,...)	77
14.6. Orientación a servicios, arquitectura REST (REpresentational State Transfer)	77
15.Ejercicios	79
15.1. Multiinterface	79
15.2. Arquitectura en tres capas, MVC	80
15.3. Excursiones	80
15.4. Hotel	81
16.apéndice A: ejemplos de posibles herramientas de trabajo	89
16.1. Para programar en Java	89
16.2. Para programar en C++	90
16.3. Para programar en C#	90
16.4. Para programar en Python	91
16.5. Para programar en HTML5/CSS/Javascript	92
16.6. Una herramienta transversal: almacenamiento de código con control de versiones	93
16.7. Otra herramienta transversal: motor de base de datos	93
17.apéndice B: ejemplos de posibles soluciones...	94
17.1. ...a los ejercicios del capítulo 3	94
17.2. ...a los ejercicios del capítulo 5	137
17.3. ...a los ejercicios del capítulo 8	147
17.4. ...a los ejercicios del capítulo 11	147
17.5. ...a los ejercicios del capítulo 13	147
17.6. ...a los ejercicios del capítulo 15	148

Requisitos previos

Este libro asume que se está familiarizado y se tiene soltura manejando los aspectos básicos del lenguaje que se vaya a utilizar para realizar sus ejercicios.

Los aspectos que se dan por conocidos son:

- Un entorno sobre el que programar, con las herramientas básicas de: editor, compilador (o intérprete) y depurador.
- Ejecución de programas, cuál es el punto de arranque (main).
- Depuración de programas, cómo colocar puntos de interrupción (breakpoints) y cómo seguir el programa paso a paso visualizando valores en las variables escogidas.
- Declaración de variables para almacenar distintos tipos de datos. Asignación de valores a variables. Conversión de unos tipos de datos en otros (cast).
- El concepto de alcance (scope): saber en qué parte del código “vive” (es válida) una variable o función.
- Interfaz textual con el usuario: cómo mostrar algo en pantalla y cómo recoger lo que el usuario teclee.
- Las construcciones básicas para gobernar el flujo de ejecución: if/else, for, while, switch,...
- La construcción básica para manejar colecciones de valores: array
- La construcción básica para definir valores predefinidos: enum
- El almacenamiento básico de datos (persistencia): escribir y leer archivos de texto.
- Estructuración básica de una aplicación en funciones y módulos (programación modular).

Estos conceptos se contemplan en todos los libros o cursos introductorios de todos los lenguajes de programación. Algunos de esos recursos se citan en <https://www.susosise.es/documentos/ElCodigoFuenteNoMuerde.pdf>

Capítulo 1

Técnicas de trabajo

Estas técnicas son muy recomendables para realizar cualquier trabajo de programación. Por eso se describen aquí, al principio del libro. Con el ánimo de que se aprendan desde un inicio y se vayan aplicando, en la medida de lo posible, en todos los ejercicios que se vayan haciendo.

Practicar es la mejor manera de adquirir confianza en el uso de cualquier herramienta.

1.1. Control de versiones

Un repositorio con control de versiones da mucha tranquilidad a la hora de escribir código. Cada paso que se va dando, va quedando registrado. Y así:

- En todo momento se sabe cuál es la última versión válida.
- Se puede consultar cuándo se cambió qué.
- Se puede recuperar un estado anterior si fuese necesario.
- Varias personas pueden trabajar colaborativamente sobre un mismo código.
- Las copias de seguridad son más sencillas de mantener siempre al día.

Aviso: Un sistema de control de versiones como Git requiere de bastante experiencia de trabajo con él para sentirse cómodo. No recomendaría comenzar a utilizarlo alegremente en un proyecto importante. Mejor practicar primero en proyectos personales, donde nos podamos permitir “trastear” y aprender cuando algo no vaya como esperábamos que fuera.

Breve introducción a Git y Sourcetree

Aquí se dan cuatro pinceladas para comenzar a trabajar con Git a través de Sourcetree (mi favorito). (Se puede utilizar cualquier otro interface para trabajar con Git.) Para más detalles, acudir a las web oficiales:

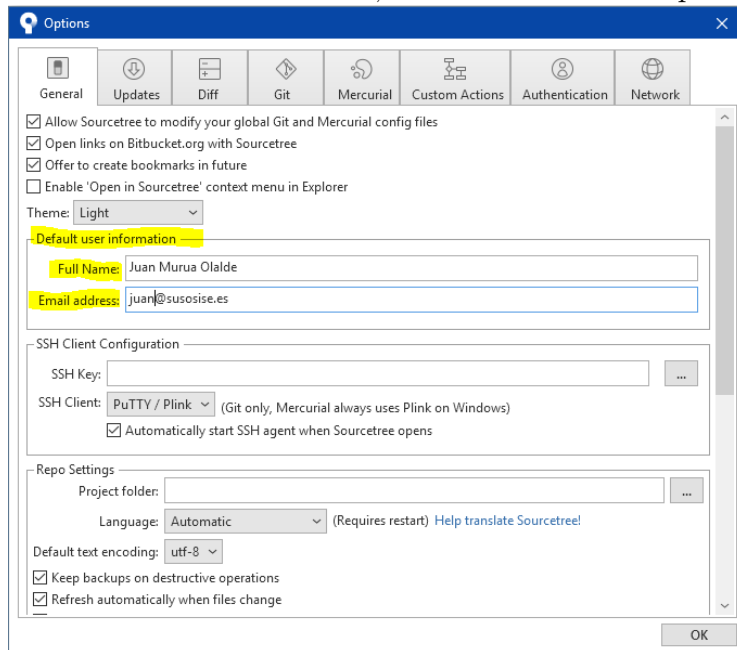
<https://git-scm.com/doc>

<https://www.atlassian.com/git>

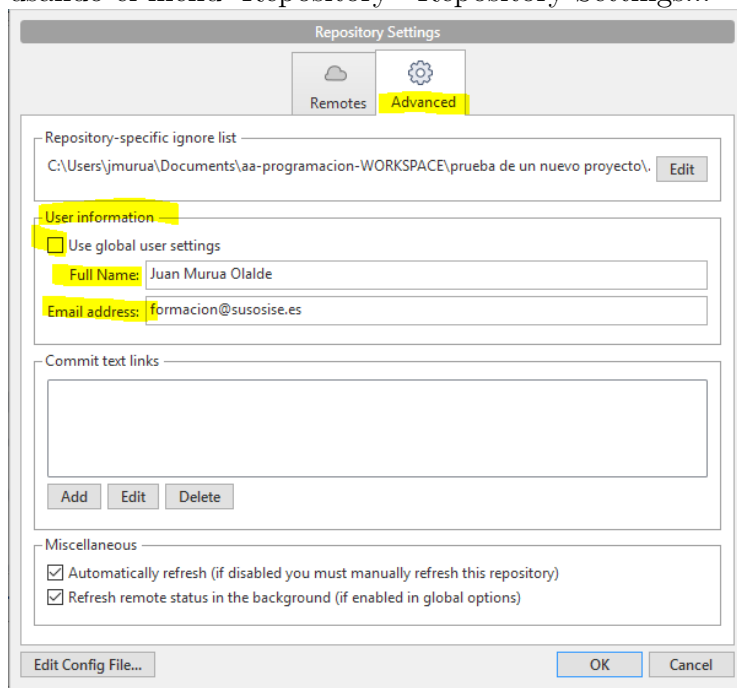
<https://confluence.atlassian.com/get-started-with-sourcetree>
o a otras fuentes de información.

¡Importante!, antes de comenzar a trabajar tenemos que configurar nuestros datos identificativos. Todos los cambios (commits) que guardemos desde ese PC, irán identificados con ellos.

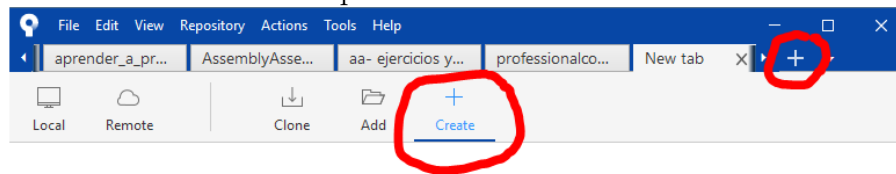
Para hacerlo en Sourcetree, ir al menú 'Tools' 'Options':



Nota: También es posible indicar unos datos distintos solo para un proyecto concreto, usando el menú 'Repository' 'Repository Settings...'



Para comenzar a trabajar. Crear un nuevo repositorio para un proyecto. Mejor hacerlo en una nueva carpeta.



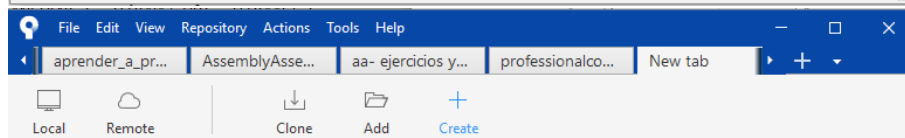
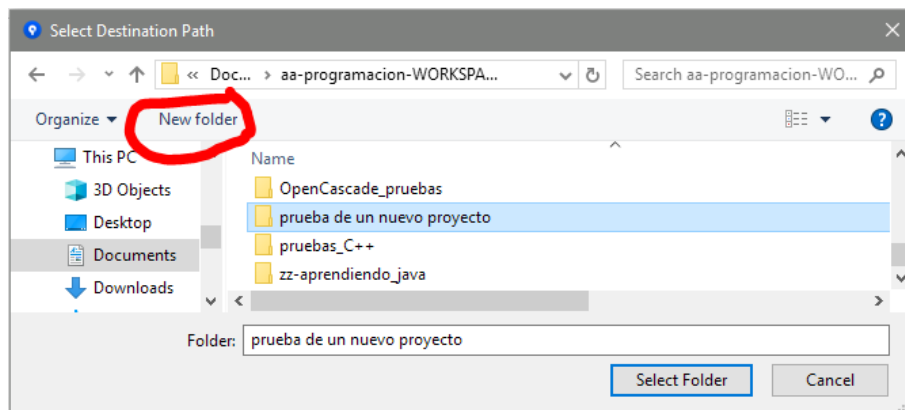
Create a repository

Destination Path:

Name:

Git

☐ Create Repository On Account



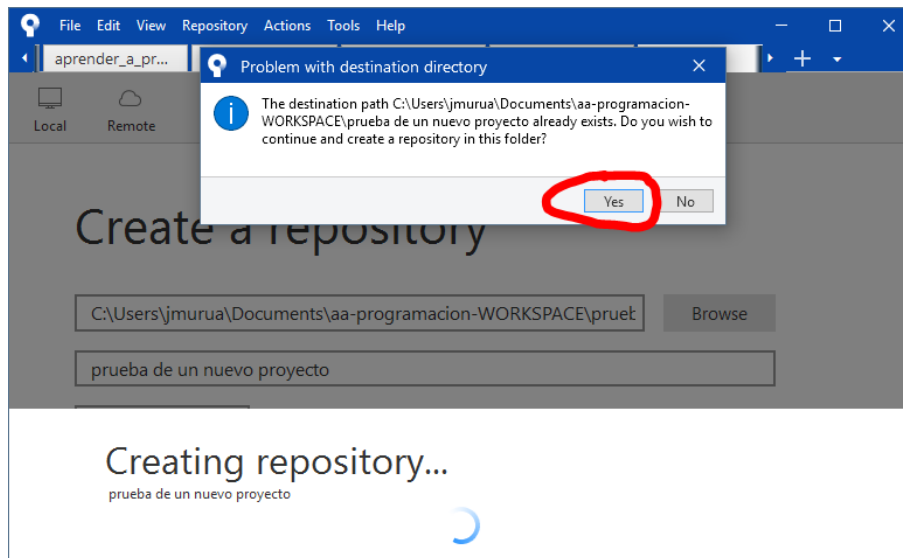
Create a repository

C:\Users\jmurua\Documents\aa-programacion-WORKSPACE\prueb

prueba de un nuevo proyecto

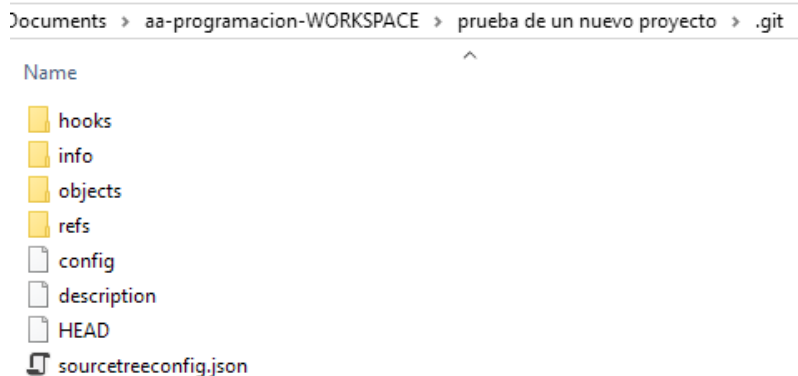
Git

☐ Create Repository On Account

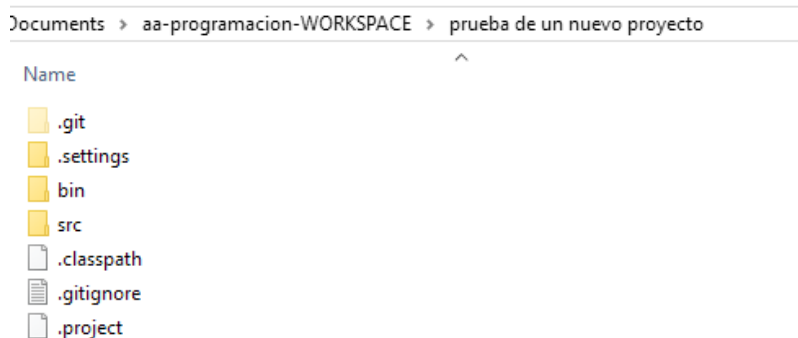


nota: También se puede crear un repositorio desde la línea de comandos, usando la orden ‘git init’ <https://git-scm.com/docs/git-init>

Al crear un repositorio, se genera la subcarpeta oculta ‘.git’ bajo la carpeta de proyecto indicada. En esa subcarpeta es donde Git almacenará el contenido y el registro de los cambios que se vayan incorporando en cada “commit”.



importante: En la carpeta de proyecto es necesario añadir manualmente un archivo de texto con el nombre ‘.gitignore’. Este archivo se encarga de que no todo el contenido del proyecto se guarde en el sistema de gestión de versiones. Por ejemplo, los archivos intermedios de compilación, los binarios finales, ... no es necesario guardarlos. Ya que son (re)generados automáticamente cada vez que se compila el proyecto..

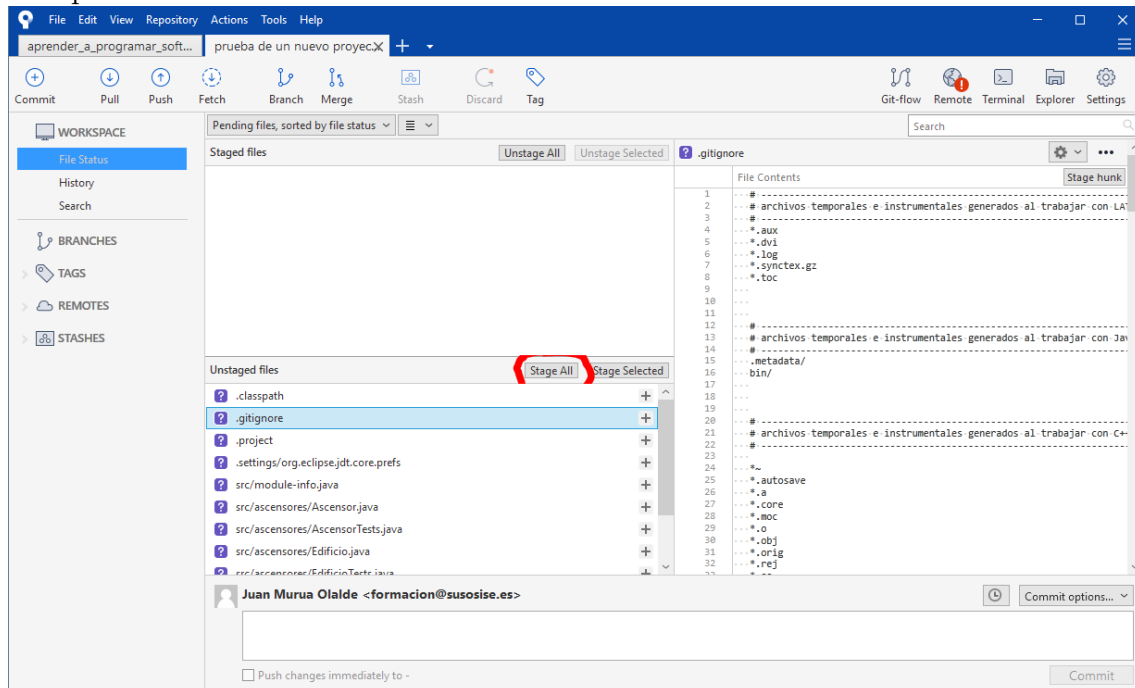


Por ejemplo, un trozo del contenido de .gitignore podría ser:

```
# archivos temporales e instrumentales generados en Java - Eclipse
.metadata/
bin/

# archivos temporales e instrumentales generados en C# - Visual Studio
[Bb]in/
[Dd]ebug/
[Rr]elease/
[Oo]bj/
*.suo
*.user
```

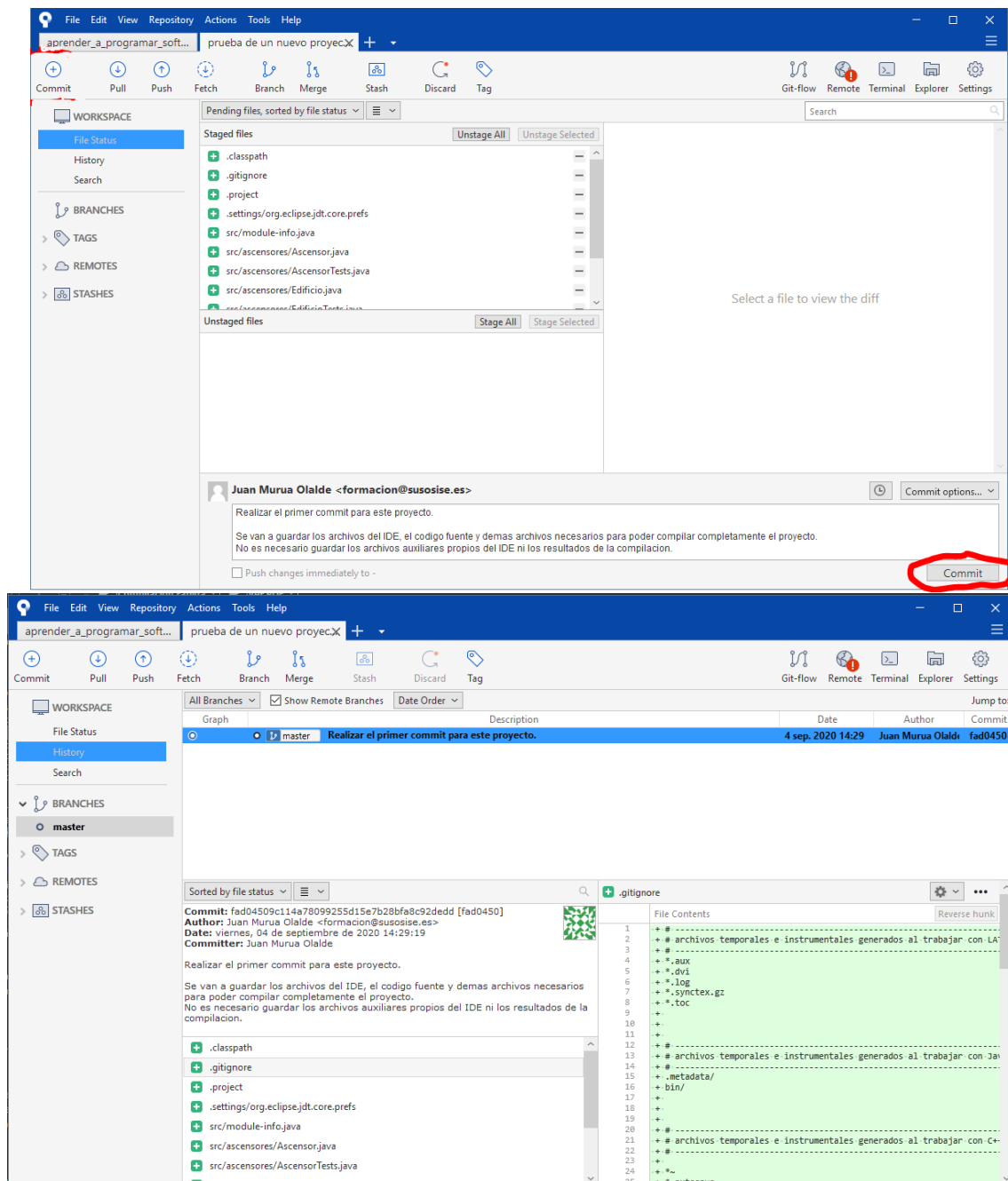
Sourcetree (dándole unos segundos para refrescarse tras acceder a su pantalla) nos mostrará en todo momento los archivos que hayan sido modificados desde la última vez que se hizo un “commit”.



Nota: Prestar atención al hecho de que la carpeta ‘bin’, que está en la carpeta de código, está siendo ignorada por Sourcetree(Git).

Para guardar esos cambios, son necesarios dos pasos:

1. Poner en el ‘stage’ los archivos de los que se desean guardar cambios clicando el botón que corresponda.
2. Escribir una explicación en el ‘commit’ y clicar en el botón correspondiente.



La forma básica de trabajar a partir de ahí es la de:

- Cada vez que se retoma el trabajo en el proyecto, acordarse siempre de comenzar echando un vistazo al estado del proyecto en Sourcetree. Con el tiempo se volverá un hábito saludable.
- Si se está compartiendo el trabajo con otras personas (es decir, hay repositorios remotos ligados a este repositorio local). Se ha de hacer siempre un **'pull'** antes de comenzar a trabajar, para traer los cambios que esas otras personas hayan podido realizar. Y comenzar así nuestro trabajo sobre la última versión disponible del proyecto.
- **Trabajar...**

- Cuando se llega a algún punto significativo. . . , añadir archivos al stage y hacer un **‘commit’** para guardar los cambios hasta ese punto.
- **Seguir trabajando. . .**
- Cuando se llega a algún punto significativo. . . , añadir archivos al stage y hacer un **‘commit’** para guardar los cambios hasta ese punto.
- . . .
- Si se está compartiendo el trabajo con otras personas, acordarse de hacer un **‘push’** para llevar esos commit al repositorio remoto compartido. (Nota: si el trabajo se ha realizado en una rama (**‘branch’**) distinta de master, es necesario hacer un **‘merge’** previo al push entre la rama de trabajo y la rama master.) (Nota: algunos prefieren hacer **‘rebase’s** en lugar de **‘merge’s**, pero la diferencia entre ambos procesos es un concepto avanzado más allá del alcance de esta breve introducción.)

Un consejo útil: hacer commit con frecuencia, como mínimo uno cada día de trabajo o así. Esto lleva a centrarse en metas concretas que se puedan completar del todo (**“done-done”**) en un día o así. Potenciando así el saludable hábito de la *programación incremental*: mantener siempre el código en un estado compilable/utilizable y nunca con un montón de frentes abiertos. . .

Otro consejo útil: compartir (hacer push) con frecuencia. Tener en cuenta que los compañeros no van a ver nuestro trabajo hasta que lo hagamos. Están basando su trabajo sobre el último estado conocido del código. Cuanto más tardemos en compartir, más probabilidades de que se produzcan colisiones entre nuestro trabajo y el suyo por haber escrito dos personas sobre la misma parte del mismo archivo. (<https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts>)

Un consejo importante: bajo ningún concepto, NUNCA, . . . intentar corregir un commit ya realizado. Siguiendo el saludable hábito de la *transparencia*, la forma correcta de subsanar un error en un commit es hacer otro commit guardando nuevos cambios correctores y explicando en su comentario la existencia de un error en el anterior commit y que es corregido por este nuevo commit.

Git tiene comandos que permiten alterar, revertir o borrar commits ya realizados. . . ¡pero son un camino seguro al desastre para cualquier persona novata!

1.2. Test unitarios

Los test **son una parte más del código**. A medida que escribimos código “para hacer” algo, escribimos también el correspondiente código “para comprobar” que ese algo se hace como se pretendía hacerlo.

truco: Es más sencillo si trabajamos de forma incremental, en pequeños pasos; cada paso una funcionalidad. Es decir, no esperar a tener una enorme porción de código “para hacer” antes de ponernos a escribir sus correspondientes partes de código ‘para comprobar’.

El orden en que se escriba cada parte no es relevante. Es más, los seguidores del paradigma ‘Test-driven development’ (TDD) recomiendan escribir primero el código “para comprobar”, ejecutarlo, comprobar que el test falla y escribir luego el código “para hacer” que permita al test ejecutarse con éxito.

Los test **son una red de seguridad** imprescindible para refactorizar y evolucionar(modificar) cualquier programa sin correr demasiados riesgos de romper la funcionalidad existente.

(nota: Todos tendemos a evitar tocar aquello que percibimos como peligroso. Un código sin test tiende, con el tiempo, a convertirse en una serie de “pastiches” ligados entre sí sin demasiada coherencia. El miedo lleva a reducir los cambios a los mínimos imprescindibles. Evitando refactorizar lo que ya funciona. E impidiendo así que las nuevas aportaciones se integren con coherencia en el conjunto ya existente.)

Los test **constituyen una excelente documentación técnica** para el programa. Recogen detalladamente lo que este ha de hacer, todos los requisitos y la intención de diseño. Muchas veces es más sencillo enterarse qué hace un determinado código leyendo sus test que leyendo el propio código.

(nota: Todos los programadores profesionales sabemos que la otra alternativa: la de documentar el código reflejándolo en algo externo totalmente desligado de él, nunca se mantiene y acaba siempre desactualizada (es decir, inútil). Sin embargo, los test sí que se mantienen regularmente; de hecho, no hay más que ejecutarlos para ver cuales fallan.)

Un ejemplo:

```
public class AnalizadorDeTextos
{
    private String texto;
    private java.util.ArrayList<String> palabras;

    public AnalizadorDeTextos(String texto)
    {
        this.texto = texto;

        this.palabras = new java.util.ArrayList<String>();
        separarLasPalabras();
    }

    private void separarLasPalabras()
    {
        String expresionRegularDeFiltro = "\\s+";
        for (String palabra : texto.split(expresionRegularDeFiltro))
        {
            this.palabras.add(palabra);
        }
    }

    public Integer getNumeroDePalabrasEnElTexto()
    {
        return this.palabras.size();
    }
}
```

```
    }

    public java.util.ArrayList<String> getListaDePalabrasEnElTexto()
    {
        return this.palabras;
    }
}

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

class AnalizadorDeTextosTest
{
    @Test
    void testGetNumeroDePalabrasEnElTexto()
    {
        AnalizadorDeTextos analizadorSinPuntuacion
            = new AnalizadorDeTextos("Esto
                                     es una prueba");

        assertEquals((Integer)4,
                     analizadorSinPuntuacion.
                     getNumeroDePalabrasEnElTexto());

        AnalizadorDeTextos analizadorConPuntuacion
            = new AnalizadorDeTextos("Esto es una
                                     prueba, pardiez.");
        assertEquals((Integer)5,
                     analizadorConPuntuacion.
                     getNumeroDePalabrasEnElTexto());

        AnalizadorDeTextos analizador1 = new AnalizadorDeTextos("prueba
                                                                    ");
        assertEquals((Integer)1, analizador1.
                     getNumeroDePalabrasEnElTexto());

        AnalizadorDeTextos analizador0 = new AnalizadorDeTextos("");
        assertEquals((Integer)0, analizador0.
                     getNumeroDePalabrasEnElTexto());
    }

    @Test
    void testGetListaDePalabrasEnElTexto()
    {
        AnalizadorDeTextos analizadorSinPuntuacion
            = new AnalizadorDeTextos("Esto
                                     es una prueba");

        ArrayList<String> resultadoSinPuntuacion = new ArrayList<String>
            ();
        resultadoSinPuntuacion.add("Esto");
        resultadoSinPuntuacion.add("es");
    }
}
```

```

        resultadoSinPuntuacion.add("una");
        resultadoSinPuntuacion.add("prueba");
        assertEquals(resultadoSinPuntuacion,
            analizadorSinPuntuacion.
                getListaDePalabrasEnElTexto());

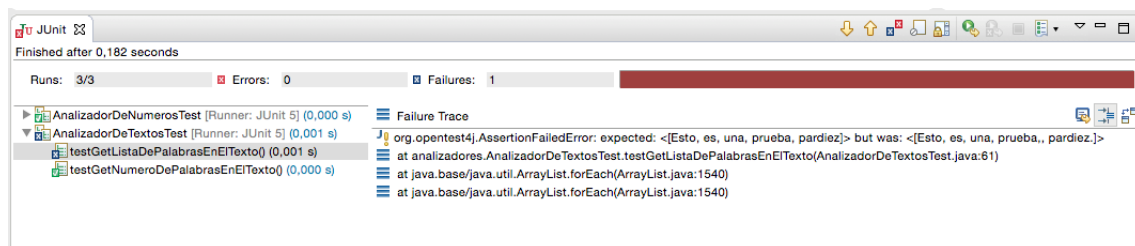
        AnalizadorDeTextos analizadorConPuntuacion
            = new AnalizadorDeTextos("Esto es una
                prueba, pardiez.");
        ArrayList<String> resultadoConPuntuacion = new ArrayList<String>
            >();
        resultadoConPuntuacion.add("Esto");
        resultadoConPuntuacion.add("es");
        resultadoConPuntuacion.add("una");
        resultadoConPuntuacion.add("prueba");
        resultadoConPuntuacion.add("pardiez");
        assertEquals(resultadoConPuntuacion,
            analizadorConPuntuacion.
                getListaDePalabrasEnElTexto());

        AnalizadorDeTextos analizador1 = new AnalizadorDeTextos("prueba
            ");
        ArrayList<String> resultado1 = new ArrayList<String>();
        resultado1.add("prueba");
        assertEquals(resultado1, analizador1.
            getListaDePalabrasEnElTexto());

        AnalizadorDeTextos analizador0 = new AnalizadorDeTextos("");
        ArrayList<String> resultado0 = analizador0.
            getListaDePalabrasEnElTexto();
        assertEquals(0, resultado0.size());
    }

}

```



Otro ejemplo:

```

public class Ascensor
{
    private int plantaMasBaja;
    private int plantaMasAlta;

    private int plantaDondeEstaAhora;

    public Ascensor(Integer cantidadDePlantasPorEncimaDeCero,

```



```
        Integer cantidadDePlantasPorDebajoDeCero)
    {
        this.plantaDondeEstaAhora = 0;
        if (cantidadDePlantasPorEncimaDeCero > 0
            && cantidadDePlantasPorDebajoDeCero >= 0)
        {
            this.plantaMasBaja = cantidadDePlantasPorDebajoDeCero * -1;
            this.plantaMasAlta = cantidadDePlantasPorEncimaDeCero;
        }
        else
        {
            throw new java.lang.IllegalArgumentException();
        }
    }

    public void IrALaPlanta(Integer planta)
    {
        if (planta >= this.plantaMasBaja && planta <= this.
            plantaMasAlta)
        {
            this.plantaDondeEstaAhora = planta;
        }
    }

    public Integer getPlantaDondeEstaAhora()
    {
        return this.plantaDondeEstaAhora;
    }

    public String toString()
    {
        return "Este ascensor puede ir desde la planta "
            + this.plantaMasBaja + " hasta la planta " + this.
                plantaMasAlta
            + System.lineSeparator()
            + "Ahora esta en la planta " + this.
                getPlantaDondeEstaAhora();
    }
}

class AscensorTest
{
    @Test
    void Comprobar_situaciones_excepcionales()
    {
        assertThrows(java.lang.IllegalArgumentException.class,
            () -> {Ascensor prueba = new Ascensor(-2, 2);} );
        assertThrows(java.lang.IllegalArgumentException.class,
            () -> {Ascensor prueba = new Ascensor(2, -2);} );
    }
}
```

```

@Test
void Comprobar_operaciones_normales()
{
    Ascensor ascensorDePruebas = new Ascensor(2,2);
    assertEquals((Integer) 0, ascensorDePruebas.
        getPlantaDondeEstaAhora());

    ascensorDePruebas.IrALaPlanta(1);
    assertEquals((Integer) 1, ascensorDePruebas.
        getPlantaDondeEstaAhora());

    ascensorDePruebas.IrALaPlanta(-1);
    assertEquals((Integer) (-1), ascensorDePruebas.
        getPlantaDondeEstaAhora());

}

@Test
void Comprobar_operaciones_anomalias()
{
    Ascensor ascensorDePruebas = new Ascensor(2,2);
    ascensorDePruebas.IrALaPlanta(333);
    assertEquals((Integer) 0, ascensorDePruebas.
        getPlantaDondeEstaAhora());

}

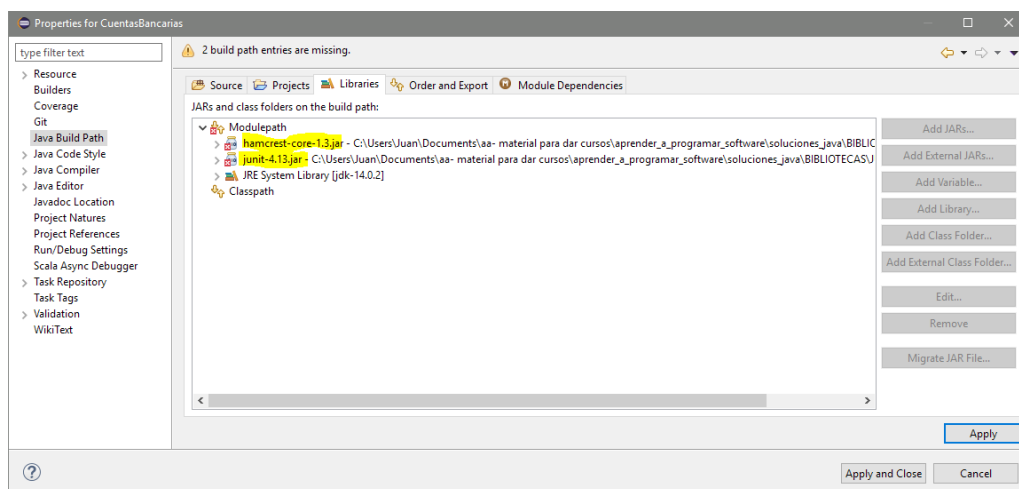
}

```

Breve introducción a JUnit4

Aquí se dan cuatro pinceladas para comenzar a trabajar con JUnit4 dentro de Eclipse. Para más detalles, acudir a la web oficial (<https://junit.org/junit4/>) o a otras fuentes de información.

En las propiedades del proyecto (menú 'Project' 'Properties'), en el apartado 'Java Build Path', en la pestaña 'Libraries', incorporar los dos .jar de biblioteca: hamcrest-1.3.jar y junit-4.13.jar



Por cada clase de código, incorporar al proyecto otra clase de test. En las clases de test hay que importar los objetos JUnit que se vayan a utilizar:

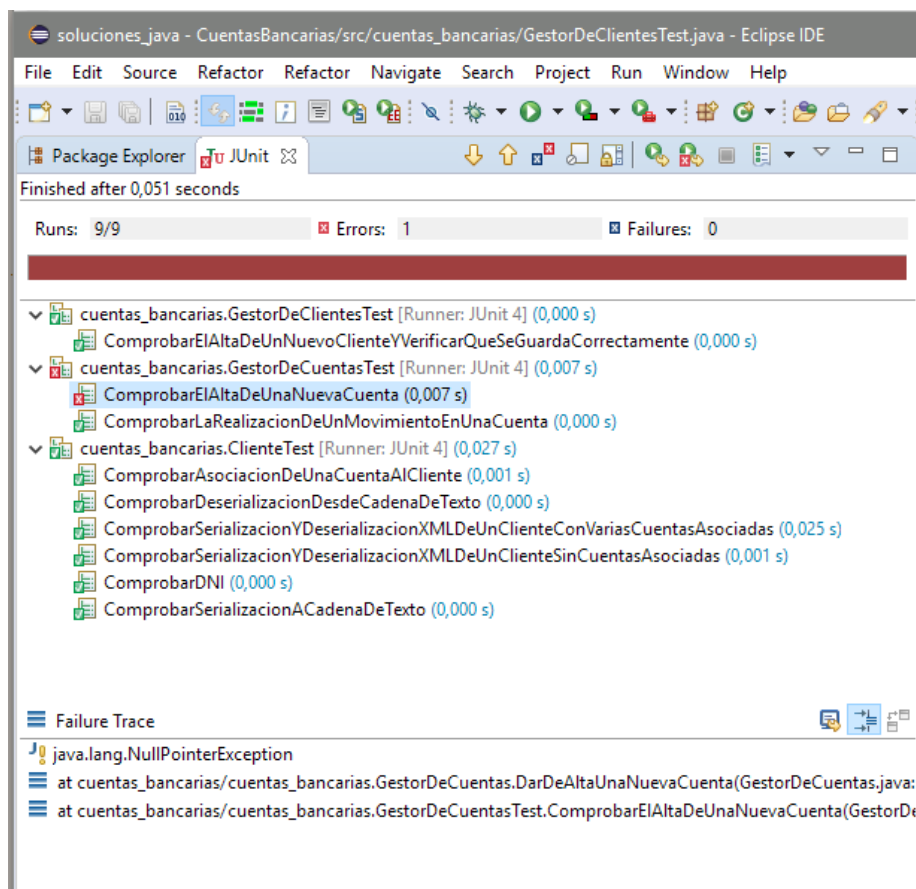
```
import org.junit.Test;
import static org.junit.Assert.*;
```

Dentro de una clase de test, sus métodos(funciones) de test han de ser públicos y se han de identificar con la correspondiente propiedad @Test:

```
@Test
public void ComprobarAsociacionDeUnaCuentaAlCliente()
{
    assertEquals(0, clienteDePrueba.getIdsDeLasCuentasAsociadas().size());
    clienteDePrueba.asociarUnaCuentaAlCliente('123456789');
    java.util.ArrayList<String> cuentas = clienteDePrueba.getIdsDeLasCuentasAsociadas();
    assertEquals(1, cuentas.size());
    assertEquals('123456789', cuentas.get(0));
}
```

Procurar que el nombre de cada método(función) refleje lo mejor posible la comprobación que realiza. No preocuparse porque los nombres resulten largos.

JUnit tiene soporte directo por parte de Eclipse, así es que los test se ejecutan directamente con clic-dcho ‘Run As’ ‘JUnit Test’.



Breve introducción a QTest

Aquí se dan cuatro pinceladas para comenzar a trabajar con QTest dentro de Qt Creator. Para más detalles, acudir a la web oficial (<https://doc.qt.io/qt-5/qtest-overview.html>) o a otras fuentes de información.

Breve introducción a los proyectos test de Visual Studio

Aquí se dan cuatro pinceladas para comenzar a trabajar con test unitarios dentro de Visual Studio. Para más detalles, acudir a la web oficial (<https://docs.microsoft.com/es-es/visualstudio/test/getting-started-with-unit-testing-view=vs-2019>) o a otras fuentes de información.

Breve introducción a unittest

Aquí se dan cuatro pinceladas para comenzar a trabajar con unittest en Python. Para más detalles, acudir a la web oficial (<https://docs.python.org/3/library/unittest.html>) o a otras fuentes de información.

1.3. Refactorizar

Refactorizar: reescribir código sin afectar en nada a lo que este hace.

El código sigue haciendo exactamente lo mismo. Pero se mejora su coherencia interna, se hace más claro de leer, se mejora su mantenibilidad, se evolucionan las técnicas utilizadas, se hace más fácil de modificar, se mejora...

Es importante conocer las ayudas que el editor puede ofrecer a este respecto:

- Renombrar variables, funciones, clases, módulos,... ; cambiando automáticamente su nombre allá donde han sido utilizadas.
- Mover variables, métodos, clases,... de una clase, módulo, archivo, carpeta,... a otra ; actualizando automáticamente las referencias allá donde han sido utilizadas (o marcando dichos puntos, si necesitan ajuste manual).
- Cambiar parámetros en la signatura de una función o método ; actualizando automáticamente los puntos donde ha sido utilizada (o marcando dichos puntos, si necesitan ajuste manual).
- Extraer un tozo de código a su propia función; definiendo automáticamente los parámetros de entrada y de salida necesarios.
- etc.

Y nunca está de más destacar la red de seguridad aportada por los test unitarios. Tras cada refactorización, los test nos aseguran que el código sigue haciendo lo

mismo que hacia antes, (o nos avisan de los puntos donde hemos alterado alguna funcionalidad sin querer).¹

¹Ver más información sobre estas y otras herramientas de programación en el capítulo 1 “Herramientas” del documento https://www.susosise.es/documentos/Proyecto_Software.pdf

Capítulo 2

Interfaz gráfico de usuario: GUI (Graphic User Interface)

2.1. Formas de construirlos

Hay dos formas de definir las pantallas:

2.1.1. Indicando ubicaciones y dimensiones fijas

Dibujando tal cual el interfaz a presentar. Es decir, dando prescripciones exactas de donde ha de ir cada elemento.

Tiene la ventaja de ser sencillo y evidente, ya que es totalmente estático.

Pero tiene la pega de que:

- Puede resultar imposible acceder a ciertos controles, si se trabaja en una pantalla de menos resolución que la inicialmente prevista.
- Puede visualizarse con tamaño diminuto, si se trabaja en una pantalla de mucha más resolución y densidad de la inicialmente prevista.

2.1.2. Indicando relaciones y recomendando dimensiones

Dando intenciones de diseño y dejando que sea el propio sistema quien dibuje el interfaz, según pueda hacerlo en cada ocasión. Para ello:

- Se elige una distribución general de la pantalla (layout). Esta distribución dictará las diferentes zonas y las formas generales de acomodar elementos.
- Se ponen componentes dentro de cada zona, en un determinado orden de colocación e indicando los márgenes a respetar para dimensiones y espaciados (mínimas, preferidas, máximas).

Tiene la ventaja de adaptarse automáticamente a diversos tamaños y formatos de pantalla. Algo muy útil hoy en día, ya que la aplicación será utilizada desde diversos

dispositivos: sobremesas, portátiles, tabletas, móviles,... con pantallas de diferentes resoluciones y densidades.

Pero tiene la pega de ser más complejo de definir y de requerir ensayos prueba/error para comprobar su correcto comportamiento en diversos escenarios de uso. (Además, al ser dinámico, puede no resultar siempre exactamente como queremos que quede.)

2.2. Event Loop , Event Dispatcher ::: Reaccionar a lo que sucede en la pantalla, en el ratón o en el teclado

(nota: Para una descripción general de cómo funciona este sistema, ver más adelante la sección 14.1 Orientación a eventos (event-driven).)

2.3. Responsive GUI ::: Reaccionar a lo que sucede en el interior de la aplicación

‘WorkerThread’ y otros sistemas para ejecutar el código en diversas hebras de ejecución coordinadas entre sí. (Sin perder de vista que, normalmente, la hebra principal suele ser aquella donde corre el interfaz gráfico –donde ha arrancado la aplicación y desde la que se inician las demás hebras–.)

2.4. Elementos habituales en los GUIs

Un poco de historia y curiosidades:

- Los primeros prototipos: Engelbart y “la madre de todas las demos”. <https://www.youtube.com/playlist?list=PLCGFadV4FqU2yAqCzKaxnKKXgnJBURKTE>
- El primer equipo viable en los EEUU: Xerox Alto.
https://es.wikipedia.org/wiki/Xerox_Alto
- El primer equipo viable en Europa: Oberon
[https://en.wikipedia.org/wiki/Oberon_\(operating_system\)](https://en.wikipedia.org/wiki/Oberon_(operating_system))
<http://www.projectoberon.com/>
- El primer sistema comercial con éxito: Apple Macintosh
<https://es.wikipedia.org/wiki/Macintosh>

2.4.1. Controles con los que el usuario puede interactuar

Los más habituales suelen ser:

- tipo “Label”: algo donde mostrar texto.
- tipo “TextBox”: algo donde introducir texto.
- tipo “Button” o “PushButton”: algo donde hacer clic para desencadenar una acción.
- tipo “RadioButton”: algo para elegir una, y solo una, de entre varias opciones.
- tipo “CheckBox”: algo para activar/desactivar una opción.
- tipo “Slider”: algo para indicar cantidades de forma continua, deslizándose entre un mínimo y un máximo.
- tipo “ListView” o “ComboBox”: algo para elegir uno, o varios, de entre una lista (corta) de elementos predefinidos.
- tipo “TreeView”: algo para elegir uno, o varios, de entre una colección (larga) de elementos predefinidos organizados de forma arbórea.
- tipo “ProgressBar”: algo con lo que mostrar el progreso de una tarea larga.
- tipo “PictureBox”: algo donde mostrar imágenes.
- tipo “Player”: algo donde mostrar vídeos.
- tipo “Table”: algo donde interactuar con datos tabulares.
- tipo “Scroll”: algo para permitir desplazar el contenido y así poder visualizarlo cuando no cabe entero en el espacio previsto para él.

2.4.2. Distribuciones generales con las que organizar la pantalla

Las más habituales suelen ser:

- tipo “panel”: se usan para agrupar varios componentes y poder así colocarlos como si fueran un solo componente.
- tipo “flow”: los componentes se colocan de forma secuencial uno detrás de otro, en sentido horizontal o en sentido vertical.
- tipo “grid”: los componentes se colocan en las casillas de una rejilla con un cierto número de columnas y de filas.
- tipo “boxes”: el espacio se distribuye en zonas fijas donde se colocan componentes; normalmente esas zonas suelen ser cinco: arriba (norte), a la izquierda (oeste), en el centro, a la derecha (este), abajo (sur).
- tipo “cards”: se dispone de varias páginas, permitiendo saltar fácilmente de una a otra (usualmente mediante unas pestañas o algo similar).

Estas distribuciones (‘layouts’) se pueden combinar entre sí. Por ejemplo: podemos poner un “flow” dentro de una celda concreta de un “grid”, podemos poner un determinado “grid” dentro de la zona centro de un “boxes”, podemos poner diferentes “grid”s en las diferentes páginas de un “cards”, etc.

Algunos enlaces prácticos (Una imagen vale más que mil palabras...)

A Visual Guide to Java Swing Layout Managers: <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

JavaFX Layout Panes: https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

QT Layout Management: <https://doc.qt.io/qt-5/layout.html>

.NET WPF Layouts, a Visual Quick Start: <https://www.codeproject.com/Articles/30904/WPF-Layouts-A-Visual-Quick-Start>

WPF Layout System: <https://docs.microsoft.com/es-es/dotnet/desktop/wpf/advanced/layout?view=netframeworkdesktop-4.8>

Python Tkinter Grid Geometry Manager: <https://tkdocs.com/tutorial/grid.html>

Capítulo 3

Ejercicios

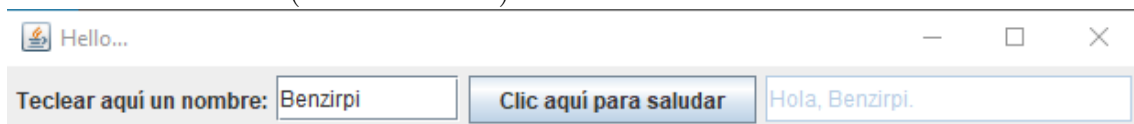
((una disculpa del autor: este capítulo esta “en construcción...” y sigo trabajando en él; queda bastante trabajo para completarlo.))

(nota: Ver posibles soluciones al final del libro, en el capítulo 17.)

Los ejercicios de este capítulo son específicos para adquirir soltura con la biblioteca GUI que se vaya a utilizar. Para adquirir experiencia, conviene seguir practicando... dotando de interfaz gráfico a todos los demás trabajos que se hagan en los siguientes capítulos.

3.1. Hello, Benzirpi.

Algo sencillo para empezar: un textbox donde teclear un nombre y un botón que rellena otro textbox (de solo lectura) con un saludo a ese nombre.



3.2. Bocatas

Algo sencillo para elegir opciones: unos cuantos radiobuttons y checkboxes.

The screenshot shows a Java Swing window titled "Bocatas...". It contains three panels for selecting ingredients:

- Tipo de pan:** Radio buttons for "normal", "integral", "bagete" (selected), and "de molde".
- Ingrediente principal:** Radio buttons for "lomo" (selected), "tortilla de jamon York", "tortilla de atun", "tortilla de queso", and "calamares".
- Ingredientes adicionales:** Checkboxes for "cebolla fresca", "cebolla caramelizada", "pimientos" (checked), and "queso" (checked).

At the bottom, there is a button labeled "Clic aquí para componer el bocata" and a text field labeled "Resultado:" containing the text "Bocata con pan bagete, de lomo, pimientos, queso."

3.3. Selección plana

Algo sencillo para elegir un elemento de una lista desplegable o de una lista abierta.

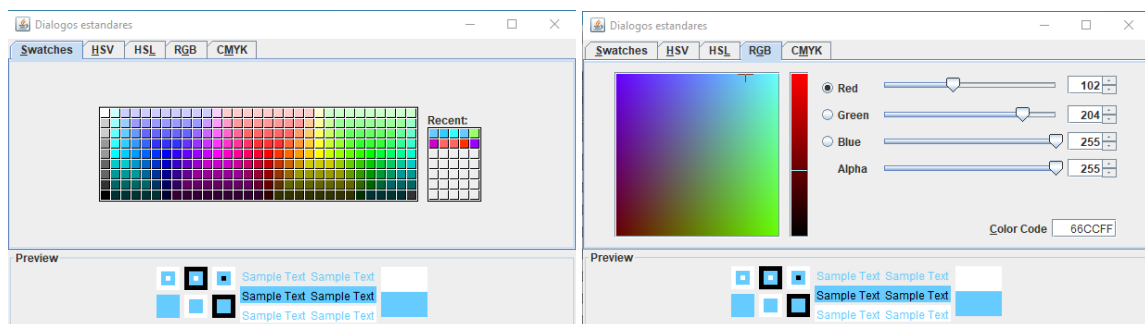
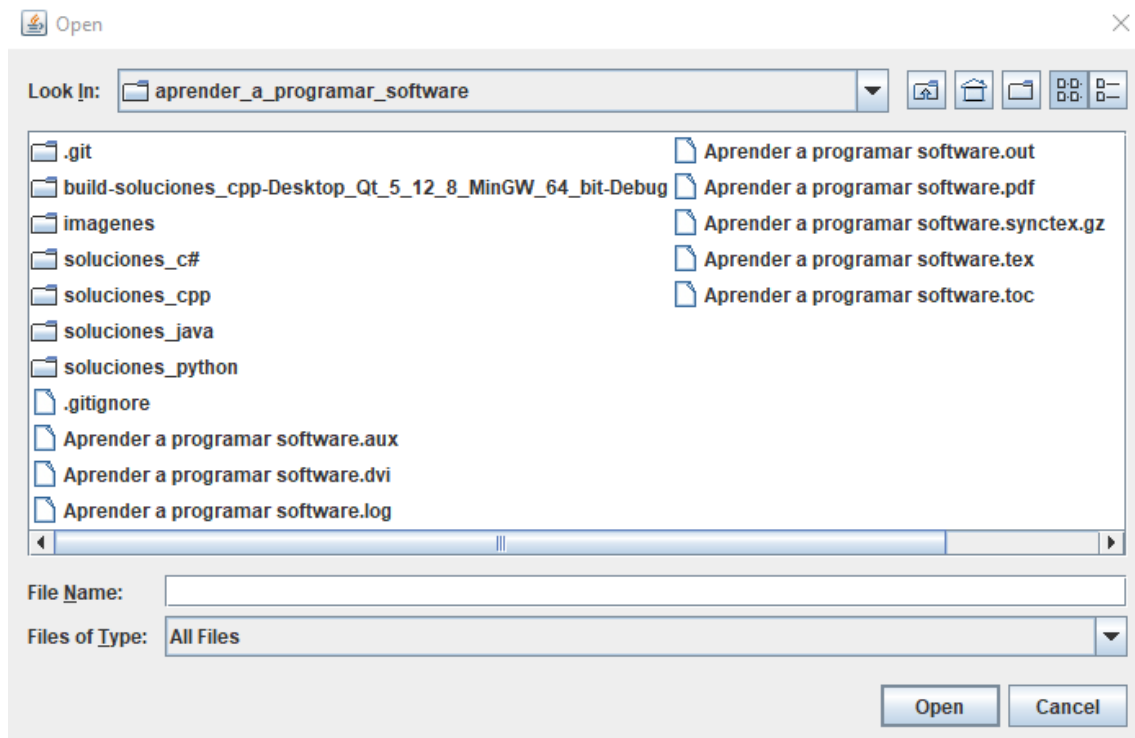
The screenshot shows a Java Swing window titled "Seleccionar en una lista". It features a dropdown menu labeled "Seleccionar premio:" with the selected item "un boligrafo". Below it is a list box labeled "Seleccionar concursante:" containing the following entries:

- Pedro XXXXXXXXX ZZZZZZ
- Ana XXXXXXXXX ZZZZZZ (highlighted)
- Maria XXXXXXXXX ZZZZZZ
- Alberto XXXXXXXXX ZZZZZZ
- Jose XXXXXXXXX ZZZZZZ
- Raul XXXXXXXXX ZZZZZZ

There is a button labeled "Clic aquí para procesar la seleccion" and a text field labeled "Resultado:" containing the text "Ana XXXXXXXXX ZZZZZZ ha ganado un boligrafo !".

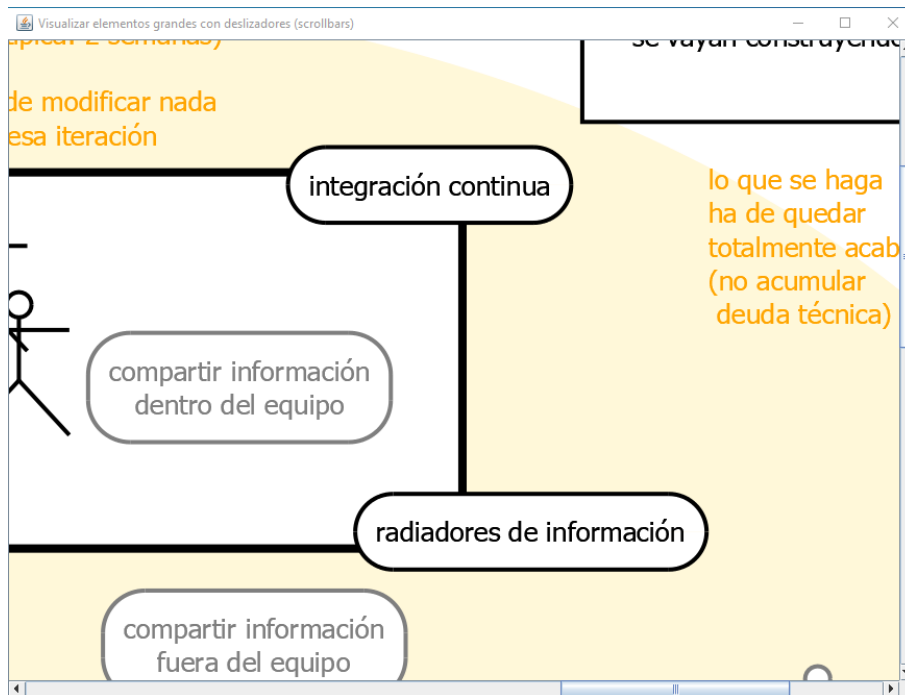
3.4. Diálogos estándares

Manejo de los diálogos del sistema operativo para: seleccionar carpetas, seleccionar archivos, seleccionar colores,...



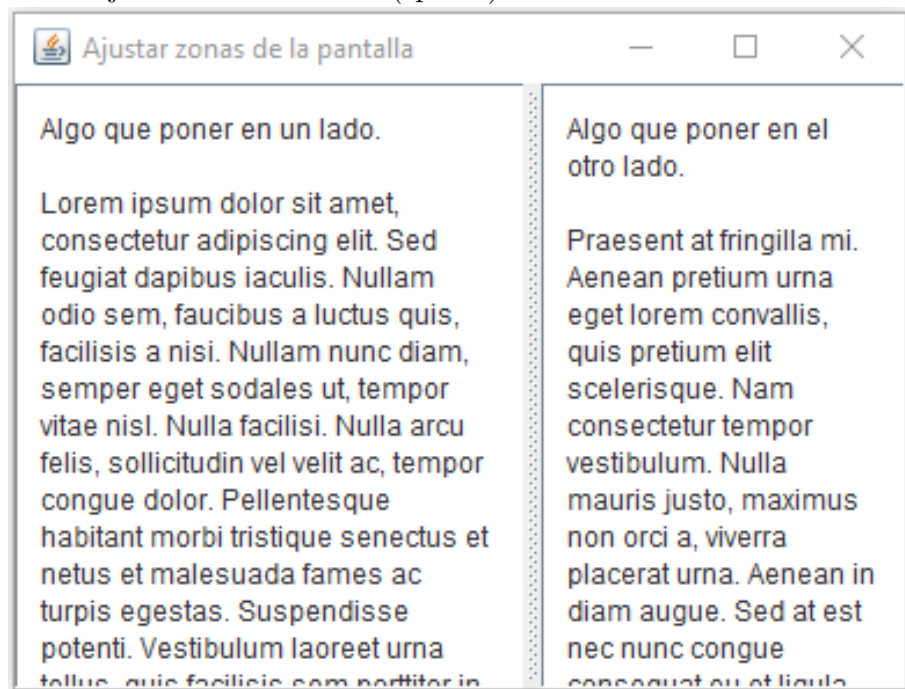
3.5. Scroll

Cuando se ha de mostrar un contenido que no cabe en el espacio disponible, se puede recurrir a unos deslizadores (scrollbars) horizontales o verticales para permitir desplazar dicho contenido dentro del espacio disponible.



3.6. Split

Cuando se desea permitir al usuario modificar sobre la marcha el espacio dedicado dentro de la ventana a los componentes dentro de ella, se puede recurrir a un separador ajustable entre zonas (splitter).

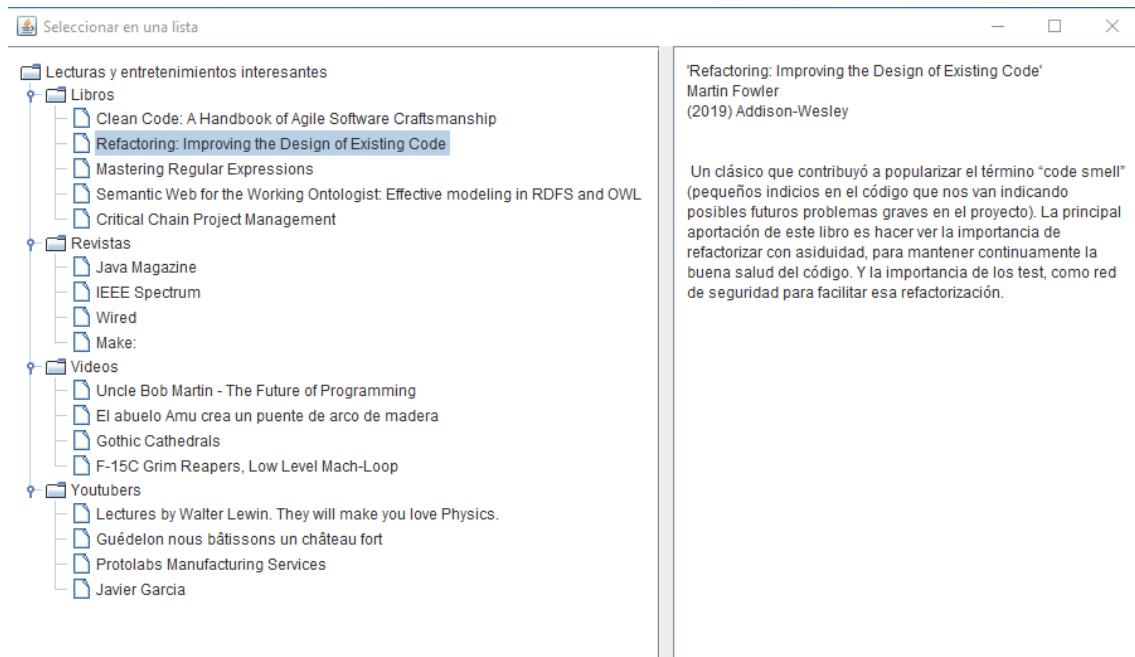


3.7. Calendario

Pedir una fecha mostrando un calendario.

3.8. Selección arbórea

Presentar elementos con estructura jerárquica en árbol, navegar a través de ella mostrando u ocultando ramas, seleccionar algún elemento, y hacer algo con él (por ejemplo, mostrar información detallada de ese elemento concreto).

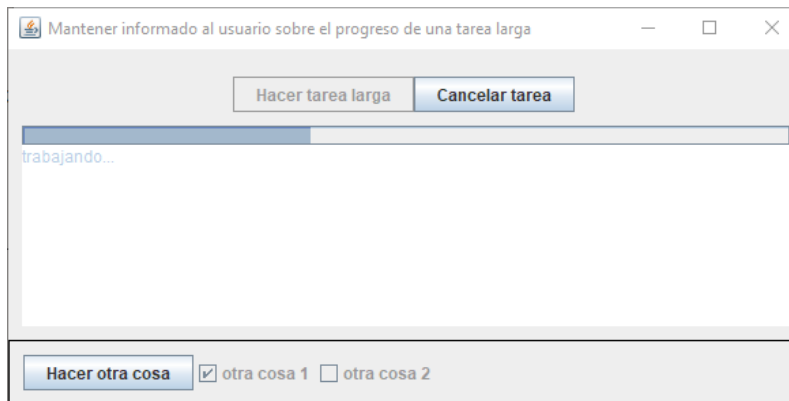


3.9. Barra de progreso

Cuando se desencadena una tarea larga, es conveniente mantener al usuario informado de cómo va progresando dicha tarea. Así se evita dar la impresión que el sistema se ha colgado.

nota: No suele ser sencillo. Se ha de recurrir a la multitarea para conseguir que el interfaz de usuario (main thread) se vaya refrescando periódicamente mientras se está ejecutando la tarea (worker thread). Pero es algo tan necesario que casi todos los entornos de programación cuentan con algún mecanismo específico para implementar esa comunicación periódica.

Una sugerencia de ejercicio para practicar:



Una lectura interesante: <https://www.oracle.com/technical-resources/articles/javase/swingworker.html>

3.10. Internacionalización (i18n) y localización (L10n)

- Internacionalización (i18n) son los mecanismos para adaptar el software a distintos idiomas y países.
- Localización (L10n) son los mecanismos para adaptar el software a los distintos usos y costumbres de cada país: moneda, pesos y medidas, formatos numéricos, unidades de medida, formatos de fecha/hora, convenciones culturales, sentido de escritura,...

Ambos dos son aspectos muy importantes para cualquier software que no sea algo pequeño “de andar por casa”. Y, aún en estos, hay que tener en cuenta que muchas veces crecen y acaban desplegándose por multitud de sitios a donde no se había previsto que llegaran.

Tanto la internacionalización como la localización son prácticamente imposibles de implementar “a posteriori”. Es decir, se han de ir incorporando desde el primer momento; desde la primera línea de código que se escriba.

Si conocemos y nos sentimos cómodos usando los mecanismos que tenga nuestra plataforma de desarrollo para facilitar esos aspectos. No nos costará mucho más utilizarlos en todos nuestros trabajos; aunque en ese momento estemos pensando en un solo idioma y un solo país. Así, si mas tarde se han de incorporar más idiomas o más países, el camino estará preparado.

3.11. Ayudas y documentación

Es importante conocer los mecanismos que tenga nuestra plataforma de desarrollo para mostrar documentación de forma dinámica: indicaciones al poner el ratón un tiempo quieto sobre un determinado elemento(tooltips), acceso al manual de uso (ayuda, tecla F1), advertencias directas para algunos aspectos especialmente delicados (pop-ups), etc.

Son aspectos importantes si se van a desarrollar programas de una cierta envergadura. Programas que vayan a ser usados por muchas personas distintas, a lo largo de mucho tiempo.

Algunas recomendaciones

Creo que es mejor no incordiar mucho a los usuarios en su día a día. Suele ser suficiente con disponer de:

- Unos tooltips cortos y precisos acerca de lo que hace cada parte del interface. Tooltips que solo aparezcan si el usuario deja manifiestamente el ratón quieto durante unos 2 segundos sobre algo.
- Y un menú o botón de ‘Ayuda’ bien visible, que abra el manual de usuario.

Por experiencia sé que casi nadie se lee el manual de usuario. Ni aún en el caso de que esté bien redactado, sea claro y sea ameno.

Pero aún así, creo que merece la pena dotar a nuestros programas de una buena documentación. Algo que realmente permita a una persona curiosa aprender la filosofía de funcionamiento del programa y los conceptos principales involucrados en su manejo.

Escribir una mera recopilación de cómo hacer qué, menú a menú y botón a botón, solo por cumplir el expediente, no merece la pena. Además de ser inútiles, han sido ese tipo de manuales los que han dado tan mala fama a los manuales de usuario.

De ponerse a producirlo, suele ser más rentable enfocar el manual de usuario como un curso de autoestudio.

Si además se le dota de un buen índice de temas, podrá ser utilizado también, en cierta medida, como referencia puntual para resolver dudas.

Capítulo 4

Objetos: clases e instancias

4.1. Encapsulación

nota: ver más adelante la sección 14.2 Orientación a objetos (object-oriented).

4.2. Polimorfismo

Posibilidad de definir varias funciones con el mismo nombre, pero con diferentes firmas (diferencias en alguno de los parámetros de entrada o datos de salida).

4.3. Herencia

Se utiliza cuando hay varias clases similares, clases que comparten todas ellas algunas propiedades y métodos comunes que son idénticos en todas ellas.

Es posible definir una clase madre con la parte común, y luego derivar varias clases hijas a partir de ella. Todas las clases hijas tienen automáticamente todo lo que tiene la clase madre, más lo que se añade de forma particular para cada una de ellas.

Cuando se vaya a utilizar solo su parte común, podemos declarar variables, parámetros o colecciones del tipo madre, para referirnos o contener instancias de cualquiera de las clases hijas.

4.4. Sobrecarga de funciones (overwrite)

Se utiliza junto con la herencia, cuando alguna clase hija necesita hacer algo peculiar en la implementación interna de alguno de los métodos comunes que recibe de la clase madre. Puede redefinirlo sobrescribiéndolo (overwrite) con una declaración propia de ese método.

4.5. Interface (clase abstracta)

Se utiliza cuando se necesitan unos ciertos métodos con forma exactamente igual, con una signatura concreta (parámetros de entrada y salida concretos) en unas cuantas clases distintas. Pero esos métodos realizan trabajos internos distintos según cada clase.

(nota: Se podría utilizar herencia, pero habría que sobrecribir los métodos en todas las clases hijas. La clase madre tan solo dictará la signatura de los métodos. Y eso es precisamente lo que hace un interface.)

Cuando una clase dice que implementa un interface, el compilador le obliga a que implemente todos y cada uno de los métodos de ese interface. Los puede implementar como quiera, pero ha de implementarlos; y ha de hacerlo respetando las signaturas (los parámetros de entrada y salida) definidas.

Cuando se vaya a utilizar solo su parte común, podemos declarar variables, parámetros o colecciones del tipo del interface, para referirnos o contener instancias de cualquiera de las clases que implementen dicho interface.

4.6. Cuándo usar herencia y cuándo interface

Ambas se utilizan cuando se ha de garantizar que un cierto número de clases/objetos tienen ciertos elementos comunes a todos ellos; Permitiendo así ser utilizados en ciertas ocasiones como si todos fueran el mismo tipo de objeto. Pero, ¿cuándo usar herencia y cuando interfaz?

Algunas reglas prácticas:

- Cuando la parte de código común a todas las clases/objetos es mucho mayor que la parte específica de cada cual \Rightarrow implementar la parte común en una clase madre de la cual se derivarán las clases hijas específicas.
- Cuando la parte de código común a todas las clases/objetos se preve que va a ser muy estable en toda la vida del software \Rightarrow implementar la parte común en una clase madre de la cual se derivarán las clases hijas específicas.
- Cuando la parte de común a todas las clases/objetos es mucho menor que la parte específica de cada cual \Rightarrow definir la parte común en un interface y obligar a las clases a implementar dicho interface.

En caso de duda \Rightarrow Interface suele ser la opción más clara y flexible cara a futuras modificaciones o ampliaciones.

Un aviso: Si la herencia a un nivel (madre \rightarrow hijas) suele ser complicada a veces. La herencia a más niveles (madre \rightarrow hijas \rightarrow nietas...), suele ser camino con muchas probabilidades de problemas en un futuro. Cuando se vea necesaria multiherencia en alguna aplicación, mejor repensar de nuevo la arquitectura para ver si realmente es o no imprescindible.

4.7. Factorias para “fabricar” objetos (inyección de dependencias)

Se utiliza cuando una clase A necesita utilizar objetos de otra clase X, pero no puede o no conviene que los instancie por sí misma. Se suele resolver definiendo una clase factoría capaz de instanciar objetos de diversas clases según los parámetros que se le pasen. La clase A utiliza esa factoría para obtener el objeto que necesita en cada momento.

Un caso de uso bastante típico es cuando el tipo concreto de objeto X a utilizar no se conoce hasta el momento en que el programa está funcionando (tiempo de ejecución); pudiendo ser de un tipo en una ocasión y de otro tipo en otra. Haciendo así imposible preverlo directamente en el código (tiempo de compilación).

Otro caso de uso bastante típico es cuando un objeto X es costoso de crear (por ejemplo, implica una conexión auditada y encriptada con una base de datos remota) y se necesita para los test unitarios de una determinada clase A. Utilizando A desde un test, se puede indicar a la factoria que le suministre un objeto similar a X (con la misma forma funcional externa) pero más sencillo de instanciar y reducido a la mínima expresión necesaria para el test (pseudo-objeto, “mock”).

Un ejemplo ilustrativo:

La clase A utiliza un controlador de datos X para adquirir una lista de clientes llamando a su método `getClientesDeLaProvincia(provincia)`. Pero unas veces los datos están en una base de datos remota y otras veces en un archivo XML local, (es decir, unas veces usa el controlador ‘GestorDeDatosSQL’ y otras usa el controlador ‘GestorDeDatosXML’, ambos dos controladoresX implementando un `interfaceX` que incluye el método ‘`getClientesDeLaProvincia(provincia)`’). Se podría resolver con algo así como este código:

```
...
String tipoDeFuenteDeDatos;
InterfaceX fuenteDeDatos;
ArrayList<Cliente> clientes;
...
if (tipoDeFuenteDeDatos == 'SQL')
{
    fuenteDeDatos = new GestorDeDatosSQL();
}
else if (tipoDeFuenteDeDatos == 'XML')
{
    fuenteDeDatos = new GestorDeDatosXML();
}
...
clientes = fuenteDeDatos.getClientesDeLaProvincia('Cordoba');
...
```

Pero ¿qué pasa si luego queremos extender a otros controladores, como por ejemplo a ‘GestorDeDatosJSON’ o ‘GestorDeDatosSemanticos’?...

Podemos ampliar la construcción if-elseif en el código en la clase A; *pero el trabajo de esa clase es procesar datos de clientes, no el saber de donde proceden esos datos*. Quedaría más limpio si liberamos a la clase A de esa responsabilidad extra, adjudicando esa tarea a una clase específica ad-hoc (una factoria) encargada de crear los gestores de datos necesarios. De este modo, la clase A se vería simplificada:

```
...
String tipoDeFuenteDeDatos;
InterfaceX fuenteDeDatos;
ArrayList<Cliente> clientes;
...
fuenteDeDatos = FactoriaDeGestores.getGestorDeDatos(tipoDeFuenteDeDatos);
...
clientes = fuenteDeDatos.getClientesDeLaProvincia('Cordoba');
...
```

Y toda la parte de:

```
...
if (tipoDeFuenteDeDatos == 'SQL')
{
    return new GestorDeDatosSQL();
}
else if (tipoDeFuenteDeDatos == 'XML')
{
    return new GestorDeDatosXML();
}
else if (tipoDeFuenteDeDatos == 'JSON')
{
    return new GestorDeDatosJSON();
}
else if (tipoDeFuenteDeDatos == 'RDF')
{
    return new GestorDeDatosSemanticos();
}
...
```

iría dentro del método 'getGestorDeDatos(tipo)' de la clase 'FactoriaDeGestores'.

Capítulo 5

Ejercicios

((una disculpa del autor: este capítulo esta “en construcción...” y sigo trabajando en él; queda aún mucho trabajo para completarlo.))

(nota: Ver posibles soluciones al final del libro, en el capítulo 17.)

5.1. Áreas y perímetros de figuras geométricas

5.1.1. Utilizando herencia

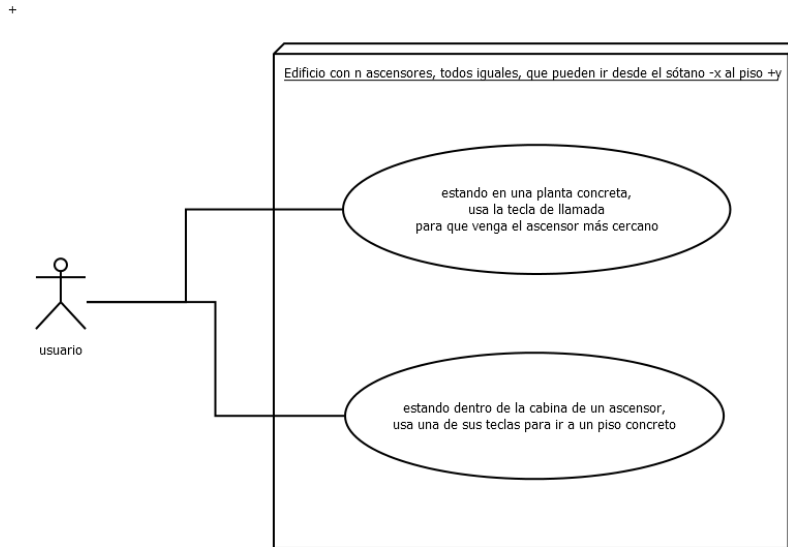
El cálculo de área y el de perímetro se calcula en el constructor de cada clase hija y (si lo hay) en el metodo setter de las dimensiones de la figura. Los métodos getter, `getArea()` y `getPerimetro()`, son iguales para todas las hijas y se limitan a devolver el valor precalculado.

5.1.2. Utilizando interface

El cálculo de area y el de perímetro se implementa en cada clase, en los métodos `getArea()` y `getPerimetro()` obligados por el Interface.

5.2. Ascensores

Caso de uso:



Sugerencia de interfaz de usuario textual:

```
Tenemos un edificio con 3 ascensores
Que pueden ir desde la planta -2 a la planta +4
```

```
Situacion actual de los ascensores: [0] [0] [0]
```

```
¿Que desea hacer?
```

```
-0- Terminar y salir.
```

```
-1- Mover un ascensor concreto a un piso concreto.
```

```
-2- Enviar un ascensor a un piso.
```

```
1
¿Que ascensor quieres mover? (1...3)
```

```
2
¿A que piso? (-2...+4)
```

```
4
```

```
Tenemos un edificio con 3 ascensores
Que pueden ir desde la planta -2 a la planta +4
```

```
Situacion actual de los ascensores: [0] [4] [0]
```

```
¿Que desea hacer?
```

```
-0- Terminar y salir
```

```
Tenemos un edificio con 3 ascensores
Que pueden ir desde la planta -2 a la planta +4
```

```
Situacion actual de los ascensores: [0] [4] [0]
```

```
¿Que desea hacer?
```

```
-0- Terminar y salir.
```

```
-1- Mover un ascensor concreto a un piso concreto.
```

```
-2- Enviar un ascensor a un piso.
```

```
2
```

```
¿A que piso? (-2...+4)
```

```
1
```

```
Tenemos un edificio con 3 ascensores
Que pueden ir desde la planta -2 a la planta +4
```

```
Situacion actual de los ascensores: [1] [4] [0]
```

```
¿Que desea hacer?
```

```
-0- Terminar y salir
```

Sugerencia de interfaz de usuario grafico:

Manejo de ascensores

Situacion actual de los ascensores: [3] [-2] [0]

El edificio tiene 4 pisos y 2 sótanos.

Estoy en el piso:

Hay 3 ascensores (y son informáticos :-), se numeran empezando por 0)

Estoy en el ascensor: y quiero ir al piso:

Capítulo 6

Datos en memoria de trabajo

6.1. Estructuras lineales (arrays, colas,...)

Han sido modo básico de almacenar información desde los primeros tiempos de la informática. Consisten en una serie lineal contigua de datos, uno detrás de otro; con soporte para las operaciones de añadir, leer, modificar y eliminar elementos de la secuencia.

La estructura lineal más básica es el típico array. Por ejemplo, en C

```
void Usar_un_array_c()
{
    char colores[5][12];
    strcpy(colores[0], "rojo");
    strcpy(colores[1], "azul");
    strcpy(colores[2], "verde");
    strcpy(colores[3], "naranja");
    strcpy(colores[4], "amarillo");

    strcpy(colores[2], "gris");

    for(int i = 0; i < 5 ; i++)
    {
        printf("%s\n", colores[i]);
    }
}
```

Por ejemplo, en C++

```
void Usar_un_array()
{
    std::array<std::string, 5> colores = {"rojo", "azul", "verde",
                                         "naranja", "amarillo"};

    colores[2] = "gris";

    for(const auto& color : colores)
    {
        std::cout << color << std::endl;
    }
}
```

```

    }

}

```

Por ejemplo, en java

```

public static void OrdenacionBoubleSort()
{
    int CANTIDADDEELEMENTOS = 5;
    Double[] ristraDeNumeros = new Double[CANTIDADDEELEMENTOS];
    ristraDeNumeros[0] = 4.6;
    ristraDeNumeros[1] = 2.1;
    ristraDeNumeros[2] = 7.8;
    ristraDeNumeros[3] = 5.0;
    ristraDeNumeros[4] = 6.2;
    System.out.println("ristra original:");
    for(int i = 0 ; i < CANTIDADDEELEMENTOS; i++)
    {
        System.out.println(ristraDeNumeros[i]);
    }

    for(int i = 0 ; i < CANTIDADDEELEMENTOS - 1 ; i++)
    {
        for(int j = 0 ; j < CANTIDADDEELEMENTOS - i - 1 ; j++)
        {
            if(ristraDeNumeros[j] > ristraDeNumeros[j+1])
            {
                Double temporal = ristraDeNumeros[j];
                ristraDeNumeros[j] = ristraDeNumeros[j+1];
                ristraDeNumeros[j+1] = temporal;
            }
        }
    }

    System.out.println("ristra después de ordenarla:");
    for(int i = 0 ; i < CANTIDADDEELEMENTOS; i++)
    {
        System.out.println(ristraDeNumeros[i]);
    }
}

```

Según el tipo de tarea para la que esté optimizada, entre las estructuras de capacidad fija se pueden distinguir:

- **ARRAYs:** Están pensados para recorrerse secuencialmente, de principio a fin. Son eficientes para insertar/eliminar registros en el final de la serie o para leer/modificar un registro cualquiera sabiendo la posición (índice) que ocupa dentro de la serie. Otras operaciones son más costosas, por ejemplo: insertar un registro en una posición concreta implica “empujar” los posteriores para hacer sitio al recién llegado; eliminar un registro concreto implica “mover” los posteriores para cubrir el hueco; buscar por contenidos implica ir leyendo toda la serie hasta dar con el registro deseado;...

- COLAs (array FIFO) (QUEUE): Están pensadas para que sea eficiente añadir nuevos registros y recuperarlos en sentido directo (primero en entrar, primero en salir). Suelen tener solo dos operaciones: meter (enqueue) y sacar (dequeue).
- PILAs (array LIFO) (STACK): Están pensadas para que sea eficiente añadir nuevos registros y recuperarlos en el sentido inverso (último en entrar, primero en salir). Suelen tener solo dos operaciones: poner (push) y quitar (pop).

Y entre las estructuras de capacidad variable se pueden distinguir:

- LISTAS ENCADENADAS: Están pensadas para que sea eficiente recorrerlas desde el principio hacia el final y para insertar/eliminar registros en cualquier punto de la lista.
- LISTAS DOBLEMENTE ENCADENADAS: Similares a las anteriores, pero el doble enlace entre elementos hace que sea eficiente recorrerlas en cualquiera de las dos direcciones: del principio al final o del final al principio.

Hace unas décadas, saber manejar bien este tipo de estructuras era vital en cualquier programa. Debido a las limitaciones de capacidad de las máquinas, era imprescindible exprimir al máximo toda su memoria y su procesador; utilizando en cada caso la estructura óptima para cada tarea.

Hoy en día, solo tiene importancia en ciertos casos muy concretos:

- operaciones realizadas muchísimas veces (por ejemplo, en el núcleo de procesamiento de un sistema operativo)
- o que han de ser muy rápidas (por ejemplo, sistemas transaccionales que ejecutan millones de transacciones por segundo)
- o que se han de completar en un tiempo máximo predeterminado (por ejemplo, sistemas críticos en tiempo real)
- o en el caso de sistemas embebidos con recursos muy limitados (por ejemplo, pequeños microcontroladores de 8bit).

Y, aún en esos casos especiales, a medida que avanza el hardware, se van difuminando cada vez más las diferencias entre controlar directamente la estructura de datos o simplemente utilizar alguno de los contenedores (listas, diccionarios,...) de la biblioteca estándar del lenguaje que estemos utilizando.

6.2. Listas (list, vector,...)

Son como un array, solo que con tamaño variable que se ajusta automáticamente según las necesidades.

Se utilizan cuando la forma principal de utilizar los objetos contenidos va a ser a base de recorrer la colección hacia adelante o hacia atrás.

```
void Usar_una_lista()
{
```

```
std::list<std::string> colores;
colores.push_back("rojo");
colores.push_back("azul");
colores.push_front("verde");
colores.push_front("naranja");
colores.push_back("amarillo");
colores.push_front("verde");

std::cout << "La lista tiene " << colores.size() << " elementos
." << std::endl;
std::cout << "El primer elemento es " << colores.front() << std
::endl;
std::cout << "Y el último es " << colores.back() << std::endl;
std::cout << "La lista original es asi:" << std::endl;
for(std::string color : colores)
{
    std::cout << color << std::endl;
}

colores.reverse();

std::cout << "Si la invertimos queda asi:" << std::endl;
for(std::string color : colores)
{
    std::cout << color << std::endl;
}

colores.sort();

std::cout << "Si la ordenamos queda asi:" << std::endl;
for(std::string color : colores)
{
    std::cout << color << std::endl;
}

colores.unique();

std::cout << "Si le quitamos duplicados queda asi:" << std::endl
;
for(std::string color : colores)
{
    std::cout << color << std::endl;
}

}
```

La lista tiene 6 elementos.
El primer elemento es verde
Y el último es amarillo
La lista original es asi:
verde
naranja
verde
rojo
azul
amarillo

```
Si la invertimos queda asi:  
amarillo  
azul  
rojo  
verde  
naranja  
verde  
Si la ordenamos queda asi:  
amarillo  
azul  
naranja  
rojo  
verde  
verde  
Si le quitamos duplicados queda asi:  
amarillo  
azul  
naranja  
rojo  
verde
```

También es importante destacar el hecho de que las bibliotecas actuales suelen permitir indicar explícitamente el tipo de elemento a almacenar dentro del contenedor

```
lista<int>, lista<string>, lista<Cliente>, lista<AlbaranDeEnvio>,  
etc.
```

Esto es muy útil; de esta forma, el compilador puede detectar y alertar de situaciones donde se intente realizar alguna operación o conversión no válida para el tipo concreto de elemento a manejar.

Por ejemplo, en c++,

```
void Usar_un_vector()  
{  
    std::vector<std::string> colores;  
    colores.push_back("rojo");  
    colores.push_back("azul");  
    colores.push_back("verde");  
    colores.push_back("naranja");  
    colores.push_back("amarillo");  
  
    colores.erase(colores.begin() + 2);  
  
    colores.insert(colores.begin() + 2, "añadido");  
  
    std::cout << "La lista tiene " << colores.size() << " elementos  
    ." << std::endl;  
    for(std::string color : colores)  
    {  
        std::cout << color << std::endl;  
    }  
}
```

Por ejemplo, en java,

```
public static void OrdenacionDeUnArrayList()
{
    java.util.ArrayList<Double> ristraDeNumeros;
    ristraDeNumeros = new java.util.ArrayList<Double>();
    ristraDeNumeros.add(3.6);
    ristraDeNumeros.add(1.1);
    ristraDeNumeros.add(6.8);
    ristraDeNumeros.add(4.0);
    ristraDeNumeros.add(5.2);
    System.out.println("ristra original:");
    ristraDeNumeros.forEach(x -> System.out.println(x));

    ristraDeNumeros.sort((x, y) -> x > y ? 1 : -1);

    System.out.println("ristra después de ordenarla:");
    ristraDeNumeros.forEach(x -> System.out.println(x));
}
```

6.3. Diccionarios (dictionary, map,...)

Son como una lista, pero asociando una clave—índice a cada elemento. Esto permite: optimizar búsquedas, evitar duplicados (no se admiten dos claves iguales), mantener los elementos ordenados en todo momento,...

Se utilizan cuando la forma principal de utilizar los objetos contenidos va a ser a base de buscar y localizar objetos individuales concretos dentro de la colección.

Todo acceso que se realice a través de las claves, será muy rápido. Los accesos que no involucren claves, se comportarán como los de una lista común.

```
struct Persona
{
    std::string ID;
    std::string nombre;
    std::string apellido;
    tm fecha_de_nacimiento;
    int otros_datos;
};

void Usar_un_diccionario()
{
    std::map<std::string, Persona> lista_de_socios;

    Persona unSocio;
    unSocio.ID = "BBBB";
    unSocio.nombre = "Benzirpi";
    unSocio.apellido = "Mirvento";
    unSocio.fecha_de_nacimiento.tm_year = 1985;
    unSocio.fecha_de_nacimiento.tm_mon = 6;
    unSocio.fecha_de_nacimiento.tm_mday = 14;

    lista_de_socios[unSocio.ID] = unSocio;
```

```

Persona otroSocio;
otroSocio.ID = "BBBC";
otroSocio.nombre = "Giacometa";
otroSocio.apellido = "Fiorine";
otroSocio.fecha_de_nacimiento.tm_year = 1985;
otroSocio.fecha_de_nacimiento.tm_mon = 6;
otroSocio.fecha_de_nacimiento.tm_mday = 14;

lista_de_socios[otroSocio.ID] = otroSocio;

std::cout << "Lista de socios del club:" << std::endl;
for(std::map<std::string, Persona>::iterator elemento =
    lista_de_socios.begin();
    elemento != lista_de_socios.end(); ++elemento)
{
    std::string clave = elemento->first;
    Persona socio = elemento->second;
    std::cout << clave << " $\rightarrow$ " << socio.nombre << " "
        << socio.apellido <<
}

}

Lista de socios del club:
BBBB $\rightarrow$ Benzirpi Mirvento
BBBC $\rightarrow$ Giacometa Fiorine

```

```

std
::
endl
;

```

6.4. Estructuras arbóreas (trees) y grafos (graphs)

Son estructuras básicas cuando se trabaja con algoritmos de clasificación o de búsqueda. Su tratamiento suele requerir programas recursivos.

De hecho, son la base algorítmica sobre la que funcionan muchas bases de datos. Si realmente necesitamos toda la potencia de las estructuras arbóreas en nuestra aplicación, salvo excepciones, suele ser mejor utilizar una base de datos en lugar de implementar esos algoritmos arbóreos directamente por nosotros mismos.

nota: Si tenemos curiosidad, ... una referencia donde profundizar en el tema. ... <https://www.programiz.com/dsa>

6.5. Bases de datos

Son necesarias cuando se trabaja con grandes cantidades de información. (nota: A día de hoy, en 2018, estaríamos hablando a partir de cientos de miles de registros o

de cientos de Mb. de almacenamiento; con cantidades de información menores, suele ser más práctico utilizar listas o diccionarios.)

Suelen ser programas dedicados, usualmente instalados en un servidor. Disponen de un ‘motor’ de tratamiento de datos (estructuras y servicios especializados para manejarlos de forma eficaz y eficiente).

Hay tres tipos principales de bases de datos:

Relacionales : Almacenan los datos en tablas (filas y columnas), relacionadas entre sí por identidades concretas entre columnas. Están diseñadas para el tratamiento de información estructurada y son muy ágiles cuando se accede a la información utilizando las identidades y relaciones (índices) predefinidas en su estructura. Se conocen también como bases de datos SQL, porque es el lenguaje estándar que se suele utilizar para trabajar con ellas.

Semánticas : Trabajan con triadas ‘objeto-|tipo de relación|-objeto’, (triples RDF). De esta forma, junto a los propios datos en sí, se almacena la naturaleza de los mismos y las relaciones entre ellos. Así, el sistema de tratamiento puede “comprender el significado” de cada dato, ponerlo “en contexto” y “descubrir” (inferir) nueva información que no estaba presente de forma explícita en los propios datos. Estas bases de datos se suelen consultar con el lenguaje SPARQL. En ellas suele ser importante disponer de una ontología especializada según el campo de aplicación; esta ontología se suele expresar con el lenguaje OWL.

No-relacionales (o non-SQL) : Tienen distintas formas de almacenar datos y diferentes estrategias de manipulación de los mismos. Suelen estar adaptadas al tratamiento de datos no estructurados (por ejemplo: imágenes, sonidos, series temporales,...) o de enormes cantidades de datos (BigData).

6.6. La importancia de especificar el tipo de los datos: “generics”

En las primeras versiones de las colecciones, era responsabilidad del programador saber qué clase de objetos contenía cada colección. Quedando bajo su responsabilidad realizar las correspondientes conversiones de tipo (cast) para utilizarlos correctamente. De esta forma, los posibles errores no se detectan hasta el momento de ejecutar dichas conversiones.

En todas las versiones recientes se utilizan plantillas genéricas (generics) para especificar en el propio código la clase de objetos a almacenar en cada colección utilizada. De esta forma, pasa a ser responsabilidad del compilador detectar cualquier error que pudiera haber.

<https://www.tutorialsteacher.com/csharp/csharp-generic-collections>

6.7. Las propias funciones como un tipo de dato más: “expresiones Lambda”

El paradigma de *programación funcional* es aquel en el que las propias funciones son un tipo de dato más. Y, por tanto, se pueden pasar como parámetros a otras funciones o se pueden devolver como resultado de una función. Es una manera de programar diferente, con muchas ventajas y que posiblemente sea la que se emplee habitualmente en un futuro.

Aquí solo voy a citar un uso puntual de ese paradigma. Un uso que está siendo cada vez más habitual en cada vez más lenguajes, aunque sean orientados a objeto y no funcionales.

Las expresiones Lambda son una manera rápida de definir una función sobre la marcha. Se suelen utilizar para pasar una función auxiliar a un método que se encarga de aplicarla sobre el conjunto de elementos de una colección.

Por ejemplo, para definir el criterio de ordenación de una lista de alumnos con el método ‘.sort’. Se podría definir de forma tradicional:

```
class ComparadorPorNotasMedias implements java.util.Comparator<Alumno>
{
    @Override
    public int compare(Alumno unAlumno, Alumno otroAlumno)
    {
        if(unAlumno.notaMedia > otroAlumno.notaMedia)
        {
            return 1;
        }
        else
        {
            return -1;
        }
    }
}
```

```
listaDeAlumnos.sort(new ComparadorPorNotasMedias());
```

O se podría definir con una expresión Lambda:

```
listaDeAlumnos.sort((unAlumno, otroAlumno) -> {
    if(unAlumno.notaMedia > otroAlumno.notaMedia)
    {return 1;}
    else
    {return -1;} });
```

Otros dos ejemplos, uno para definir el criterio de eliminación con el método ‘.removeIf’ y otro para definir la acción a realizar sobre cada elemento de la colección con el método ‘forEach’:

```
listaDeAlumnos.removeIf(x -> x.poblacion == 'Villabenzirpi');
```

```
listaDeAlumnos.forEach(x -> x.notaMedia = x.notaMedia * 1.03);
```

Capítulo 7

Datos almacenados (persistencia)

7.1. Archivos de texto “plano”

Son (¿Eran?) los más utilizados, por su sencillez. Normalmente suelen contener un registro por cada fila de texto.

Al no disponer de metadatos, requieren que sea su usuario quien sepa interpretar de forma adecuada cada dato dentro de cada registro. En algunas ocasiones suele disponerse una primera fila con títulos, para ayudar a dicha interpretación.

Los mecanismos más habituales para separar datos entre sí suelen ser:

- Disponer una longitud fija para cada dato almacenado. En aquellos casos donde no se rellenen el número de caracteres reservados, se completan con espacios en blanco (padding).

Pedro	Rodriguez Pike	23/04/1973	56,3	168,5	22/11/2021
Marta	Batiato Rueda	21/09/1984	45,8	170,2	22/11/2021
Tej	Mirvento	25/11/1967	100,2	172	22/11/2021
Luisa	Perez Bila	23/07/1985	56,2	161,5	22/11/2021

- Utilizar una cierta combinación especial de caracteres para marcar las fronteras entre un dato y otro. Los datos faltantes se reconocen por dos separadores seguidos sin nada entre ellos.

(nota: los separadores más utilizados son la coma [,] o el punto y coma [;] o el tabulador [] ; sin embargo, un separador que funciona especialmente bien es la combinación “espacios - dos puntos - espacios” [:], permitiéndose cualquier número de espacios tanto por delante de los dos puntos como por detrás.

nombre , apellidos , fecha_nacimiento , peso_kg , altura_cm , fecha_medicion
Pedro ,Rodriguez Vazquez,23/04/1973,"56,3","168,5",22/11/2021
Marta,Batiato Rueda,21/09/1984,"45,8","170,2",22/11/2021
Tej,Mirvento,25/11/1967,"100,2",172,22/11/2021
Luisa ,Perez Bila ,23/07/1985,"56,2","161,5",22/11/2021

nombre;apellidos;fecha_nacimiento;peso_kg;altura_cm;fecha_medicion
Pedro;Rodriguez Vazquez;23/04/1973;56,3;168,5;22/11/2021
Marta;Batiato Rueda;21/09/1984;45,8;170,2;22/11/2021
Tej;Mirvento;25/11/1967;100,2;172;22/11/2021
Luisa;Perez Bila;23/07/1985;56,2;161,5;22/11/2021

nombre	:	apellidos	:	nacimiento	:	peso_kg	:	altura_cm	:	fecha_medicion
Pedro	:	Rodriguez Pike	:	23/04/1973	:	56,3	:	168,5	:	22/11/2021
Marta	:	Batiato Rueda	:	21/09/1984	:	45,8	:	170,2	:	22/11/2021
Tej	:	Mirvento	:	25/11/1967	:	100,2	:	172	:	22/11/2021
Luisa	:	Perez Bila	:	23/07/1985	:	56,2	:	161,5	:	22/11/2021

7.2. Expresiones regulares

Al principio pueden parecer intimidantes. Pero su potencia y sencillez para tratar información textual compensa con creces el esfuerzo inicial de aprender a utilizarlas.

Se trata de formar un patrón y luego ir pasandolo a lo largo de todo el texto para ver qué partes del mismo encajan con ese patrón.

La riqueza expresiva del lenguaje de expresiones regulares (regex) permite definir patrones con una precisión y una capacidad de discriminación muy altas.

Se pueden utilizar para:

- Verificar si un texto tiene o no una determinada forma. (Por ejemplo, la expresión

`^[+-]*[0-9]+$`

solo encaja con algo compuesto por uno o más dígitos, con o sin un signo + o un signo - por delante; es decir, solo encaja con un número entero.)

- Dividir el texto en campos según un cierto separador (split). (Por ejemplo, la expresión

`\s+:\s+`

puede localizar separadores compuestos por uno o más espacios seguidos del signo dos puntos y seguido por uno o más espacios.)

- Extraer ciertas partes. (Por ejemplo, la expresión

`^([+-]*[0-9]+),*([0-9]+)*$`

permite localizar dos grupos de números separados por una coma, el primer grupo corresponde a la parte entera del número y el segundo grupo a la parte decimal. Si no hay coma, solo se extrae la parte entera; es decir, la presencia de la coma y del segundo grupo de números es opcional.)

- Buscar ciertos patrones y utilizar los trozos extraídos para rehacer o modificar partes del texto.
- Localizar un patrón con un cierto contexto por delante o por detrás de él.
- etc.

<https://regexr.com/>

<https://docs.microsoft.com/es-es/dotnet/standard/base-types/regular-expression-language-quick-reference>

<https://docs.python.org/3/howto/regex.html>

7.3. Archivos XML

Utilizando el formato XML (Extensible Markup Language) es posible identificar con precisión cada dato concreto dentro de cualquier estructura de datos, incluso en estructuras con múltiples niveles de anidamiento o en archivos con datos faltantes o sobrantes. Cada registro y cada dato están identificados mediante sus correspondientes etiquetas descriptoras.

```
<pacientes>
  <persona>
    <nombre>Pedro</nombre>
    <apellidos>Rodriguez Pike</apellidos>
    <fecha_nacimiento>23/04/1973</fecha_nacimiento>
    <altura unidad="cm">168,5</altura>
    <mediciones>
      <medicion>
        <peso unidad="kg">56,3</peso>
        <fecha_medicion>22/11/2021</fecha_medicion>
      </medicion>
      <medicion>
        <peso unidad="kg">64,7</peso>
        <fecha_medicion>15/12/2021</fecha_medicion>
      </medicion>
    </mediciones>
  </persona>
  <persona>
    <nombre>Marta</nombre>
    <apellidos>Batiato Rueda</apellidos>
    <fecha_nacimiento>21/09/1984</fecha_nacimiento>
    <altura unidad="cm">170,2</altura>
    <mediciones>
      <medicion>
        <peso unidad="kg">45,8</peso>
        <fecha_medicion>22/11/2021</fecha_medicion>
      </medicion>
      <medicion>
        <peso unidad="kg">46,2</peso>
        <fecha_medicion>15/12/2021</fecha_medicion>
      </medicion>
    </mediciones>
  </persona>
</pacientes>
```

La propia estructura del archivo está definida mediante metadatos (DTD o Schema). De tal forma que es posible comprobar si un archivo XML está o no bien formado, con arreglo a una determinada especificación para un determinado uso.

Este tipo de archivos se suelen tratar utilizando una biblioteca de aplicación específica (un ‘parser’ XML). El tratamiento se puede realizar siguiendo uno de estos métodos:

- DOM: cargar todo el archivo XML en memoria y tratarlo globalmente.
- SAX: ir pasando secuencialmente el archivo por el parser, línea a línea, e ir

reaccionando según lo que se va encontrando.

Admiten también tratamientos más complejos, usando transformaciones XSLT.

7.4. Archivos JSON

JSON (JavaScript Object Notation) es un formato muy popular para almacenamiento e intercambio de información. En cierto aspecto es similar a XML, pero con mucha menos repetición de etiquetas.

```
"pacientes": [
  {
    "persona" {
      "nombre": "Pedro",
      "apellidos": "Rodriguez Pike",
      "fecha_nacimiento": "23/04/1973",
      "altura_cm": "168,5",
    }
    "mediciones": [
      {
        "peso_kg": "56,3",
        "fecha_medicion": "22/11/2021"
      },
      {
        "peso_kg": "64,7",
        "fecha_medicion": "15/12/2021"
      }
    ]
  },
  {
    "persona" {
      "nombre": "Marta",
      "apellidos": "Batiato Rueda",
      "fecha_nacimiento": "21/09/1984",
      "altura_cm": "170,2",
    }
    "mediciones": [
      {
        "peso_kg": "45,8",
        "fecha_medicion": "22/11/2021"
      },
      {
        "peso_kg": "46,2",
        "fecha_medicion": "15/12/2021"
      }
    ]
  }
]
```

7.5. Archivos CBOR u otras representaciones binarias

Las representaciones binarias son muy compactas, ocupan mucho menos espacio que los formatos textuales antes citados. Pero tienen la contrapartida de ser ilegibles si no se conocen las especificaciones exactas de la representación concreta utilizada.

CBOR (Concise Binary Object Representation) es, en cierta medida, una versión compacta de JSON.

7.6. Bases de datos

Se utilizan cuando se manipulan enormes cantidades de datos, con relaciones complejas entre ellos. Suelen requerir de programas específicos para su tratamiento, denominados motores de base de datos.

Para tratar con bases de datos relacionales (las más habituales), es imprescindible conocer el lenguaje SQL.

Para tratar con otros tipos de bases de datos (nosql databases), se utilizan otros lenguajes. (Por ejemplo SPARQL para bases de datos semánticas.)

Cuando la cantidad de datos es especialmente enorme o las relaciones entre los datos son especialmente complejas, suelen ser de gran ayuda las técnicas especializadas de minería de datos (data-wharehouse, data-mining, data-analytics,...).

Capítulo 8

Ejercicios

((una disculpa del autor: este capítulo esta “en construcción...” y sigo trabajando en él; queda aún mucho trabajo para completarlo.))

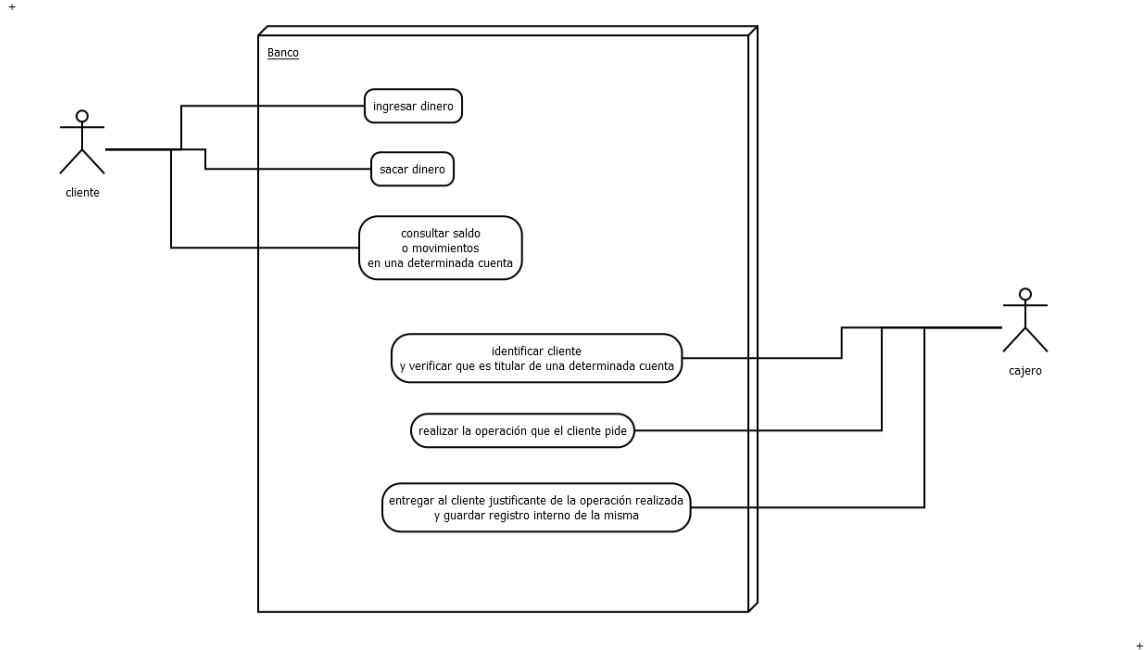
(nota: Ver posibles soluciones al final del libro, en el capítulo 17.)

La idea es utilizar las colecciones pertinentes para la gestión de datos en memoria, según el ejercicio. Y utilizar persistencia para guardar/recuperar esos datos a/desde disco.

Es conveniente aplicar diversos tipos de persistencia (txt, XML, JSON, BD) en cada ejercicio. Así se puede trabajar el desacople entre lógica y almacenamiento; si está bien planteado, se debe poder cambiar el formato de persistencia sin afectar en nada al resto del programa (ni a la lógica ni, mucho menos, al interfaz de usuario). nota: ver más adelante, la sección 14.3 en la página 70.

8.1. Cuentas bancarias

Caso de uso:



Nota: Para simplificar el ejercicio, cada cuenta tendrá un solo cliente titular. Aunque un mismo cliente podrá ser titular de varias cuentas.

Tarjetas de requisitos/funcionalidades:

<p>Como cliente, necesito ingresar dinero.</p> <p>La forma de validar sera:</p> <ul style="list-style-type: none"> ■ consulto saldo. ■ ingreso dinero. ■ vuelvo a consultar saldo y compruebo que el ingreso se ha realizado correctamente. 	<p>Como cliente, necesito consultar saldo.</p> <p>La forma de validar sera:</p> <ul style="list-style-type: none"> ■ pido consulta de saldo. ■ obtengo los últimos 8 movimientos de la cuenta y el saldo actual en ella.
--	--

<p>Como cliente, necesito sacar dinero.</p> <p>La forma de validar sera:</p> <ul style="list-style-type: none"> ■ consulto saldo. ■ saco dinero. ■ vuelvo a consultar saldo y compruebo que el reintegro se ha registrado correctamente. 	<p>Como cajero, necesito identificar al cliente y verificar el numero de cuenta.</p> <p>La forma de validar sera:</p> <ul style="list-style-type: none"> ■ introduzco el número de cuenta y consulto los datos del cliente titular de la misma. ■ compruebo esos datos con respecto al DNI del cliente. <p>Puede darse el caso de que el cliente no me de el número de cuenta. En ese caso:</p> <ul style="list-style-type: none"> ■ introduzco el DNI del cliente y consulto las cuentas de las que es titular (numero y saldo). ■ el cliente me indica en cual de las cuentas desea operar.
<p>Como cajero, necesito atender a las operaciones que el cliente solicite.</p> <p>La forma de validar sera:</p> <ul style="list-style-type: none"> ■ puedo realizar un ingreso de dinero en una cuenta. ■ puedo realizar una retirada de dinero en una cuenta. ■ puedo consultar los últimos movimientos y el saldo. 	<p>Como dueño del banco, necesito tener un registro seguro de todas las transacciones realizadas.</p> <p>La forma de validar sera:</p> <ul style="list-style-type: none"> ■ realizar cada una de las operaciones posibles en el sistema: consultar, crear, modificar, etc. ■ consultar el registro para ver que todas y cada una de las operaciones se han registrado correctamente: fecha, hora, tipo de operación, datos afectados por la operación (nota: en las modificaciones, incluir tanto los datos “antes” como los datos “después”).

8.2. Gestión de accesos

Un mapa para gestionar usuarios autorizados a entrar. Y una lista para almacenar los accesos (o intentos de acceso) que se han producido.

8.3. Gestión de pedidos

Manejar datos del tipo “cabecera-líneas”. Cabecera con los datos generales del pedido: fecha, cliente, dirección de envío, persona que recibirá el envío,... Y líneas con los datos de los artículos solicitados: código, descripción, cantidad,...

Capítulo 9

Tratamiento de incidencias y errores

9.1. Excepciones

El tratamiento de excepciones entra en acción cuando sucede algo que no debería suceder (una excepción) y el flujo normal del programa se ve gravemente alterado por circunstancias ajenas fuera del control de este.

El mecanismo más habitual para tratarlas es la construcción try-catch-finally:

- try: la parte de código a vigilar por si se produjera una excepción en alguna de sus sentencias. Se ha de tener en cuenta que la ejecución se detiene cuando se produce una excepción en una sentencia; y el resto de sentencias en el ‘try’ situadas detrás de ella no se ejecutarán. La regla de oro aquí es limitar al mínimo las sentencias dentro de cada ‘try’. Es decir, huir de actitudes donde se tienda a poner todo el programa en un enorme try simplemente “por si acaso”. Es necesario pensar y prever dónde pueden darse realmente excepciones (situaciones realmente fuera del control del programa).
- catch: la parte que recoge la excepción y reacciona a ella. La regla de oro aquí es recoger siempre excepciones concretas. Es decir, huir de simplificar y recoger “cualquier excepción”. Es necesario pararse a pensar cuáles pueden ser las excepciones que se pueden dar y diseñar reacciones específicas para cada una de ellas (aunque luego resulte que esas reacciones sean la misma para todas).
- finally: la parte que se ha de ejecutar siempre, haya o no haya excepción. Se suele utilizar en aquellas situaciones en que, pase lo pase, ciertos aspectos necesarios para continuar el flujo normal del programa ha de quedar siempre garantizados. Por ejemplo, para cerrar una conexión que de otra manera quedaría abierta si se da una excepción antes de terminar toda la parte ‘try’.

Como regla general:

las excepciones **”lanzarlas cuanto antes, interceptarlas lo más tarde posible”**.

Es contraproducente interceptar excepciones allá donde no se puede hacer casi nada con ellas...

...mejor dejarlas subir por la pila (stack) de llamadas en la aplicación (throws...)

...hasta el sitio donde se pueda hacer algo realmente útil.

Por ejemplo, una excepción (situación anómala) que se produzca al guardar algo en disco:

- El módulo gestor de disco no tiene información suficiente, no podría hacer nada más allá de reintentar y reintentar la escritura.
- Sin embargo entre el gestor de lógica de negocio y el gestor de interface si que podrían hacer algo como por ejemplo: avisar al usuario de que necesita liberar espacio en disco y proponerle un botón para que reintente el guardado una vez haya liberado espacio.

9.2. Registro (log)

Es muy útil conservar información sobre la ejecución interna del programa. Sobre todo conservar un registro de los errores. De tal forma que se pueda realizar un análisis forense a posteriori en caso de problemas.

Es útil en cualquier aplicación. Pero especialmente importante en aplicaciones de procesamiento masivo desatendido de datos (procesos batch) o en aplicaciones con lógicas internas complejas.

La información de cada evento ha de incluir el contexto necesario para conocer cuándo/dónde se han producido.

Tener en cuenta que el almacenamiento puede crecer de forma desmesurada si no se limpia de vez en cuando.

truco: Disponer de dos archivos de tamaño fijo (x registros), cuando uno de ellos se llena se vacía el otro y se pasa a almacenar los registros en él; de esta forma, siempre se conservan por lo menos los últimos x registros (en el peor caso) o casi los últimos 2x registros (en el mejor caso). nota: En algunos sistemas operativos puede haber problemas para vaciar un archivo y ponerse a escribir inmediatamente en él. En esos casos, usar tres archivos: cuando se llena uno, se vacía otro y se pasa a escribir en otro que se había vaciado en la limpieza anterior.

Tener en cuenta que unos eventos pueden requerir ser comunicados a alguien de forma inmediata y notoria (aviso en pantalla o activación/envío de una alarma), pero otros eventos no.

truco: Establecer una jerarquía de eventos (Fatal, Error, Warning, Info, Debug, ALL) y un mecanismo para determinar cómo se ha de registrar cada tipo de evento. Suele ser conveniente que este mecanismo permita al usuario configurar de forma sencilla cómo desea que se trate cada tipo de evento; es decir, cuales requieren dar aviso además de ser registrados y cuales solo ser registrados.

Tener en cuenta que puede ser interesante poder consultar el registro en tiempo real (monitorización continua) o desde otras ubicaciones (monitorización remota).

truco: Esto requiere capacidades de comunicación a través de la red. Siendo muy importante que dichas comunicaciones no ralenticen ni, mucho menos, provoquen errores en caso de problemas en la red.

<https://logging.apache.org/log4j/2.x/manual/index.html>

<https://logging.apache.org/log4j/2.x/manual/usage.html>

9.3. Ejercicios

El tratamiento de excepciones es algo inherente a ciertas operaciones (por ejemplo, tratar con archivos o con la red o con otros periféricos externos); es una necesidad que surge en cualquier programa. Por ello, en lugar de hacer ejercicios específicos para este tema, es mejor tratar este aspecto cuando se presente de forma natural.

El registro es algo que se puede añadir a cualquiera de los otros ejercicios planteados. Por ello, tampoco se plantean ejercicios específicos para este tema.

Capítulo 10

Comunicaciones

Cuando se necesita transmitir/recibir información hacia/desde otro ordenador conectado a través de una red.

10.1. Usando protocolo HTTP

Aunque habitualmente lo relacionamos con la transmisión de páginas web, se puede emplear para transmitir cualquier otro tipo de información.

Suele ser la base de las ‘aplicaciones web’ y de los ‘servicios REST’.

Si se dispone de una biblioteca de funciones que lo implemente (y casi todos los lenguajes modernos la tienen), es un protocolo bastante simple de utilizar:

- Un ‘cliente’ pide algo a un ‘servidor’,
- y el ‘servidor’ cumple esa petición.

Los métodos disponibles están estandarizados y son:

GET, HEAD para recibir información del servidor.

POST, PUT, PATCH para enviar información al servidor.

DELETE para eliminar información en el servidor.

TRACE, OPTIONS, CONNECT para manejar ciertos aspectos de la comunicación entre el cliente y el servidor.

<https://tools.ietf.org/html/rfc7231>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

...

El uso más habitual de esos métodos suele ser:

- **GET**: pedir un recurso, leer datos del servidor.
- **POST**: enviar unos datos al servidor (por ejemplo, el contenido de los campos de un formulario que el usuario acaba de rellenar), y/o solicitar que el servidor realice una determinada acción.

- PUT: sustituir un recurso completo (todos los datos de ese recurso) actualizarlo con los datos que se envían.
- PATCH: sustituir parte de un recurso, actualizarlo con los datos (parciales) que se envían.
- DELETE: borrar un recurso, eliminarlo del servidor.

10.2. Usando protocolos tcp/ip

Son la base sobre la que se fundamentan muchas de las comunicaciones entre equipos de lo más diverso. Su amplia difusión se debe a que sus especificaciones se establecen por consenso público: <https://www.ietf.org/standards/>

Resumiendo mucho, tenemos dos maneras básicas de trabajar:

UDP (User Datagram Protocol) : permite enviar pequeños mensajes con muy poco trabajo. El protocolo udp funciona bajo un estilo de “envía y olvida”, es decir: no garantiza la recepción en destino.

<https://tools.ietf.org/html/rfc768>

...

TCP (Transmission Control Protocol) : permite establecer un canal formal (socket) de comunicación entre dos puntos concretos. El protocolo tcp permite un intercambio de información controlado.

<https://tools.ietf.org/html/rfc675>

<https://tools.ietf.org/html/rfc761>

...

Nota: El protocolo HTTP citado en la sección anterior, es parte también de la familia de protocolos tcp/ip. De hecho, se apoya sobre el protocolo TCP; usandolo como base del intercambio de datos entre el equipo ‘cliente’ y el equipo ‘servidor’.

10.3. Usando protocolos especializados para un tipo de tarea concreta

En el mercado existen multitud de estándares y de protocolos para manejar flujos de datos de lo más diverso. Cuando se tenga entre manos algo donde las comunicaciones sean muy importantes, merece la pena buscar un poco para encontrar el protocolo más adecuado para manejarlas.

Algunos ejemplos ilustrativos:

<https://mqtt.org/>

<https://coap.technology/>

<https://nats.io/>

<https://zeromq.org/>

<https://www.amqp.org/>

Capítulo 11

Ejercicios

((una disculpa del autor: este capítulo no lo he abordado aún; el libro esta “en construcción...” y sigo trabajando en él; queda mucho trabajo para completarlo.))

(nota: Ver posibles soluciones al final del libro, en el capítulo 17.)

Capítulo 12

Procesamiento paralelo

Una forma de realizar un trabajo más rápido, es poner más trabajadores trabajando en ello. Pero, ¡ajo!, es vital que los trabajadores no se estorben entre sí, ni se bloqueen o retrasen excesivamente por esperarse unos a otros.

Es más, hay situaciones en que el procesamiento paralelo puede ser imposible. Por ejemplo:

- Tareas que, por su intrínseco carácter secuencial, solo pueden aprovechar un único trabajador.
- Tareas que, por los requisitos de sincronización o de preparación que tienen, llega un punto en que el coste de incorporar más trabajadores es mayor que el beneficio aportado por esos trabajadores adicionales.

De todas formas, en todos los sistemas siempre ha habido necesidad de realizar varias tareas en paralelo. Si no es más, para aprovechar los “espacios muertos” entre los momentos de interacción con el usuario (el aspecto más lento del sistema); o para refrescar convenientemente la presentación al usuario según se están realizando tareas largas.

Hasta hace unos años, este paralelismo era virtual (una sola unidad de procesamiento atendía todas las tareas) y suponía una carga adicional al sistema (la sobrecarga de congelar una tarea para reactivar otra, y de hacerlo a una velocidad tal que todas las tareas parecieran estar funcionando a la vez). De ahí que solo se utilizara cuando era estrictamente necesario.

Pero hoy en día, el paralelismo es real. El hardware de computación está llegando al límite de frecuencia de reloj y de optimización interna de cada unidad. Para seguir mejorando, los fabricantes han optado por integrar múltiples unidades de computación (cores) en cada procesador y multiples (co)procesadores en cada sistema.

Es decir, la programación de software paralelo está cobrando cada vez más importancia. Hoy en día es imprescindible para aprovechar toda la capacidad de los sistemas hardware.

12.1. Estrategias para repartir tareas

Este es el “caballo de batalla” del procesamiento paralelo: cómo dividir cada trabajo a realizar en múltiples tareas que puedan ser ejecutadas de forma independiente y cómo ir recombining resultados parciales de todas ellas para obtener el resultado final perseguido.

Todo ello sin que las múltiples tareas se “estorben” entre sí.

pendiente investigar y escribir acerca de las principales estrategias para hacer esto.

12.2. Salvaguardas para evitar problemas

Un aspecto básico es el que por lo menos unas tareas no estropeen el trabajo de otras. Sobre todo cuando varias de ellas han de turnarse para compartir un mismo recurso del sistema.

12.2.1. operaciones atómicas

Atención al modificar cualquier dato, la secuencia leer-modificar-escribir ha de hacerse siempre de forma monolítica (atómica) (en un solo paso).

Esto es debido a que, en un entorno con múltiples líneas de ejecución simultáneas, puede suceder que tras leer un ejecutante una cierta posición de memoria para trabajar con su valor, otro ejecutante lea el valor o escriba otro valor en dicha posición antes de que el primero termine de escribir su resultado.

El ejemplo básico de operación atómica es la secuencia leer-modificar-escribir un dato. Pero pueden existir otras situaciones donde sea necesario asegurar que se realiza “en un solo paso” una secuencia de operaciones más compleja.

Es decir: prestar siempre atención a qué podría suceder si entremedio de una secuencia de operaciones se da alguna “interferencia” externa. Si puede dar lugar a problemas, hay que asegurarse de que esa secuencia siempre se completa “en un solo paso” (sin posibilidad de interferencia externa).

12.2.2. transacciones

Otro tipo de operación monolítica es aquella donde es necesario garantizar su ejecución íntegra y completa. En este tipo de operaciones, el problema no son posibles “interferencias” externas en medio de las operaciones, sino una posible ejecución incompleta de la secuencia de operaciones.

nota: Esta situación no es exclusiva del procesamiento paralelo. Puede darse también en sistemas monotarea/monousuario. Es muy habitual, por ejemplo, trabajando con bases de datos relacionales, cuando se modifican datos en varias tablas relacionadas entre sí.

La solución suele consistir en un mecanismo para:

- Marcar el inicio de la secuencia de operaciones (start transaction).
- Ir realizando las operaciones.
- Marcar el final de la secuencia de operaciones (commit transaction)

Si, por lo que sea, la secuencia no llega a su final (commit), el mecanismo transaccional ha de asegurar que se revierten todas las operaciones realizadas hasta el momento de la interrupción (rollback transaction).

nota: Suele ser habitual que el mecanismo transaccional, en lugar de las acciones tal cual, las vaya haciendo “de forma virtual”, anotándolas en una lista (log de transacción). Si se llega al final (commit), el mecanismo transaccional lee la lista y realiza las acciones realmente. Si no se llega al final, simplemente borra la lista y no hace nada.

Es decir: siempre se garantiza que se ejecutan todas las operaciones de la secuencia o que no se ejecuta ninguna de ellas.

12.2.3. semáforos

Atención al utilizar un recurso. En sistemas paralelos, los recursos que admitan solo un único usuario cada vez, han de tener siempre un mecanismo que impida intentos de uso concurrente.

La solución más habitual suele ser un semáforo que indique si el recurso está en uso o no. La tarea que vaya a utilizar el recurso ha de:

- Leer el semáforo para ver si el recurso está libre.
- Si está ocupado, esperar y volver a leer el semáforo más tarde.
- Si está libre, “apropiarse” del semáforo y señalar ‘recurso ocupado’.
- Al terminar el trabajo, señalar ‘recurso libre’ y “liberar” el semáforo.

El tema de “apropiarse” del semáforo, es para evitar que otra tarea cambie el estado del mismo mientras el recurso está en uso. nota: Según ciertos detalles de cómo manejar estos semáforos, podemos tener distintas “variantes semaforicas”: semaphore, flag, token, mutex, monitor, signal,...

nota: En los sistemas más sofisticados, suele haber una cola para acceder al recurso. Cada tarea que vaya a usar el recurso “se apunta” en la cola. El mecanismo de gestión de la cola va asignando el uso del recurso según un cierto algoritmo de prioridades. Obviamente, nadie puede utilizar el recurso sin pasar por la cola.

12.2.4. bloqueos

Un problema grave que se puede derivar del uso de semáforos o similares es “el bloqueo mortal” (dead lock). Se da cuando una tarea tiene bloqueado un recurso y

está esperando a que se libere otro para completar su trabajo y liberar el recurso. Pero, mientras tanto, otra tarea ha bloqueado ese segundo recurso y está esperando al primero para completar su trabajo y liberar el suyo.

Es decir, `tarea1` tiene bloqueado `recurso1` y está esperando a `recurso2`. Mientras que `tarea2` tiene bloqueado `recurso2` y está esperando a `recurso1`.

12.2.5. Programación funcional

Como el paradigma funcional no tiene variables mutables, es mucho más difícil que se puedan dar interacciones no deseadas entre distintos caminos de ejecución.

A priori, parece el paradigma ideal para aprovechar plenamente las capacidades del procesamiento paralelo.

12.3. Algunos usos del procesamiento paralelo

12.3.1. Multitarea

Por ejemplo, para evitar que una tarea muy larga acapare el sistema y parezca que este ha quedado congelado por no visualizar lo que está haciendo. O, por ejemplo, para aprovechar los espacios de tiempo entre los lentos procesos humanos; de tal forma que se pueda realizar otros trabajos mientras se atiende el interfaz de usuario. O, por ejemplo, para atender a varios eventos o a varios usuarios a la vez.

Cuando cohabitan varias tareas, el control de las mismas puede ser:

- Colaborativo (non-preemptive): Cada tarea se preocupa de comportarse “con decoro” y ceder tiempo de ejecución a las demás tareas que pudiera haber activas en cada momento. Es decir, una vez una tarea está en ejecución no puede ser interrumpida por ninguna otra.
- Organizado (preemptive): Una tarea maestra se encarga de ir gestionando el tiempo y los recursos para las demás tareas, asignando a cada una lo que sea posible teniendo en cuenta ciertas prioridades preestablecidas. Es decir, algunas tareas tienen más prioridad que otras y pueden interrumpirlas cuando sea necesario.

Cuando múltiples tareas comparten un mismo recurso, pueden surgir situaciones problemáticas. De ahí que sea necesario contar con los adecuados mecanismos de control de acceso para asegurar que el comportamiento de esos recursos compartidos no depende del orden en que las tareas accedan a ellos. (thead-safe)

La coordinación de procesos, hebras y recursos es un tema con problemáticas complejas y sutiles. Es importante conocer y utilizar las bibliotecas específicas que traiga el compilador para tratar de forma segura con estos elementos.

12.3.2. Gráficos

El procesamiento de gráficos ha sido uno de los primeros usos donde se han venido utilizando múltiples (co)procesadores. Hoy en día prácticamente todos los sistemas tienen algún tipo de GPU (Graphics Processing Unit) complementando la CPU (Central Processing Unit) y liberándola de la tarea de manejar la pantalla.

Cuando se vaya a trabajar con gráficos en pantalla, procurar utilizar bibliotecas y funciones que aprovechan dicha GPU. Estas bibliotecas (OpenGL, DirectX, Metal,...) suelen ser algo más complejas de usar que las “tradicionales” (gtk, GDI, QuickDraw,...), pero merece la pena utilizarlas.

12.3.3. Cálculos

Cuando se realizan grandes cálculos de ingeniería o científicos. O cuando se procesan enormes cantidades de datos para destilar información. Es importante aprovechar plenamente las memorias caché, las unidades vectoriales y los múltiples núcleos que tienen los procesadores actuales.

Más aún si se están utilizando clusters con cientos o miles de esos procesadores, dedicados expresamente a tareas HPC (High Performance Computing).

Merece utilizar alguna de las bibliotecas y plataformas específicas que están siendo desarrolladas para este trabajo. Estas plataformas suelen cubrir aspectos tales como: alinear datos, compartir datos de partida, encadenar procesamientos parciales, sincronizar tareas, consolidar resultados parciales, etc.

<https://www.openmp.org/>

<https://developer.nvidia.com/cuda-zone>

<https://docs.microsoft.com/es-es/dotnet/standard/parallel-programming/>

<https://software.intel.com/content/www/us/en/develop/tools/parallel-studio.html>

Capítulo 13

Ejercicios

((una disculpa del autor: este capítulo no lo he abordado aún; el libro esta “en construcción...” y sigo trabajando en él; queda mucho trabajo para completarlo.))

(nota: Ver posibles soluciones al final del libro, en el capítulo 17.)

Capítulo 14

Organización interna del código

La adecuada organización interna del código, es uno de los aspectos de la programación más complicados de aprender.

Solo se puede llegar a dominar a base de participar en muchos proyectos (grandes) de diferentes ámbitos. Participando en ellos de forma activa, prestando atención a lo que suele tender a facilitar las cosas o a complicarlas. En el largo plazo, pensando en toda la vida útil del software.

Obviamente, también ayuda ir recogiendo ideas y aprendiendo de la experiencia de otras personas (libros, conferencias,...). Probandolas en la medida de nuestras posibilidades.

14.1. Orientación a eventos (event-driven)

Es una arquitectura muy típica en cualquier aplicación que tenga un interfaz gráfico de usuario (basado en ventanas, botones, controles,...) o un interface hardware específico (basado en teclas, pulsadores, luces de aviso, pantallas informativas,...)

Hay un bucle central (event loop)(event dispatcher) que está continuamente vigilando “lo que sucede...” en el sistema (una tecla pulsada, un clic de ratón sobre un botón, un mensaje que llega a través de las comunicaciones, un timer que alcanza la cuenta de tiempo preprogramada, un cierto valor en una determinada variable, etc) y que va llamando a la función correspondiente para hacerse cargo de cada “suceso”.

Una aplicación se construye a base de indicar qué evento ha de llamar a qué función. O, visto desde otro punto de vista, a base de indicar qué función responde a qué evento.

Algunos aspectos a tener en cuenta:

- Un mismo evento puede tener varias funciones que respondan a él. El bucle irá llamando a todas ellas según corresponda.
- Una función, dentro de su ejecución, puede crear y lanzar eventos. Haciendo que el bucle los procese como cualquier otro evento del sistema.

- El bucle central puede delegar en otros bucles secundarios la vigilancia de ciertos eventos. Pero sigue siendo responsabilidad del bucle central llamar a esos bucles secundarios de forma periódica.
- Ciertas funciones se pueden ejecutar de forma asíncrona. Es decir, sin que el bucle espere a que termine la ejecución de la función.
- Es importante que ninguna función pueda bloquear el bucle durante demasiado tiempo. Cuánto tiempo es “demasiado...”, dependerá de lo que se espere del sistema en cada momento concreto de su uso.

nota: En algunos sistemas o aplicaciones, se suele dotar al bucle central de suficientes mecanismos de supervisión para que mantenga siempre el control último; habilitándolo para limitar o abortar cualquier función que tarde “más de la cuenta” en ejecutarse. Pero el uso de estos mecanismos debería ser siempre el último recurso; lo más importante es diseñar de forma adecuada las funciones que se utilizan para que se garanticen los tiempos de respuesta requeridos.

nota: En los últimos años están teniendo mucho auge sistemas o aplicaciones distribuidos, basados en multitud de partes o servicios que colaboran entre sí (ver más adelante, en la sección 14.4). En este tipo de sistemas, también se suele hablar de ‘orientación a eventos’ cuando las diversas partes se ejecutan de forma asíncrona y desacoplada. Cada parte va notificando a las demás de lo que hace, a base de publicar ‘eventos’ cuando realiza ciertas acciones; ‘eventos’ que las otras partes interesadas recogen y procesan (o no) cuando les conviene.

14.2. Orientación a objetos (object-oriented)

Más que una arquitectura, es un paradigma de programación, una forma de escribir código. Se podría decir que es la evolución de la programación estructurada (funciones y módulos).

En la programación orientada a objeto, cada módulo (clase) contiene todo lo que necesita para realizar una tarea o misión concreta:

- tanto los datos (propiedades),
- como el comportamiento (métodos).

Y se garantiza que ningún objeto pueda interferir dentro de otro, si no es a través de los caminos expresamente habilitados para ello (propiedades o métodos declarados como ‘public’).

La principal ventaja de este paradigma es hacer patentes las dependencias entre los diversos elementos dentro de la aplicación. Una vez son patentes esas dependencias, es más sencillo minimizarlas, reduciéndolas a las mínimas imprescindibles. Además, cuanto menos dependencias haya entre elementos, resulta más fácil ver a qué puede afectar cualquier cambio que se haga en un determinado elemento.

Mediante un adecuado reparto de responsabilidades se puede llegar a una estructura donde cada objeto se dedica a hacer algo concreto y donde cada tarea concreta solo

es realizada por un tipo concreto de objeto.

Aviso:

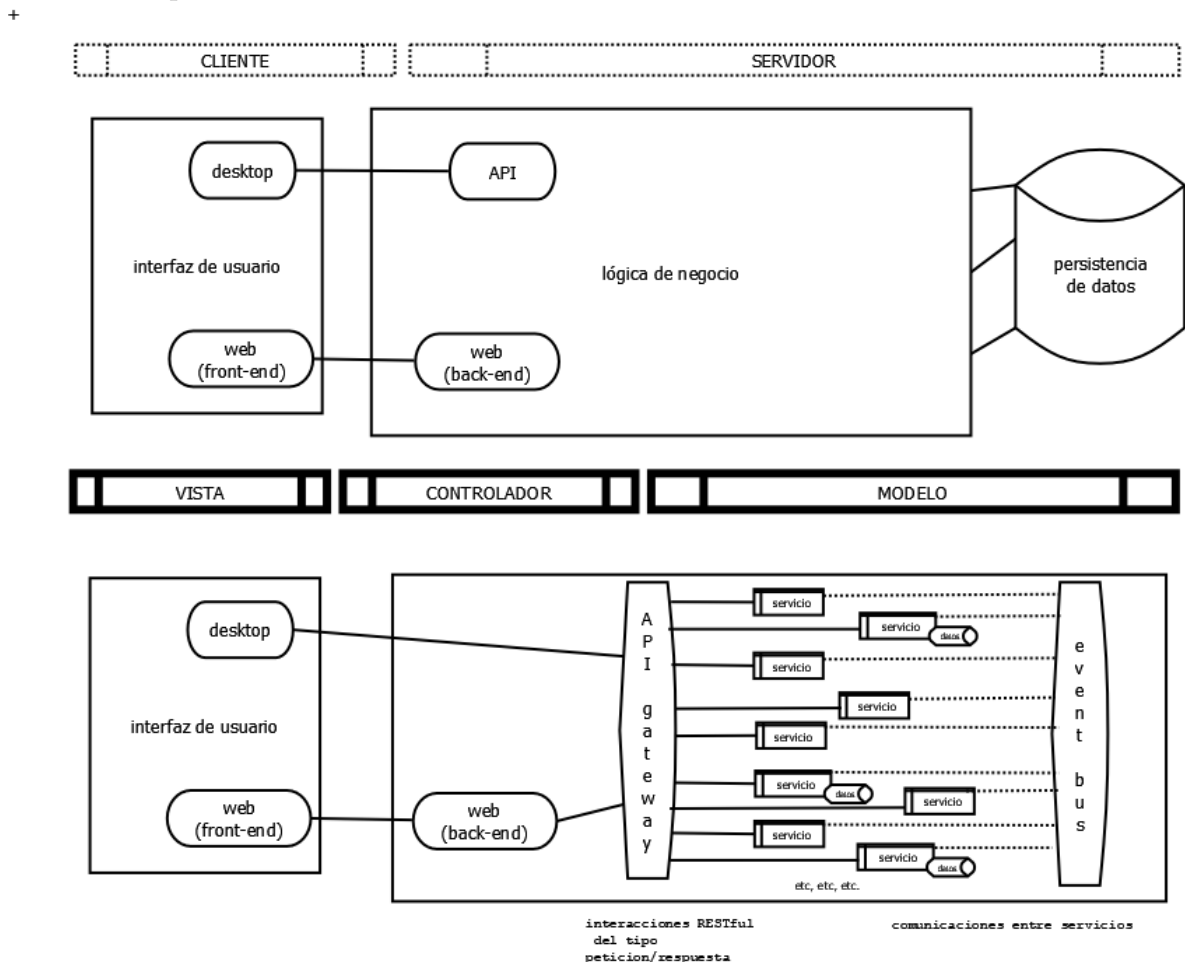
A no ser que se tenga mucha experiencia previa en aplicaciones similares, es imposible acertar a la primera.

Un adecuado reparto de funcionalidades (responsabilidades) entre diversos objetos se va obteniendo a base de ir moviendo datos (propiedades) o funciones (métodos) de un objeto a otro. Según se va viendo más lógico o más claro que estén en uno o en otro objeto, se van ajustando estos.

14.3. Arquitecturas multi-capa (n-tier)

Esta arquitectura se basa en separar la aplicación en diversas capas especializadas. De tal forma que se minimizan las dependencias entre ellas, limitándolas a los puntos de interacción (interfaces) entre capas.

Una muestra de algunas de las distintas posibilidades para estructurar un programa en varias capas:



El principal objetivo en cualquiera de estas arquitecturas multi-capa es desacoplar las capas entre sí.

Por ejemplo, separando:

- la forma de trabajar (medios de manipulación: ratón, teclado, gestos, voz,...) y de presentar información (gráfica, textual, auditiva,...),
- del trabajo a realizar (información manejada y operaciones ejecutadas)
- de los datos a manipular (datos creados/leídos/modificados/borrados a través de esas operaciones) (nota: CRUD Create/Read/Update/Delete).

En una arquitectura multi-capa bien ejecutada, las capas tienen pocas dependencias (bajo acoplamiento) entre ellas. Y las dependencias están definidas de forma bien clara ('interfaces' concretos); con la intención de que las relaciones entre capas sean lo más estables posible.

Se facilita así la evolución del programa al permitir modificar capas concretas sabiendo de antemano cómo afectarán esos cambios a otras capas adyacentes.

De forma ideal, una arquitectura multi-capa bien ejecutada permite sustituir elementos en una de las capas sin afectar para nada a lo que hay en las otras capas, (o con mínimos cambios en alguna capa adyacente).

Por ejemplo, debería ser posible sustituir sin demasiados problemas un interfaz gráfico basado en ventanas/ratón/teclado, por otro interfaz textual basado en comandos/parámetros, o por otro interfaz basado en servicios web REST. O por ejemplo, debería ser posible sustituir una persistencia basada en archivos XML, por otra persistencia basada en archivos JSON, o por otra persistencia basada en una base de datos. O...

nota: No es que se vayan a realizar ninguna de esas sustituciones a lo largo de la vida útil del programa. El objetivo no es ese. Se trata de pensar en que se podrían realizar, para así ayudar a detectar dependencias y evitarlas.

Cuanto menos dependencias existan, más fácil será comprender, mantener y evolucionar las distintas partes del programa.

Una organización muy típica consiste en organizar el código en tres capas:

- **MODELO**: la parte que representa el dominio de la aplicación y que engloba la gestión de datos y la lógica de trabajo.
- **VISTA**: la forma en que el modelo se muestra a los usuarios.
- **CONTROLADOR**: la forma en que los usuarios interactúan con el modelo a través de la vista.

El orden de las letras en el acrónimo MVC es importante. Se obtienen mejores resultados si primero se construye el modelo, luego la vista y, por último, el controlador. Aunque sin olvidar que todo proceso de construcción de software es iterativo y, por tanto, se irán realizando ajustes en el modelo, en la vista o en el controlador según se avanza en el trabajo en cualquiera de esas partes.

notas:

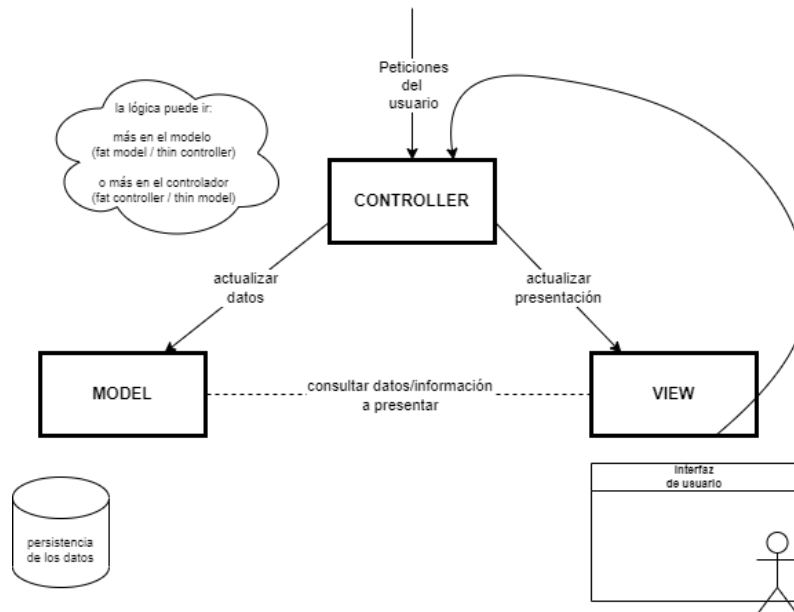
Mientras se construye el modelo, para ejercitarlo y ver que funciona bien, se pueden utilizar test

unitarios. No es necesario tener un interfaz de usuario (vista y controlador) para probarlo.

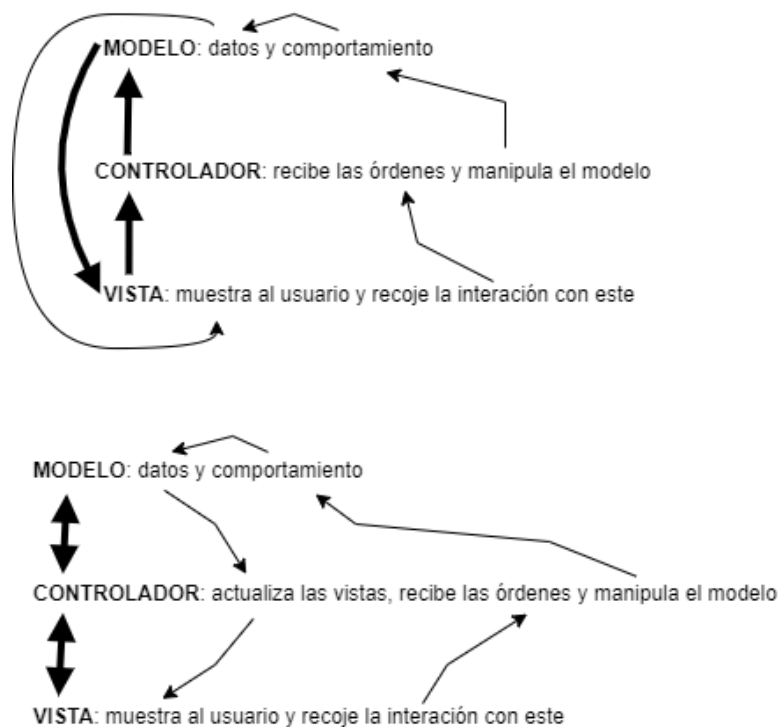
El modelo ha de recoger tanto la información (datos manejados) como el comportamiento (operaciones realizadas sobre esos datos). Un modelo que contemple solo datos se suele denominar “modelo anémico”.

El controlador y la vista tienden a tener más acoplamiento entre sí. Es normal que sea así.

“Usuarios” se ha de entender en sentido amplio, pudiendo ser tanto personas, como otros programas, como...; es decir, es aquello que interactúa con la aplicación desde “el exterior” de esta.



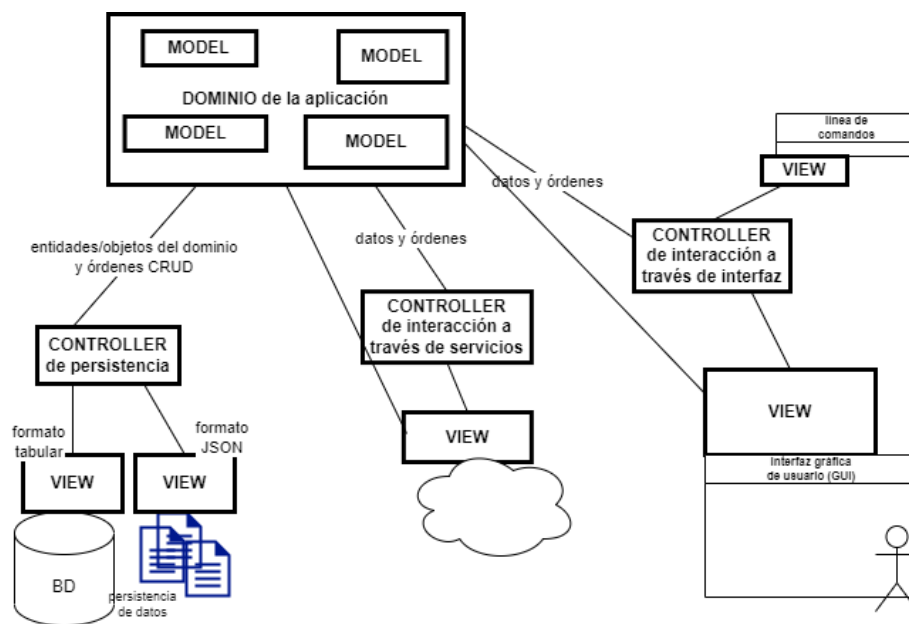
MVC puede tener varias interpretaciones, no todo el mundo lo interpreta igual:



El patrón de arquitectura en tres capas también se conoce bajo otras denominaciones además de MVC (Model-View-Controller): MVP (Model-View-Presenter), MVVM (Model-View-ViewModel), PMVC (Persistence-Model-View-Controller), etc.

Además, la idea puede ser extendida y ser aplicada en un sentido más amplio:

- “MODELOS”: gestionan la información y los algoritmos de la aplicación.
- “CONTROLADORES”: hacen de adaptadores entre las vistas y los modelos.
- “VISTAS”: gestionan las peculiaridades tecnológicas de la interacción con el entorno exterior de la aplicación.



Pasando a otro tema. Una metodología muy práctica para guiar la organización del código en capas, sobre todo en softwares complejos, es la metodología **DDD** (Domain-Driven Design).

Un dominio es el contexto, entorno o disciplina donde se sitúa un determinado trabajo que se desea realizar.

En la metodología DDD, se separan con claridad:

- Los aspectos intrínsecos funcionales relacionados con cada uno de los dominios del problema a resolver.
- Los aspectos tecnológicos incidentales relativos a la implementación de la solución con las herramientas concretas que se estén usando para ello.

Así, el programa resultante puede evolucionar de forma flexible. Según lo demande la parte de funcionalidades (dominio) o la parte tecnológica (infraestructura).

Pero DDD va mucho más allá de la separación en capas o en dominios. Contempla múltiples aspectos acerca de cómo encarar el diseño de un sistema software.

(Eric Evans , <https://www.domainlanguage.com/ddd/>)

Comenzando por el concepto troncal de ‘ubiquitous language’: todas las personas que participan en una determinada parte (‘bounded context’) del sistema han de hablar un lenguaje común; y ese lenguaje común consensuado se ha de ver reflejado también en el propio código, en las denominaciones de variables, funciones, clases,...

La metodología DDD distingue diversos tipos de relaciones entre los diversos dominios que cohabitan dentro de un sistema.

Según cómo un dominio utilice o se sirva de otro, podemos tener:

- Colaboración (Partnership): ambas partes evolucionan de forma conjunta; teniendo en cuenta las necesidades de una y de otra.
- Cliente-Servidor (Customer-Supplier): una parte proporciona servicios a la otra; y, en consecuencia, evoluciona teniendo en cuenta esa dependencia.
- Conformismo (Conformist): una parte utiliza (tal cual) servicios de la otra; y, en consecuencia, se va adaptando (como pueda) a la evolución de esa otra.
- Aislamiento (Anticorruption Layer): una parte utiliza servicios de la otra, pero lo hace a través de una capa de adaptación intermedia (proxy/adapter/facade); esto le proporciona una cierta independencia frente a la evolución de esa otra.

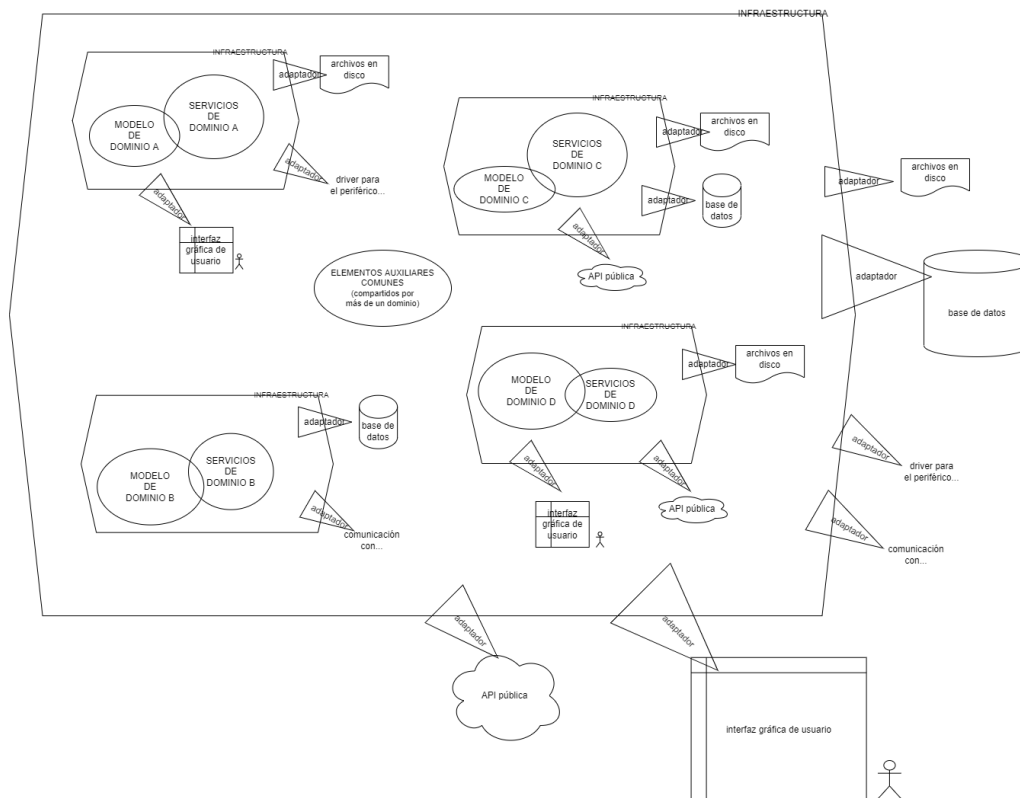
Según el grado de dependencia entre dominios, podemos tener:

- Independencia (Separate Ways): cada parte hace su trabajo tal cual, sin ninguna dependencia entre ellas; ambas pueden evolucionar cada una por su lado.
- Coordinación (Open Host Service): las partes adoptan un protocolo de comunicación para coordinarse; mientras ambas respeten ese protocolo en sus interacciones, ambas pueden evolucionar cada una por su lado.
- Compartición (Shared Kernel): las partes comparten una porción de modelo que sirve a ambas; cada una ha de mantener alineados sus respectivos conceptos internos relacionados con ese modelo común; pero, por lo demás, pueden evolucionar cada una por su lado.
- Monolito (Big Ball of Mud): no hay límites claros entre distintas responsabilidades y distintos dominios; las partes hacen un amplio uso directo unas de otras, según convenga; el sistema evoluciona “como se pueda”...

<http://laputan.org/mud/mud.html#Abstract>

Las capas en un esquema DDD podrían encuadrarse en uno de estos tipos:

- **MODELO De Dominio : (DOMAIN MODEL) :** Recoge los “entes” propios del dominio. Recoge la “lógica de negocio” a nivel de cada uno de esos “entes” (cada uno contempla tanto la información que define su estado interno como las operaciones que admite para modificar dicho estado).
- **SERVICIOS De Dominio : (DOMAIN SERVICES) :** Recoge la “lógica de negocio” a nivel de las operaciones que involucran a más de un “ente” del dominio o a una mezcla de “entes” y elementos externos al dominio.
- **ELEMENTOS AUXILIARES Comunes : (SHARED KERNEL) :** Recoge aquellas partes de código comunes compartidas por varios dominios.
- **INFRAESTRUCTURA : (APPLICATION)(EXTERNAL WORLD) :** Recoge los “adaptadores” empleados para que las capas anteriores utilicen recursos externos (bases de datos, archivos, comunicaciones, periféricos, ...) o para que sean utilizadas desde el mundo exterior (interfaces de usuario, API pública, ...)



Un ejemplo práctico de algunos de los conceptos citados puede verse en

<https://bitbucket.org/susosise/hotel/src/main/> Aviso: está en construcción (y, por cierto, aún muy verde, solo empezando), así es que te animo a clonarlo y mirar de vez en cuando como va avanzando.

Para complementar esta sección, puede resultar interesante informarse acerca de los **principios SOLID**. Por ejemplo, leyendo la correspondiente sección en el documento https://www.susosise.es/documentos/Proyecto_Software.pdf.

14.4. Aplicaciones distribuidas

Diversas aplicaciones (o procesos) corriendo de forma independiente en una misma máquina o en diversas máquinas. Cada aplicación realiza su trabajo, pero en algunos momentos colaboran entre ellas.

Para comunicarse entre sí, las diversas aplicaciones y servicios suelen seguir alguno de estos estilos de trabajo:

- Llamadas directas a procedimientos: (RCP, Remote Procedure Call). La aplicación que usa los servicios llama a estos a través de un trozo de código local (“proxy”) (“stub”) que tiene linkado en ella. Ese trozo local sabe cómo serializar/deserializar los datos (parámetros y respuestas) y sabe cómo llamar a las funciones en el otro trozo de código remoto (el servicio). El trozo remoto puede estar corriendo en la misma máquina o en otra cualquiera.
nota: cuando todas las llamadas son dentro de una misma máquina, se puede usar IPC (Inter-Process Communication), que es más directo; pero menos escalable (limitado a una sola máquina).
- Llamadas asíncronas a recursos o servicios identificados con un URI (Universal Resource Identifier). La aplicación que usa los servicios llama a estos usando el URI correspondiente, normalmente usando el protocolo HTTP (<https://tools.ietf.org/html/rfc7231>). El servicio (recurso) identificado con esa URI le devuelve unos datos resultado, o le devuelve un archivo, o ejecuta una determinada acción concreta, o...
- Colas o tableros, de mensajes o de eventos: (MQ, Message Queue; EB, Event Bus o Event Broker; ...) La aplicación que usa los servicios deposita/envía mensajes a una cola. El servicio correspondiente recoge/recibe mensajes desde la cola, actúa en consecuencia y deposita/envía (a la misma cola o a otra) los correspondientes mensajes respuesta.
nota: Cuando la información intercambiada es más o menos sencilla o “pasiva” (solo representan estados, sucesos o envíos de información), suelen denominarse eventos. Cuando esa información es más o menos compleja o “activa” (pueden representar órdenes, peticiones de servicio,...; pueden requerir respuestas específicas; ...) suelen denominarse mensajes.
nota: Cuando es irrelevante el orden en que la información se deposita y en el que se lee, se suele denominar tablón. Cuando dicho orden es importante, se suele denominar cola.
nota: Las colas o tableros se suelen denominar también ‘temas’ (topics) y cada uno suele tener un propósito concreto y determinado.
La complejidad del sistema puede ir desde un simple publicar/subscribir, donde una parte deposita información y otras partes la leen. Hasta complejos protocolos de intercambios de mensajes/eventos para conseguir que dos o más procesos colaboren activamente entre sí, de forma asíncrona, para lograr llevar a cabo una tarea.

aviso: Una vez se comienza a trabajar en uno de esos estilos es muy difícil cambiar de estilo. Cuando se pone en marcha un proyecto grande basado en servicios, hay que

sopesar cuidadosamente el balance entre los requisitos de sincronicidad, de tiempos de respuesta, de integridad transaccional y de escalabilidad. Para, en función de ello, decidir el estilo a emplear .

Algunas referencias ilustrativas:

<https://www.mpich.org/>

<https://computing.llnl.gov/tutorials/mpi/>

<https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report/mpi31-report.htm#Node0>

14.5. Orientación a servicios (SOA, SaaS, microservices,...)

Se trata de encapsular cada una de las funcionalidades ofrecidas por el software, construyendo un servicio concreto para cada una de ellas.

Un servicio se caracteriza porque recibe cierta información al ser llamado (parámetros de petición) y devuelve cierta información (una respuesta estructurada en un determinado formato). Es todo lo que se necesita utilizarlo.

Una aplicación se construye a base de realizar llamadas a los servicios disponibles.

En los sistemas más sofisticados, suele haber algún mecanismo (un catálogo) para que las aplicaciones puedan “consultar” los servicios disponibles en el sistema y “aprender” cómo se utiliza cada uno.

14.6. Orientación a servicios, arquitectura REST (REpresentational State Transfer)

Requisitos que ha de cumplir un sistema para ser considerado REST:

- Es cliente-servidor: la parte cliente se encarga de interactuar con el usuario y la parte servidor se encarga de manipular la información (los recursos).
- No se guardan estados internos: cada llamada a un servicio es independiente, no depende de las llamadas previas a ese o a otros servicios.
- Permite ser cacheado: cada porción de información (cada recurso) establece con claridad si un cliente o un intermediario puede guardarla para reutilizarla cuantas veces la necesite sin volver a solicitarla al servidor; o si, por contra, es importante re-adquirirla cada vez que se necesite.
- Permite ser utilizado a través de varios niveles: las llamadas a los servicios se han de poder completar igualmente aunque haya intermediarios entre el

cliente y el servidor final que provee los servicios. Esto es, posibles intermediarios (por ejemplo balanceadores de carga o sistemas de seguridad) han de ser transparentes para los clientes que utilizan el sistema.

- Permite ser personalizado bajo demanda: el cliente puede recibir código desde el servidor para ampliar o personalizar alguna funcionalidad del sistema.
- Tiene un interfaz uniforme para todos los servicios involucrados:
 - Los recursos afectados en cada llamada se identifican con claridad y sin ambigüedades.
(por ejemplo, utilizando URIs o UUIDs)
 - Los recursos se manipulan a través de una representación de los mismos. Dicha representación contiene todo lo que un cliente necesita para hacer las llamadas adecuadas para consultarlos, crearlos, modificarlos o eliminarlos.
(por ejemplo llamadas GET, POST, PUT, PATCH, DELETE,...)
 - La información intercambiada en los mensajes es autodescriptiva. Cada mensaje contiene lo necesario para que su receptor sepa cómo ha de tratarla.
(por ejemplo, utilizando identificadores MIME y formatos fácilmente reconocibles tales como HTML, XML, JSON,...)
 - La navegación por las diversas etapas o estados de la aplicación se realiza a través de hipervínculos. Cada paso guía al cliente hacia los siguientes pasos disponibles a partir de ahí.
(HATEOAS, Hypermedia as the Engine of Application State)

Un enlace ilustrativo:

<https://martinfowler.com/articles/richardsonMaturityModel.html>

En cierta manera, se podría decir que para utilizar una buena API ¹ REST no se necesita consultar instrucciones prefijadas de antemano. Si se conoce el punto de inicio (la primera URL a la que llamar); a partir de ahí se continua a base de ir leyendo e interpretando ² la información intercambiada en cada paso.

nota: Habitualmente, las APIs REST se suelen implementar usando HTTP como protocolo de transporte/interacción y JSON como formato de estructuración de la información. Pero podrían ser implementadas de otras formas, utilizando cualquier otra combinación de estándares en el campo de aplicación y de texto legible por personas humanas.

¹API: exponer la lógica de un software (servidor) para poder ser consumida o utilizada por otros softwares (clientes)

²Ese ‘leer e interpretar’, obviamente, implica que se conoce el idioma en el que está escrita la información y que se conoce la jerga específica del dominio de aplicación que se esté tratando.

Capítulo 15

Ejercicios

((una disculpa del autor: este capítulo no lo he abordado aún; el libro esta “en construcción...” y sigo trabajando en él; queda mucho trabajo para completarlo.))

(nota: Ver posibles soluciones al final del libro, en el capítulo 17.)

comentario: Esta va a ser la parte más complicada de ejercitar. La idea es adquirir práctica real en proyectos de una cierta (gran) envergadura. Resolviendo cada uno de ellos con distintas arquitecturas, y viendo los pros/contras de cada una de esas arquitecturas.

Pero como eso va a resultar complicado, empezaré por ejercicios sencillos para practicar el desacople entre capas.

15.1. Multiinterface

Preparar un pequeño programilla sencillo. Por ejemplo:

- Uno que, dados un lado y otro lado de un paralelogramo, calcule y devuelva su perímetro y su área.
- Uno que permita pedir una bebida de una lista (café, leche o chocolate) e introducir un código ID de usuario (3 letras). El usuario ha de poder consultar la lista de bebidas disponibles. El programilla ha de avisar en caso de pedir una bebida agotada (usar un contador de existencias en los depósitos; con la correspondiente función de cargar un depósito). Y ha de avisar también en caso de recibir un código incorrecto (usar una lista de usuarios; con la correspondiente función de dar alta/baja de códigos ID de usuario). Los contadores y la lista han de persistir en un archivo de texto.

Utilizar el programilla (tal cual, sin modificar su núcleo para nada) desde diversos interfaces de usuario. Por ejemplo mediante:

- Unos comandos tecleados desde la línea de comandos.

- Un formulario gráfico (aplicación “desktop”).
- Unos servicios web “RESTfull”, (una API web).
- Una página web que hace uso de esos servicios.

Persistir los datos en diversos formatos. Por ejemplo: texto plano, JSON, XML, . . . (Sin modificar el núcleo del programa para nada.)

15.2. Arquitectura en tres capas, MVC

Preparar un pequeño programilla aplicando una estructura multicapa MVC con:

- Un modelo: una lista de personas, con unos pocos datos (nombre, apellidos, . . .) en cada entidad persona y un sistema de persistencia (archivo de texto o base de datos) donde guardar la lista.
- Una vista: un interfaz gráfico (GUI) a través del cual interactuar con la lista de personas. Por ejemplo, consultarlas, dar de alta una nueva, borrar una, . . .
- Un controlador: código que se encargará de recoger las instrucciones dadas a través del interfaz para trasladarselas al modelo; así como de leer datos del modelo, elaborarlos y mostrarlos en el interfaz según corresponda.

Hacerlo de dos formas distintas,
aplicando las dependencias entre sus distintas partes (objetos):

- (1) de forma directa: Cada objeto se instancia él mismo los objetos que necesite.
- (2) de forma inversa: Cada objeto recibe desde fuera los objetos que necesite (o bien inyectados a través de su constructor o bien inyectados a través un metodo setter ad-hoc).

ampliación: Siguiendo otra de las interpretaciones bastante aceptadas de MVC (de hecho, la interpretación original). Elaborar otra versión del programa donde la vista “observa” o “escucha” de forma directa el modelo para conseguir los datos que ha de mostrar. nota: Esta forma de proceder es muy útil en los casos en que el flujo de datos modelo→controlador→vista es grande, en aquellos en los que el controlador no haga mas que de mero repetidor en ese flujo de datos y/o en aquellos donde múltiples vistas han de refrescarse simultáneamente para reflejar modificaciones en los datos realizadas desde una cualquiera de ellas.

15.3. Excursiones

Se trata de realizar un programa que permita a alguien (a una persona encargada dentro del club o asociación):

- Registrar excursiones. Con cuatro datos básicos: fecha, hora, destino, coste, poblacionOrigen, poblacionDestino, tipoDeExcursion.
nota: los tipos de excursión se clasificarán según la actividad principal que se

vaya a hacer: `VisitaHistorica`, `ConciertoMusical`, `ActuacionTeatral`, `RutaNatural`,...

- Registrar personas interesadas en participar en alguna excursión. Con cuatro datos básicos: nombre, apellidos, poblacion, email, telefono, preferenciasDeTipoDeExcursion.
- Saber qué excursiones están activas (aún sin realizar).
- Apuntar participantes a una excursión. Y llevar el control de quien ya la ha pagado quien todavia no.
- Cuando se planifica una nueva excursión, enviar avisos (por email) a todas las personas de la poblacionOrigen que pudieran estar interesadas en ese tipo de excursión.

ampliación: Hacer el programa con interface web.

ampliación: Y, ya que estamos en web, permitir a las posibles personas participantes consultar las excursiones activas y registrarse/apuntarse ellas mismas a las excursiones que deseen.

15.4. Hotel

nota: Este ejercicio intenta ser un proyecto de una cierta (gran) envergadura. Se puede complementar con todo aquello que se nos ocurra. Lo que se cita aquí son algunas ideas de por dónde comenzar a trabajar.

Siguiendo la recomendación de DDD (Domain Driven Development) acerca del “ubiquitous language”: todas las personas que participan en una determinada parte del sistema (“bounded context”) han de hablar un lenguaje común.

Estas son las principales entidades que vamos a manejar:

Habitaciones: Se identifican por un ‘número de habitación’ (un ‘nombre de habitación’ que puede ser alfanumérico).

Su tipo de habitación puede ser, por ejemplo: `SENCILLA`, `DOBLE`, `SUITE`,...

Su baño puede ser, por ejemplo: `COMPARTIDO`, `DUCHA`, `BAÑERA`,...

Pueden tener diferentes tamaños de cama, por ejemplo: 90, 135, 180,...

Pueden tener diferentes orientaciones de vista desde su ventana, por ejemplo: `FRONTAL`, `POSTERIOR`, `DERECHA` o `IZQUIERDA` (considerando el hotel como un “cuadrado rectangular”, mirando desde dentro hacia la entrada principal, el frontal es la fachada donde está la entrada y el posterior queda a nuestra espalda)

Huespedes: las personas que se alojan en el hotel.

Habrá que recoger al menos los datos básicos:

- id legal (por ejemplo, número de dni, de pasaporte, de cédula,...)
- nombre y apellidos

- forma de pago (por ejemplo, contado, tarjeta, facturación,...) y los datos necesarios para realizarla (por ejemplo, para la tarjeta seria su número, nombre titular, fecha caducidad y número clave). (nota: Si un huésped no es el principal de una estancia de un grupo de personas, podemos saltarnos la forma de pago en su registro).

Para simplificar, en un principio no se contemplarán los requisitos derivados de las leyes de protección de datos de información de carácter personal. En una segunda etapa del ejercicio, se podría implementar este tema; para seguir profundizando en los aspectos de evolucionabilidad del software. (Obviamente, si esto fuera una aplicación real, el tema de la protección de datos seria una de las primeras cosas a tener en cuenta.)

Otro posible aspecto para una segunda etapa, podría ser el de la "fidelización" de huéspedes: tarjeta de cliente, tarjeta VIP, puntos, promociones, etc.

Estancias: Una estancia no es más que una asignación de un cierto huésped a una habitación (si es una persona individual) o de unos ciertos huéspedes a una/s habitación/es (si son un grupo de personas)

Tiene una fecha de entrada y una fecha de salida. Ambas válidas desde/hasta el mediodía. (Es decir, la estancia está activa desde las 12:00 de la fecha de entrada hasta las 12:00 de la fecha de salida.) Estancia activa == ¿habitaciones ocupadas.

A una estancia se le pueden ir cargando los servicios que vayan consumiendo sus huéspedes. Normalmente estos cargos se harán contra un número de habitación.

Al terminar la estancia, hay que liquidar y crear una factura.

Servicios: Todo aquello que se cobra al cliente.

Reservas: Son como estancias, pero en el futuro. La diferencia es que no tienen un número de habitación exacto asignado aún (pero si un tipo de habitación concreto y unas posibles condiciones especiales).

Se han de tener en cuenta al consultar disponibilidades en unas determinadas fechas. A efectos prácticos cuentan como habitaciones ocupadas, igual que las estancias que estén activas en esas fechas.

Empleados: Las personas que trabajan en el hotel

Roles: de los empleados. Por ejemplo:

- Limpieza y servicio de habitacionesAssign/Schedule...
- RecepciónAssign/Schedule...
- Cafetería y restauraciónAssign/Schedule...
- Mantenimiento y reparacionesAssign/Schedule...
- Administración

Avisos: Cualquier información que se deba hacer llegar a alguien y que se le pueda hacer llegar de forma asíncrona.

Aquí van algunas fichas con casos de uso a implementar y tareas a realizar. Las podemos recortar y poner en el tablón de gestión del proyecto si es que llevamos alguno. (nota: No están en orden, cada cual tendrá que reorganizarlas según las prioridades que estime oportuno.)

nota: Es importante es abordar una ficha cada vez; y no pasar a otra ficha hasta la ficha en curso esté totalmente terminada del todo, todo.

<p>Altas/bajas/modificaciones de habitaciones.</p> <ul style="list-style-type: none"> ▪ modelar la entidad Habitación ▪ crear una nueva y guardarla ▪ evitar duplicados ▪ buscar y recuperar una, o varias, o todas ▪ editar datos de una existente ▪ dar una de baja (inactivarla) 	<p>Infraestructura de configuración.</p> <p>Toda aplicación tiene algunos parámetros que conviene poder modificar "desde fuera", para aplicarlos cuando la aplicación arranca.</p> <p>nota: lo más típico es algún archivo INI, JSON o YAML, o algunas variables de entorno, o algunos parámetros en el ejecutable/comando que arranca la aplicación.</p>
<p>Liquidar una estancia.</p> <p>Generando factura detallada de todo lo consumido.</p>	<p>Cargar un servicio a una estancia.</p> <p>Normalmente, se identificará la estancia a través de un número de habitación.</p>

<p>Infraestructura de persistencia.</p> <p>El mecanismo de persistencia principal que se va a utilizar para almacenar los datos manejados.</p> <p>Pensar y preparar cómo se va a utilizar a lo largo y ancho de toda la aplicación.</p> <p>notas: Una vez decidida como implementar la persistencia, normalmente esta no se suele cambiar. Pero en este ejercicio conviene pensar en implementar más de un mecanismo de persistencia para unos mismos objetos. Por lo menos en alguna de las partes de la aplicación. Para así hacer experimentos acerca del grado de dependencia que tiene el código respecto de la persistencia utilizada.</p> <p>Por ejemplo, se podría preparar una persistencia basada en base de datos (en MariaDB, o en PostgreSQL, o en SQLite, o en...), y otra persistencia basada en archivos (en JSON, o en XML, o en CSV, o en...). Para probar cuán sencilla (o complicada) es la tarea de cambiar de utilizar una a utilizar la otra según la arquitectura interna de nuestro programa.</p>	<p>Infraestructura de interface.</p> <p>preparar la pantalla inicial y los "menús" de acceso a las distintas partes de la aplicación.</p> <p>notas: En este ejercicio conviene pensar en implementar más de un interface. Por lo menos en alguna de las partes de la aplicación. Para así hacer experimentos acerca del grado de dependencia existente entre la lógica de aplicación y el interface.</p> <p>Por ejemplo, se podría preparar un clásico interface gráfico (GUI) y otro interface vía servicios (por ejemplo, una API HTTP/REST). En teoría, se deberían poder usar ambos indistintamente.</p>
<p>Una persona consulta sus avisos.</p> <p>Tanto los dirigidos a ella personalmente, como los dirigidos a los roles que desempeña.</p>	<p>Registrar un aviso para un empleado o para un rol.</p>

<p>Registrar la estancia de un/os huésped/es en un/as habitación/es.</p> <p>En caso de que varios huéspedes vengan como grupo y se les vaya a facturar como tal grupo,</p> <ul style="list-style-type: none"> ■ Los datos generales (los de facturación) serán los de una de las personas integrantes, o del propio grupo (si este tiene entidad jurídica). ■ Pero se han de recoger también las identidades de cada una de las personas del grupo. <p>No se puede registrar una estancia sin las fechas de entrada y salida correctas. Entrada hoy o anterior. Salida posterior a la entrada.</p> <p>No se puede registrar una nueva estancia para una habitación que ya esté ocupada por una estancia activa.</p>	<p>Revisar/modificar los precios de las habitaciones.</p> <p>Obviamente, si el nuevo precio de alguna habitación se incrementa, no se puede aplicar a la estancia que estuviera ya registrada en esa habitación. Pero si que se debe aplicar si el nuevo precio se ha reducido.</p>
<p>Altas/bajas/modificaciones de servicios.</p> <p>Es necesario tener una lista de los servicios disponibles, junto con el precio de cada.</p> <p>Como va a haber muchos, será necesario prever un sistema de clasificación para ayudar a buscar un servicio concreto. Usando dos niveles: grupo y servicio; o tres niveles: grupo, subgrupo y servicio.</p> <p>Los grupos podrían ser, por ejemplo: lavandería, minibar, bar/restaurante, transporte,...</p>	<p>Revisar (modificar) los precios de los servicios.</p> <p>Lo ideal sería poder presentarlos en un interface tipo "hoja de cálculo" para facilitar la revisión.</p> <p>Sería interesante también algún mecanismo para incrementar o decrementar un porcentaje concreto a todo un subgrupo o a todo un grupo de servicios.</p>

	<p>Infraestructura de log.</p> <p>Para registrar lo que va pasando dentro de la aplicación.</p> <p>https://en.wikipedia.org/wiki/Logging_(software)</p>
<p>Altas/bajas/modificaciones de huéspedes.</p> <p>Tener en cuenta que algunos huéspedes (aquellos registrados como ‘huésped principal’ de una estancia) han de tener datos de facturación y cobro. Dichos huéspedes pueden ser personas físicas o jurídicas.</p> <p>En un principio, no se contemplarán temas sobre el tratamiento de datos de carácter personal. Pero si esto fuera una aplicación real, habría que tener en cuenta la normativa.</p>	<p>Altas/bajas/modificaciones de reservas.</p> <p>A todos los efectos, una reserva es una habitación ocupada. Es como una estancia, solo que no ocupa una habitación concreta, sino que ocupa una de un tipo determinado.</p> <p>Se ha de tener en cuenta a la hora de consultar qué habitaciones hay libres cara a una nueva estancia o una nueva reserva.</p>
<p>Gestión de tarjetas-llave.</p> <p>Se ha de llevar un registro de qué llaves están autorizadas para qué habitación.</p> <p>Se ha de poder autorizar una llave para que abra una determinada habitación. Y también poder revocar esa autorización.</p>	<p>Altas/bajas/modificaciones de roles.</p> <p>Ha de ir junto con un mecanismo de asignación de roles a empleados.</p> <p>nota: Una misma persona puede tener más de un rol.</p>
<p>Altas/bajas/modificaciones de empleados.</p>	<p>Asignar roles a empleados.</p> <p>Tener en cuenta que se podrán asignar, o bien roles a un empleado concreto, o bien empleados a un rol concreto.</p>

<p>Infraestructura de autorizaciones y permisos.</p> <p>Una vez tengamos usuarios y tengamos roles. Podemos usarlos también para seguridad interna en la aplicación:</p> <ul style="list-style-type: none"> ■ Hay que preparar un mecanismo de inicio de sesión (login) seguro. ■ Hay que asegurar que cada cual solo puede acceder a aquello que deba acceder. <p>Obviamente, esto pasa por tener también alguna manera de editar y mantener esa información de "qué" puede hacer "quién". Ese "quién" se definirá asignando a roles y no a usuarios individuales.</p> <p>(Tener en cuenta que un mismo usuario puede tener asignados varios roles si es necesario.)</p>	<p>Tarjetas-llave maestras . Auditoria de accesos.</p> <p>Los empleados necesitan tarjetas capaces de abrir cualquier habitación. El uso de estas tarjetas maestras ha de quedar auditado. Y, ya que estamos, se puede también auditar el uso de cualquier tarjeta en cualquier habitación.</p>
<p>Inicio de sesión de un empleado.</p> <p>Cada empleado puede gestionarse la contraseña que desee.</p> <p>nota: Un hotel no es "Fort Knox", será suficiente con guardar y manejar hash-es de las contraseñas.</p>	<p>Infraestructura de observabilidad.</p> <p>Una aplicación instrumentada y que reporta estadísticas (telemetría) sobre su funcionamiento interno, es una aplicación que puede monitorizarse continuamente para ver si está funcionando bien (como siempre) o si tiene problemas (hace cosas raras").</p> <p>Una vez instrumentada y reportando en algún formato estándar. La observación se puede hacer mediante una plataforma de agregación de estadísticas que incluya un cuadro de mando y un sistema de alertas. (algo tipo https://grafana.com/ o https://www.nagios.org/ o algo similar)</p>

Marcar una habitación como no utilizable.

Se ha de recoger:

- por qué está fuera de servicio
- tiempo estimado (horas, días, meses,...) que va a permanecer así

Capítulo 16

apéndice A: ejemplos de posibles herramientas de trabajo

Este es un compendio de mis herramientas favoritas a día de hoy (Julio 2020). Que quede claro que hay muchas otras disponibles y que cada cual es libre de utilizar las que prefiera.

nota: Todas las herramientas citadas, o bien son directamente software libre o bien tienen una versión de uso libre (Community Edition).

16.1. Para programar en Java

Un compilador-debugger ::: JDK (Java Development Kit), por ejemplo:

<https://adoptium.net/es/>

(o OpenJDK: <https://jdk.java.net>)

(o el JDK comercial de Oracle: <https://www.oracle.com/java/technologies/javase-downloads.html>)

Un IDE ::: Eclipse: <https://www.eclipse.org>

(o Netbeans: <https://netbeans.apache.org>)

(o IntelliJ IDEA: <https://www.jetbrains.com/es-es/idea>)

etc.

Un entorno de pruebas unitarias ::: JUnit: <https://junit.org>

Un entorno de registro de eventos ::: JLog: <https://logging.apache.org/log4j>

Una biblioteca GUI ::: incluidas en la plataforma: *java.awt*, *java.swing* (ambas ya “deprecated”). (Nota: para facilitar los primeros pasos con AWT y Swing puede ser útil un complemento tal como WindowBuilder: <https://www.eclipse.org/windowbuilder>.)

Otra biblioteca también incluida en la plataforma: *JavaFX* (la que se recomienda usar hoy en día). Es algo más compleja, pero permite implementar interfaces de usuario mucho más ricos. (Nota: para facilitar los primeros pasos, puede ser útil un complemento tal como JavaFX Scene Builder: <https://gluonhq.com/products/scene-builder/>)

Un parser XML ::: (incluido en la plataforma: javax.xml)

Interactuar con una base de datos ::: (biblioteca incluida en la plataforma: JDBC: *java.sql* , *javax.sql*) Nota: el uso de esta biblioteca requiere del correspondiente driver JDBC para el motor de base de datos utilizado. Por ejemplo <https://jdbc.postgresql.org/> o <https://dev.mysql.com/doc/connectors/en/connector-j-usagenotes-basic.html>

Un analizador estático de código

::: Maven PMD Plugin: <https://maven.apache.org/plugins/maven-pmd-plugin>

::: SonarLint <https://www.sonarlint.org/>

(Nota: El uso de Maven -o de su sucesor Gradle-, con sus variados plugins para todo el ciclo de desarrollo, puede ser una buena forma de introducirse en el mundo de la integración/despliegue continuos (CI/CD).)

Un analizador de rendimiento (profiler) ::: No he encontrado uno de licencia libre que me llamara la atención; algunos comerciales pueden ser:

YourKit: <https://www.yourkit.com>

JProfiler: <https://www.ej-technologies.com/products/jprofiler/overview.html>

16.2. Para programar en C++

Un IDE ::: QT Creator: <https://www.qt.io/developers>

Un compilador-debugger ::: (incluido en la plataforma: gcc/g++)

Un entorno de pruebas unitarias ::: (incluido en la plataforma: QTest)

Un entorno de registro de eventos ::: (incluido en la plataforma: QMessageLogger)

Una biblioteca GUI ::: (incluida en la plataforma: QWidgets –interfaces estáticos–) (también incluida en la plataforma: QML/QtQuick –algo más compleja; pero permite implementar interfaces de usuario mucho más dinámicos–)

Un parser XML ::: (incluido en la plataforma: QXml)

Interactuar con una base de datos ::: (incluido en la plataforma: QPSQL)

Un analizador estático de código ::: (incluido en la plataforma: Clang Tools)

Un analizador de rendimiento (profiler) ::: Valgrind: <https://www.valgrind.org>

16.3. Para programar en C#

Un IDE ::: Visual Studio: <https://visualstudio.microsoft.com/es/vs/community/>

Un compilador-debugger ::: (incluido en la plataforma)

Un entorno de pruebas unitarias ::: (incluido en la plataforma)

Un entorno de registro de eventos ::: (incluido en la plataforma: *System.Diagnostics*)
o Nlog (paquete nuget)

o Log4net: <https://logging.apache.org/log4net>

Una biblioteca GUI ::: (incluida en la plataforma: *Forms* –interfaces estáticos–) o (también incluida en la plataforma: *WPF* –algo más compleja; pero permite implementar interfaces de usuario mucho más dinámicos–)

Un parser XML ::: (incluido en la plataforma: *System.Xml*)

Interactuar con una base de datos ::: (biblioteca incluida en la plataforma: ADO.NET: *System.Data*) Nota: el uso de esta biblioteca requiere del correspondiente conector para el motor de base de datos utilizado. Por ejemplo <https://www.npgsql.org/> o <https://dev.mysql.com/doc/connector-net/en/connector-net-introduction.html>

Un analizador estático de código ::: (incluido en la plataforma)

Un analizador de rendimiento (profiler) ::: (incluido en la plataforma)

16.4. Para programar en Python

Un intérprete-debugger ::: Python: <https://www.python.org>

Un IDE ::: Eclipse: <https://www.eclipse.org>
con el complemento PyDev: <https://www.pydev.org>

Un entorno de pruebas unitarias ::: (incluido en la plataforma) <https://docs.python.org/3/library/unittest.html>

Un entorno de registro de eventos ::: (incluido en la plataforma) <https://docs.python.org/3/library/logging.html>

Una biblioteca GUI ::: (incluida en la plataforma) <https://docs.python.org/3/library/tk.html>
(También hay otras ::: <https://docs.python.org/3/library/othergui.html>)

Un parser XML ::: (incluido en la plataforma) <https://docs.python.org/3/library/xml.html>

Interactuar con una base de datos ::: Psycopg: <https://www.psycopg.org>

Un analizador estático de código ::: PyLint: <https://www.pylint.org>

Un analizador de rendimiento (profiler) ::: PyVmMonitor: <https://www.pyvmmonitor.com>

nota: Python es un lenguaje interactivo que se está usando cada vez más en ámbitos científicos. En esos ámbitos se suele utilizar en forma de “cuaderno”, en vez de dentro de un ‘entorno de programación profesional’. Estos cuadernos pueden ser una buena manera de comenzar a trabajar con Python.

<https://jupyter.org/>

<https://github.com/jupyterlab/debugger>

<https://blog.jupyter.org/a-visual-debugger-for-jupyter-914e61716559>

nota: Entre otros proveedores de ese servicio, Google es uno de los que permite trabajar con cuadernos Jupyter online. En su plataforma Google Colab <https://colab.research.google.com/notebooks/intro.ipynb>

16.5. Para programar en HTML5/CSS/Javascript

Los tres componentes base:

- HTML: organiza el contenido de las páginas mostradas al usuario (frontend).
 - CSS: organiza la presentación, la apariencia de esas páginas.
 - Javascript: es el lenguaje de programación para el que prácticamente todos los navegadores web tienen soporte.
- nota: en la parte del servidor web (backend), además de Javascript, se pueden emplear también otros lenguajes de programación: Python, Typescript, Java, C#, PHP, Ruby, Go,...

El interface gráfico de usuario ::: páginas HTML/CSS/javascript estandar

::: MDN <https://developer.mozilla.org/en-US/docs/Web>

W3Consortium <https://www.w3.org/standards/webdesign/>

Complementadas con alguna biblioteca javascript auxiliar para facilitar el trabajo y conseguir interfaces más ricos. Por ejemplo:

::: Bootstrap.js <https://getbootstrap.com/>

::: Angular.js <https://angularjs.org/>

::: React.js <https://reactjs.org/>

::: Vue.js <https://vuejs.org/>

::: <https://openjsf.org/projects/>

::: etc, etc,

Un intérprete-debugger ::: los propios navegadores web suelen incluir herramientas para inspeccionar el código de las páginas visualizadas. (clic-dcho, 'Inspeccionar' o la tecla [F12] o...)

Una herramienta interesante ::: <https://www.browsersync.io> (Es un pequeño servidor web en sí mismo o un complemento de otro servidor web mayor. Permite actualizar rápidamente el contenido servido, facilitando así la prueba local de páginas según se van desarrollando.)

Para aplicaciones “grandes” se hace necesario contar con un servidor web “de producción” tal como, por ejemplo:

::: Apache <https://httpd.apache.org>

::: Nginx <https://nginx.org>

Estos servidores suelen proporcionar la plataforma necesaria para generar de forma dinámica las páginas web y para ejecutar aplicaciones de lógica y datos que las soporten.

En lugar de disponer de nuestros propios servidores (“on-premise”) y la infraestructura asociada a ellos, también es posible recurrir a los servicios de plataforma de alguien externo (“en la nube”), como por ejemplo:

<https://cloud.google.com/>
<https://aws.amazon.com/es/>
<https://azure.microsoft.com/es-es/>
<https://www.digitalocean.com/>
<https://www.heroku.com/home>

etc, etc, (cada vez hay más y más oferta de plataformas “cloud”)

16.6. Una herramienta transversal: almacenamiento de código con control de versiones

Git: <https://git-scm.com>

complementado con un interfaz gráfico para hacer más claro y sencillo el trabajo,

Sourcetree: <https://www.atlassian.com/software/sourcetree>

nota: Si se va a trabajar entre varias personas, y sobre todo si estas trabajan desde ubicaciones diferentes. Se necesita contar también con un almacenamiento central compartido. Por ejemplo, en la nube, **Bitbucket:** <https://www.atlassian.com/software/bitbucket>

16.7. Otra herramienta transversal: motor de base de datos

PostgreSQL: <https://www.postgresql.org>

o

MariaDB: <https://mariadb.org/>

Capítulo 17

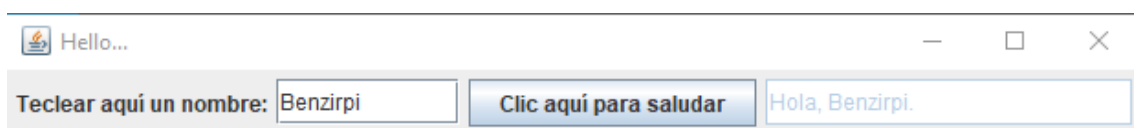
apéndice B: ejemplos de posibles soluciones...

17.1. ...a los ejercicios del capítulo 3

((una disculpa del autor: este capítulo esta “en construcción...” y sigo trabajando en él; queda mucho trabajo para completarlo.))

3.1 {Hello, Benzirpi}

En lenguaje Java (usando Swing)



Listing 17.1: 'module-info.java'

```
module hellobenzirpi {  
    requires java.desktop;  
}
```

Listing 17.2: 'InterfazGraficoDeUsuario.java'

```
package hello;  
  
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;
```

```

public class InterfazGraficoDeUsuario {

    private JFrame cuadro;
    private JTextField saludo;

    public InterfazGraficoDeUsuario()
    {
        cuadro = new JFrame();
        cuadro.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        cuadro.setTitle("Hello...");
        cuadro.getContentPane().setLayout(new FlowLayout());

        JLabel nombre_etiqueta = new JLabel("Teclear aqui un nombre:");
        JTextField nombre = new JTextField();
        nombre.setPreferredSize(new Dimension(100, 25));

        JButton saludar = new JButton("Clic aqui para saludar");
        saludar.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                saludo.setText("Hola, " + nombre.getText() + ".");
            }
        }));

        saludo = new JTextField();
        saludo.setPreferredSize(new Dimension(200, 25));
        saludo.setEnabled(false);

        cuadro.add(nombre_etiqueta);
        cuadro.add(nombre);
        cuadro.add(saludar);
        cuadro.add(saludo);
        cuadro.pack();
    }

    public void mostrarInterface()
    {
        cuadro.setVisible(true);
    }

}

```

Listing 17.3: 'Main.java'

```

package hello;

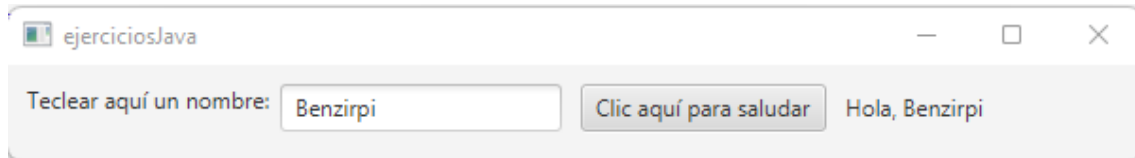
public class Main {

    public static void main(String[] args) {
        InterfazGraficoDeUsuario interfaz = new InterfazGraficoDeUsuario
            ();
        interfaz.mostrarInterface();
    }
}

```

 }

En lenguaje Java (usando JavaFx)



Listing 17.4: 'App.java'

```
package es.susosise.ejerciciosJava;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.fxml.FXMLLoader;
import javafx.stage.Stage;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.ButtonType;

public class App extends Application {

    public static void main(String[] args) {
        launch(args); // esto llama a start(...)
    }

    @Override
    public void start(Stage primaryStage) {

        try {
            java.net.URL location = getClass().getResource("
                HelloBenzirpi.fxml");
            Parent pantallaPrincipal = FXMLLoader.load(location);
            Scene scene = new Scene(pantallaPrincipal);
            primaryStage.setScene(scene);
            primaryStage.setTitle("ejerciciosJava");
            primaryStage.show();
        } catch (Exception ex) {
            Alert avisos = new Alert(AlertType.ERROR);
            avisos.setTitle("Error al crear la pantalla principal.");
            ;
            avisos.setContentText(ex.getMessage());
            avisos.showAndWait().ifPresent( respuesta -> {
                if (respuesta == ButtonType.OK) {
```

```

        Platform.exit();
    }
    } );
}

}

}

```

Listing 17.5: 'HelloBenzirpi.fxml'

```

<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.control.*?>

<GridPane fx:controller="es.susosise.ejerciciosJava.HelloBenzirpi"
  xmlns:fx="http://javafx.com/fxml"
    prefHeight="50" prefWidth="600"
    alignment="top_left" hgap="10" vgap="10">
  <padding><Insets top="10" left="10"/></padding>

  <HBox GridPane.rowIndex="0" GridPane.columnIndex="0"
    spacing="5">
    <Label text="Teclear aquí un nombre:" />
    <TextField fx:id="nombre"/>
  </HBox>

  <Button GridPane.rowIndex="0" GridPane.columnIndex="1"
    text="Clic aquí para saludar" onAction="#saludar" />

  <Label GridPane.rowIndex="0" GridPane.columnIndex="2"
    fx:id="saludo" />

</GridPane>

```

Listing 17.6: 'HelloBenzirpi.java'

```

package es.susosise.ejerciciosJava;

import javafx.fxml.FXML;
import javafx.event.ActionEvent;

public class HelloBenzirpi {

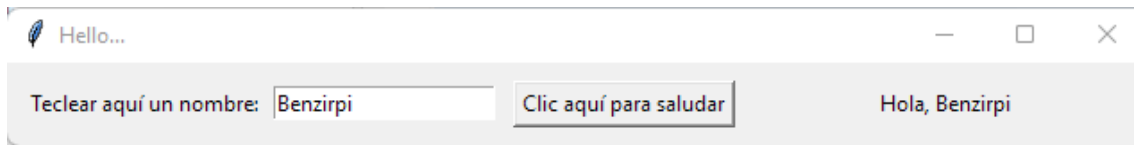
    @FXML private javafx.scene.control.TextField nombre;
    @FXML private javafx.scene.control.Label saludo;

    @FXML
    private void saludar(ActionEvent evento) {
        saludo.setText("Hola, " + nombre.getText());
    }
}

```

 }

En lenguaje Python



Listing 17.7: 'hello_benzirpi.py'

```
import tkinter as tk

def saludar():
    saludo["text"] = "Hola, " + nombre.get()

ventana = tk.Tk()
ventana.title("Hello...")

contenedor = tk.Frame()
contenedor.pack(padx=5, pady=5)

nombre_lbl = tk.Label(master=contenedor, text="Teclear aquí un
    nombre:")
nombre = tk.Entry(master=contenedor)
nombre_lbl.grid(row=0, column=0, sticky="e", padx=5)
nombre.grid(row=0, column=1)

saludar = tk.Button(
    master=contenedor,
    text="Clic aquí para saludar",
    command=saludar)
saludar.grid(row=0, column=2, padx=10, pady=5)

saludo = tk.Label(master=contenedor, width=30)
saludo.grid(row=0, column=3)

def arrancar():
    ventana.mainloop()
```

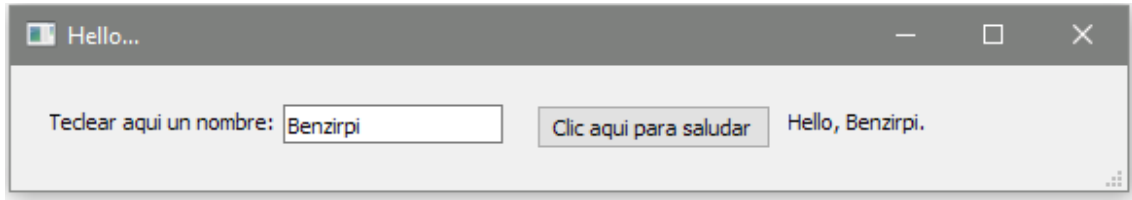
Listing 17.8: 'GUI.py'

```
import hello_benzirpi

def main():
    hello_benzirpi.arrancar()
```

```
if __name__ == '__main__':
    main()
```

En lenguaje C++, (definición 'fija' del interfaz)



Listing 17.9: 'mainwindow.h'

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_saludar_clicked();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Listing 17.10: 'mainwindow.cpp'

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
```

```

{
    delete ui;
}

void MainWindow::on_saludar_clicked()
{
    ui->saludo->setText("Hello, " + ui->nombre->text() + ".");
}

```

Listing 17.11: 'main.cpp'

```

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}

```

Listing 17.12: 'mainwindow.ui'

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>MainWindow</class>
  <widget class="QMainWindow" name="MainWindow">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>688</width>
        <height>88</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>Hello...</string>
    </property>
    <widget class="QWidget" name="centralwidget">
      <widget class="QLabel" name="nombre_etiqueta">
        <property name="geometry">
          <rect>
            <x>20</x>
            <y>20</y>
            <width>121</width>
            <height>16</height>
          </rect>
        </property>
        <property name="text">
          <string>Teclear aqui un nombre:</string>
        </property>
      </widget>
      <widget class="QLineEdit" name="nombre">

```

```
<property name="geometry">
  <rect>
    <x>140</x>
    <y>20</y>
    <width>113</width>
    <height>20</height>
  </rect>
</property>
</widget>
<widget class="QPushButton" name="saludar">
  <property name="geometry">
    <rect>
      <x>270</x>
      <y>20</y>
      <width>121</width>
      <height>23</height>
    </rect>
  </property>
  <property name="text">
    <string>Clic aqui para saludar</string>
  </property>
</widget>
<widget class="QLabel" name="saludo">
  <property name="geometry">
    <rect>
      <x>400</x>
      <y>20</y>
      <width>241</width>
      <height>16</height>
    </rect>
  </property>
  <property name="text">
    <string/>
  </property>
</widget>
</widget>
<widget class="QMenuBar" name="menubar">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>688</width>
      <height>21</height>
    </rect>
  </property>
</widget>
<widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```

En lenguaje C++, (definición 'flexible' del interfaz)



Listing 17.13: 'main.qml'

```

import QtQuick 2.12
import QtQuick.Window 2.12
import QtQuick.Layouts 1.3
import QtQuick.Controls 2.5

ApplicationWindow {
    visible: true
    title: qsTr("Hello...")
    minimumWidth: 600

    ScrollView {
        anchors.fill: parent
        padding: 10

        GridLayout {
            rowSpacing: 10
            columns: 2

            Label {
                id: nombre_etiqueta
                text: qsTr("Teclear aqui un nombre:")
            }

            TextField {
                id: nombre
                focus: true
            }

            Button {
                text: qsTr("Clic aqui para saludar")
                onClicked: {
                    saludo.text = qsTr("Hello, ") + nombre.text + qsTr("
                    .")
                }
            }

            TextField {
                id: saludo
                enabled: false
                background: Rectangle { color: "#fff6e6"}
                Layout.minimumWidth: 300
            }
        }
    }
}

```

```

    }
}
}

```

Listing 17.14: 'main.cpp'

```

#include <QGuiApplication>
#include <QQmlApplicationEngine>

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);

    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
    const QUrl url(QStringLiteral("qrc:/main.qml"));
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
                    &app, [url](QObject *obj, const QUrl &objUrl) {
        if (!obj && url == objUrl)
            QCoreApplication::exit(-1);
    }, Qt::QueuedConnection);
    engine.load(url);

    return app.exec();
}

```

3.2 {Bocatas}

En lenguaje Java

Listing 17.15: 'module-info.java'

```

module bocatas {
    requires java.desktop;
}

```


Listing 17.16: 'InterfazGraficoDeUsuario.java'

```
package bocatas;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class InterfazGraficoDeUsuario {

    private JFrame cuadroGeneral;
    private JTextField resultado;

    public InterfazGraficoDeUsuario()
    {
        cuadroGeneral = new JFrame();
        cuadroGeneral.setDefaultCloseOperation(WindowConstants.
            EXIT_ON_CLOSE);
        cuadroGeneral.setTitle("Bocatas...");
        cuadroGeneral.getContentPane().setLayout(new BorderLayout());

        JPanel cuadroCentral = new JPanel();
        cuadroCentral.setLayout(new GridLayout(0,3));

        ButtonGroup tipoDePan = new ButtonGroup();
        JRadioButton panNormal = new JRadioButton("normal");
        tipoDePan.add(panNormal);
        JRadioButton panIntegral = new JRadioButton("integral");
        tipoDePan.add(panIntegral);
        JRadioButton panBagete = new JRadioButton("bagete");
        tipoDePan.add(panBagete);
        JRadioButton panDeMolde = new JRadioButton("de molde");
        tipoDePan.add(panDeMolde);
        JPanel cuadroTipoDePan = new JPanel();
        cuadroTipoDePan.setBorder(BorderFactory.createTitledBorder(
            BorderFactory.createEtchedBorder(), "Tipo de pan:"));
        cuadroTipoDePan.setLayout(new GridLayout(4,1));
        cuadroTipoDePan.add(panNormal);
        cuadroTipoDePan.add(panIntegral);
        cuadroTipoDePan.add(panBagete);
        cuadroTipoDePan.add(panDeMolde);
        cuadroCentral.add(cuadroTipoDePan);

        ButtonGroup ingredientePrincipal = new ButtonGroup();
        JRadioButton lomo = new JRadioButton("lomo");
        ingredientePrincipal.add(lomo);
        JRadioButton tortillaJamonYork = new JRadioButton("tortilla de
            jamon York");
        ingredientePrincipal.add(tortillaJamonYork);
        JRadioButton tortillaAtun = new JRadioButton("tortilla de atun")
            ;
        ingredientePrincipal.add(tortillaAtun);
        JRadioButton tortillaQueso = new JRadioButton("tortilla de queso
            ");
        ingredientePrincipal.add(tortillaQueso);
        JRadioButton calamares = new JRadioButton("calamares");
```

```

ingredientePrincipal.add(calamares);
JPanel cuadroIngredientePrincipal = new JPanel();
cuadroIngredientePrincipal.setBorder(BorderFactory.
    createTitledBorder(BorderFactory.createEtchedBorder(), "
        Ingrediente principal:"));
cuadroIngredientePrincipal.setLayout(new GridLayout(6,1));
cuadroIngredientePrincipal.add(lomo);
cuadroIngredientePrincipal.add(tortillaJamonYork);
cuadroIngredientePrincipal.add(tortillaAtun);
cuadroIngredientePrincipal.add(tortillaQueso);
cuadroIngredientePrincipal.add(calamares);
cuadroCentral.add(cuadroIngredientePrincipal);

JPanel cuadroIngredientesAdicionales = new JPanel();
cuadroIngredientesAdicionales.setBorder(BorderFactory.
    createTitledBorder(BorderFactory.createEtchedBorder(), "
        Ingredientes adicionales:"));
cuadroIngredientesAdicionales.setLayout(new GridLayout(4,1));
JCheckBox cebollaFresca = new JCheckBox("cebolla fresca");
cuadroIngredientesAdicionales.add(cebollaFresca);
JCheckBox cebollaCaramelizada = new JCheckBox("cebolla
    caramelizada");
cuadroIngredientesAdicionales.add(cebollaCaramelizada);
JCheckBox pimientos = new JCheckBox("pimientos");
cuadroIngredientesAdicionales.add(pimientos);
JCheckBox queso = new JCheckBox("queso");
cuadroIngredientesAdicionales.add(queso);
cuadroCentral.add(cuadroIngredientesAdicionales);

cuadroGeneral.add(cuadroCentral, BorderLayout.CENTER);

JPanel cuadroInferior = new JPanel();
cuadroInferior.setLayout(new FlowLayout());

JButton componerBocata = new JButton("Clic aqui para componer el
    bocata");
componerBocata.setSize(50, 50);
componerBocata.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        StringBuilder composicionDelBocata = new StringBuilder()
            ;

        composicionDelBocata.append("Bocata con ");
        if (panNormal.isSelected()) {composicionDelBocata.append
            (" pan normal");}
        if (panIntegral.isSelected()) {composicionDelBocata.
            append(" pan integral");}
        if (panBagete.isSelected()) {composicionDelBocata.append
            (" pan bagete");}
        if (panDeMolde.isSelected()) {composicionDelBocata.
            append(" pan de molde");}

        composicionDelBocata.append(", de ");
        if (lomo.isSelected()) {composicionDelBocata.append("

```

```

        lomo");}
    if (tortillaJamonYork.isSelected()) {
        composicionDelBocata.append("tortilla de jamon York"
        );}
    if (tortillaAtun.isSelected()) {composicionDelBocata.
        append("tortilla de atun");}
    if (tortillaQueso.isSelected()) {composicionDelBocata.
        append("tortilla de queso");}
    if (calamares.isSelected()) {composicionDelBocata.append
        ("calamares");}

    if (cebollaFresca.isSelected()) {composicionDelBocata.
        append(", cebolla fresca");}
    if (cebollaCaramelizada.isSelected()) {
        composicionDelBocata.append(", cebolla caramelizada"
        );}
    if (pimientos.isSelected()) {composicionDelBocata.append
        ("", pimientos");}
    if (queso.isSelected()) {composicionDelBocata.append(",
        queso");}
    composicionDelBocata.append(".");

    resultado.setText(composicionDelBocata.toString());
    }
    });
    cuadroInferior.add(componerBocata);

    JLabel resultado_etiqueta = new JLabel("Resultado:");
    cuadroInferior.add(resultado_etiqueta);

    resultado = new JTextField();
    resultado.setPreferredSize(new Dimension(500, 25));
    resultado.setEnabled(false);
    cuadroInferior.add(resultado);

    cuadroGeneral.add(cuadroInferior, BorderLayout.SOUTH);

    cuadroGeneral.pack();
}

public void mostrarInterface()
{
    cuadroGeneral.setVisible(true);
}

}

```

Listing 17.17: 'Main.java'

```

package bocatas;

import bocatas.InterfazGraficoDeUsuario;

public class Main {

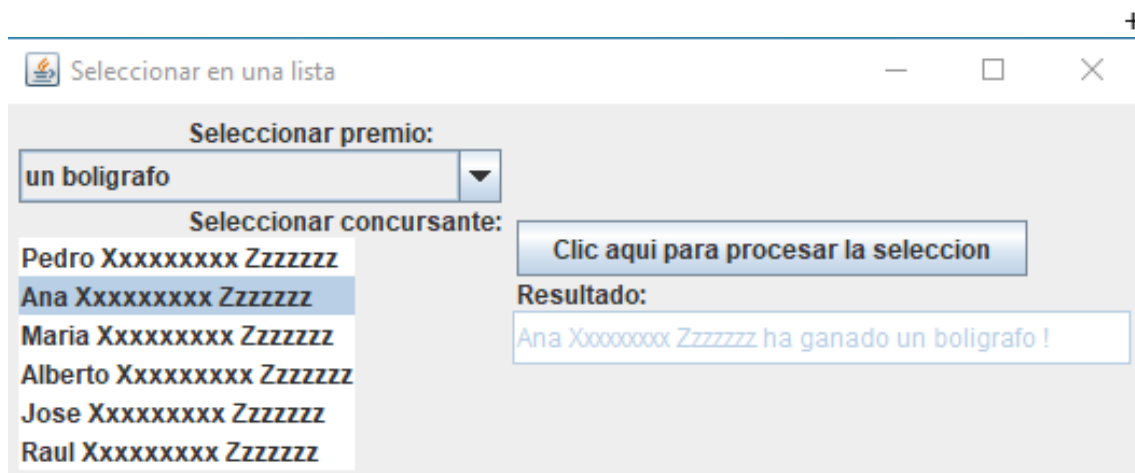
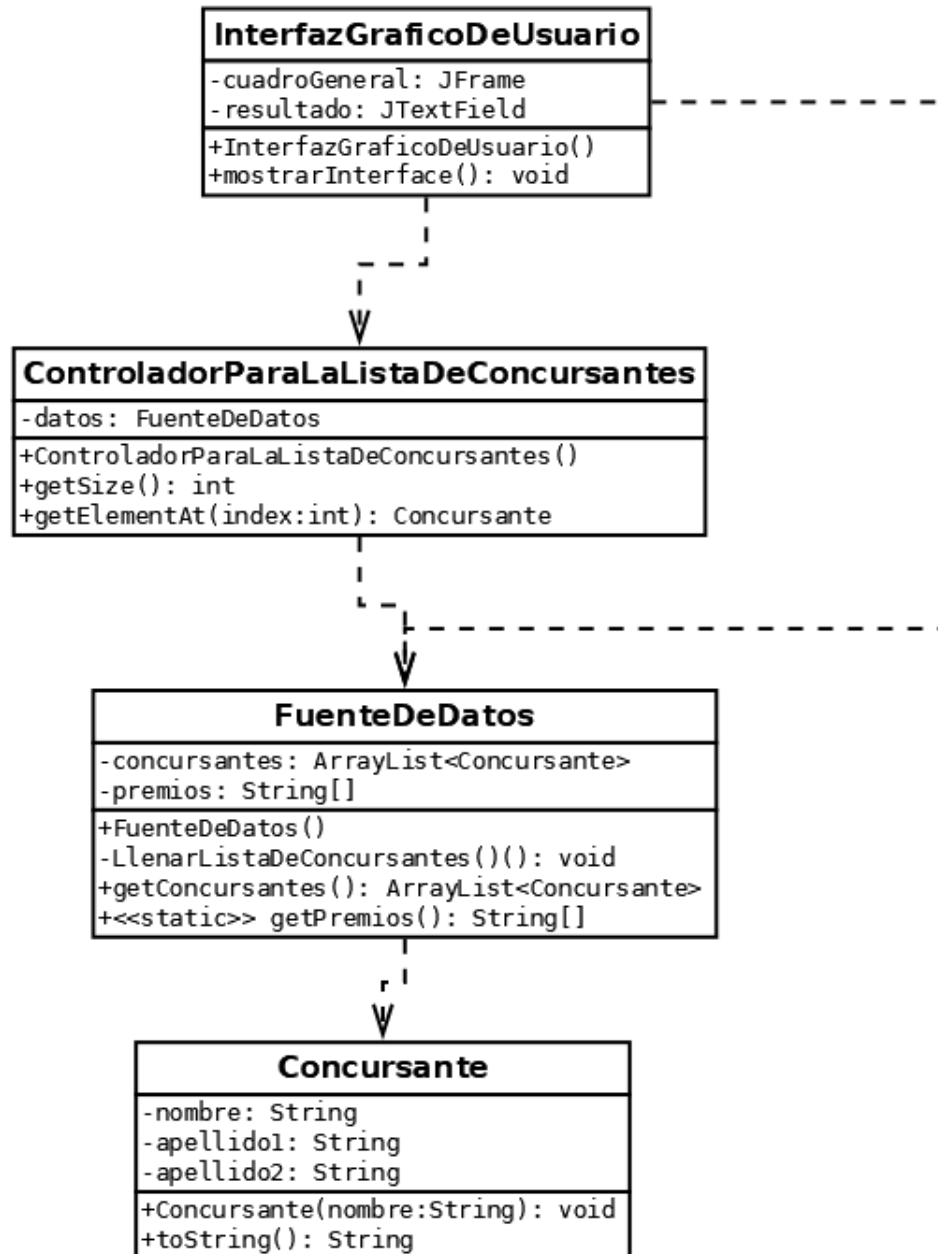
```

```
public static void main(String[] args) {  
    InterfazGraficoDeUsuario interfaz = new InterfazGraficoDeUsuario  
        ();  
    interfaz.mostrarInterface();  
}  
  
}
```

3.3 {Selección plana}

En lenguaje Java

+



Listing 17.18: 'module-info.java'

```
module seleccionPlana {  
    requires java.desktop;  
}
```

Listing 17.19: 'InterfazGraficoDeUsuario.java'

```
package seleccionPlana;  
  
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
  
public class InterfazGraficoDeUsuario {  
  
    private JFrame cuadroGeneral;  
    private JTextField resultado;  
  
    public InterfazGraficoDeUsuario()  
    {  
        cuadroGeneral = new JFrame();  
        cuadroGeneral.setDefaultCloseOperation(WindowConstants.  
            EXIT_ON_CLOSE);  
        cuadroGeneral.setTitle("Seleccionar en una lista");  
        cuadroGeneral.getContentPane().setLayout(new FlowLayout());  
  
        JPanel cuadroSecundario1 = new JPanel();  
        cuadroSecundario1.setLayout(new BorderLayout(cuadroSecundario1,  
            BorderLayout.PAGE_AXIS));  
  
        JLabel premio_etiqueta = new JLabel("Seleccionar premio:");  
        cuadroSecundario1.add(premio_etiqueta);  
  
        JComboBox<String> listaDePremios = new JComboBox<String>(  
            FuenteDeDatos.getPremios());  
        cuadroSecundario1.add(listaDePremios);  
  
        JLabel concursante_etiqueta = new JLabel("Seleccionar  
            concursante:");  
        cuadroSecundario1.add(concursante_etiqueta);  
  
        JList<Concursante> listaDeConcursantes = new JList<Concursante  
            >();  
        listaDeConcursantes.setLayoutOrientation(JList.VERTICAL);  
        listaDeConcursantes.setSelectionMode(ListSelectionModel.  
            SINGLE_SELECTION);  
        listaDeConcursantes.setModel(new  
            ControladorParaLaListaDeConcursantes());  
        cuadroSecundario1.add(listaDeConcursantes);  
  
        cuadroGeneral.add(cuadroSecundario1);  
  
        JPanel cuadroSecundario2 = new JPanel();  
        cuadroSecundario2.setLayout(new BorderLayout(cuadroSecundario2,  
            BorderLayout.PAGE_AXIS));
```

```

        JButton procesarSeleccion = new JButton("Clic aqui para procesar
            la seleccion");
        procesarSeleccion.setSize(50, 50);
        procesarSeleccion.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                if (listaDeConcursantes.getSelectedValue() == null)
                {
                    resultado.setText("no se ha seleccionado ningun
                        concursante...");
                }
                else
                {
                    resultado.setText(listaDeConcursantes.getSelectedValue()
                        .toString()
                        + " ha ganado " + listaDePremios.
                            getSelectedItem()
                        + " ! ");
                }
            }
        });
        cuadroSecundario2.add(procesarSeleccion);

        JLabel resultado_etiqueta = new JLabel("Resultado:");
        cuadroSecundario2.add(resultado_etiqueta);

        resultado = new JTextField();
        resultado.setPreferredSize(new Dimension(200, 25));
        resultado.setEnabled(false);
        cuadroSecundario2.add(resultado);

        cuadroGeneral.add(cuadroSecundario2);

        cuadroGeneral.pack();
    }

    public void mostrarInterface()
    {
        cuadroGeneral.setVisible(true);
    }

}

```

Listing 17.20: 'ControladorParaLaListaDeConcursantes.java'

```

package seleccionPlana;

import javax.swing.AbstractListModel;

public class ControladorParaLaListaDeConcursantes extends
    AbstractListModel<Concursante> {

```

```
private FuenteDeDatos datos;

public ControladorParaLaListaDeConcursantes()
{
    datos = new FuenteDeDatos();
}

@Override
public int getSize() {
    return datos.getConcursantes().size();
}

@Override
public Concursante getElementAt(int index) {
    return datos.getConcursantes().get(index);
}

}
```

Listing 17.21: 'FuenteDeDatos.java'

```
package seleccionPlana;

import java.util.ArrayList;

public class FuenteDeDatos {

    private final ArrayList<Concursante> concursantes;
    private static String[] premios = {"un adorable peluche", "un
        boligrafo", "un paquete de folios"};

    public FuenteDeDatos()
    {
        concursantes = new ArrayList<Concursante>();
        LlenarListaDeConcursantes();
    }

    private void LlenarListaDeConcursantes()
    {
        concursantes.add(new Concursante("Pedro"));
        concursantes.add(new Concursante("Ana"));
        concursantes.add(new Concursante("Maria"));
        concursantes.add(new Concursante("Alberto"));
        concursantes.add(new Concursante("Jose"));
        concursantes.add(new Concursante("Raul"));
    }

    public ArrayList<Concursante> getConcursantes()
    {
        return concursantes;
    }

    public static String[] getPremios()
```



```
    {  
        return premios;  
    }  
  
}
```

Listing 17.22: 'Concursante.java'

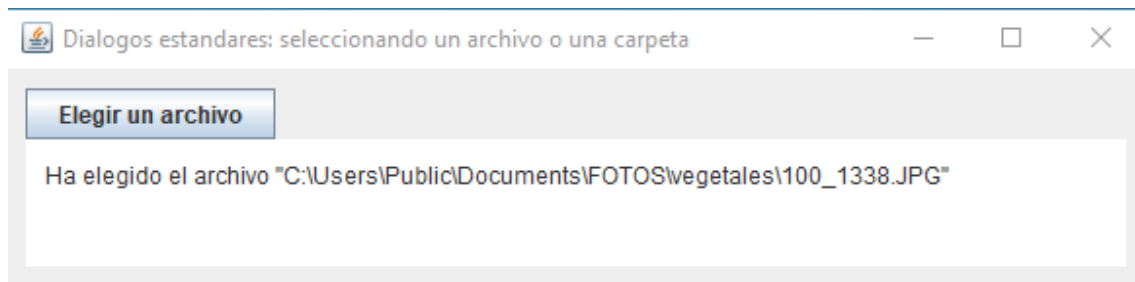
```
package seleccionPlana;  
  
public class Concursante {  
    private String nombre;  
    private String apellido1;  
    private String apellido2;  
  
    public Concursante(String nombre)  
    {  
        if (nombre == null || nombre.length() == 0 )  
        {  
            throw new IllegalArgumentException();  
        }  
        else  
        {  
            this.nombre = nombre.substring(0, 1).toUpperCase() + nombre.  
                substring(1);  
            this.apellido1 = "XXXXXXXXX";  
            this.apellido2 = "ZZZZZZZ";  
        }  
    }  
  
    public String toString()  
    {  
        return nombre + " " + apellido1 + " " + apellido2;  
    }  
  
}
```

Listing 17.23: 'Main.java'

```
package seleccionPlana;  
  
import seleccionPlana.InterfazGraficoDeUsuario;  
  
public class Main {  
  
    public static void main(String[] args) {  
        InterfazGraficoDeUsuario interfaz = new InterfazGraficoDeUsuario  
            ();  
        interfaz.mostrarInterface();  
    }  
  
}
```

3.4 {Diálogos estándares}

En lenguaje Java



Listing 17.24: 'module-info.java'

```
module dialogosEstandares {
    requires java.desktop;
}
```

Listing 17.25: 'EjercitandoJFileChooser.java'

```
package dialogosEstandares;

import javax.swing.JFrame;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JTextArea;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import java.io.File;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class EjercitandoJFileChooser {

    public void run()
    {
        JFrame ventana = new JFrame("Dialogos estandares: seleccionando
            un archivo o una carpeta");
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setPreferredSize(new Dimension(600, 150));
        JPanel cuadroGeneral = new JPanel();
        cuadroGeneral.setLayout(new BoxLayout(cuadroGeneral, BoxLayout.
            Y_AXIS));
        cuadroGeneral.setBorder(BorderFactory.createEmptyBorder(10, 10,
            10, 10));

        JTextArea resultado = new JTextArea();
        resultado.setPreferredSize(new Dimension(500, 25));
        resultado.setBorder(BorderFactory.createEmptyBorder(10, 10, 10,
            10));
    }
}
```

```

        resultado.setAlignmentX(JTextArea.LEFT_ALIGNMENT);

        JButton boton = new JButton("Elegir un archivo");
        boton.setSize(new Dimension(200, 50));
        boton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                File archivo = ElegirUnArchivo();
                if (archivo == null)
                {
                    resultado.setText("no se ha elegido ningun archivo");
                }
                else
                {
                    resultado.setText("Ha elegido el archivo '" + archivo.
                        getAbsolutePath() + "'");
                }
            }
        });

        cuadroGeneral.add(boton);
        cuadroGeneral.add(resultado);
        ventana.add(cuadroGeneral);
        ventana.pack();
        ventana.setVisible(true);
    }

    private File ElegirUnArchivo()
    {
        final JFileChooser selectorDeArchivos = new JFileChooser();
        int resultado = selectorDeArchivos.showOpenDialog(new JFrame());

        if (resultado == JFileChooser.APPROVE_OPTION)
        {
            return selectorDeArchivos.getSelectedFile();
        }
        else
        {
            return null;
        }
    }
}

```

Listing 17.26: 'EjercitandoJColorChooser.java'

```

package dialogosEstandares;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.JLabel;

```

```

import java.awt.Font;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JColorChooser;

public class EjercitandoJColorChooser {

    public void run()
    {
        JFrame ventana = new JFrame("Dialogos estandares: elegir colores
        ");
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel cuadroGeneral = new JPanel();
        cuadroGeneral.setLayout(new BoxLayout(cuadroGeneral, BoxLayout.
        Y_AXIS));
        cuadroGeneral.setBorder(BorderFactory.createEmptyBorder(10, 10,
        10, 10));

        JLabel texto = new JLabel("ESTE TEXTO SE PUEDE CAMBIAR DE COLOR"
        );
        texto.setAlignmentX(JLabel.CENTER_ALIGNMENT);
        texto.setBorder(BorderFactory.createEmptyBorder(0, 0, 20, 0));
        texto.setFont(new Font("Arial", Font.BOLD, 24));

        final JColorChooser selectorDeColores = new JColorChooser();
        selectorDeColores.setPreviewPanel(new JPanel());
        selectorDeColores.getSelectionModel().addChangeListener(new
        ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                texto.setForeground(selectorDeColores.getColor());
            }
        });

        cuadroGeneral.add(texto);
        cuadroGeneral.add(selectorDeColores);
        ventana.add(cuadroGeneral);
        ventana.pack();
        ventana.setVisible(true);
    }

}

```

Listing 17.27: 'Main.java'

```

package dialogosEstandares;

public class Main {

    public static void main(String[] args) {

        new EjercitandoJFileChooser().run();

        //new EjercitandoJColorChooser().run();
    }
}

```

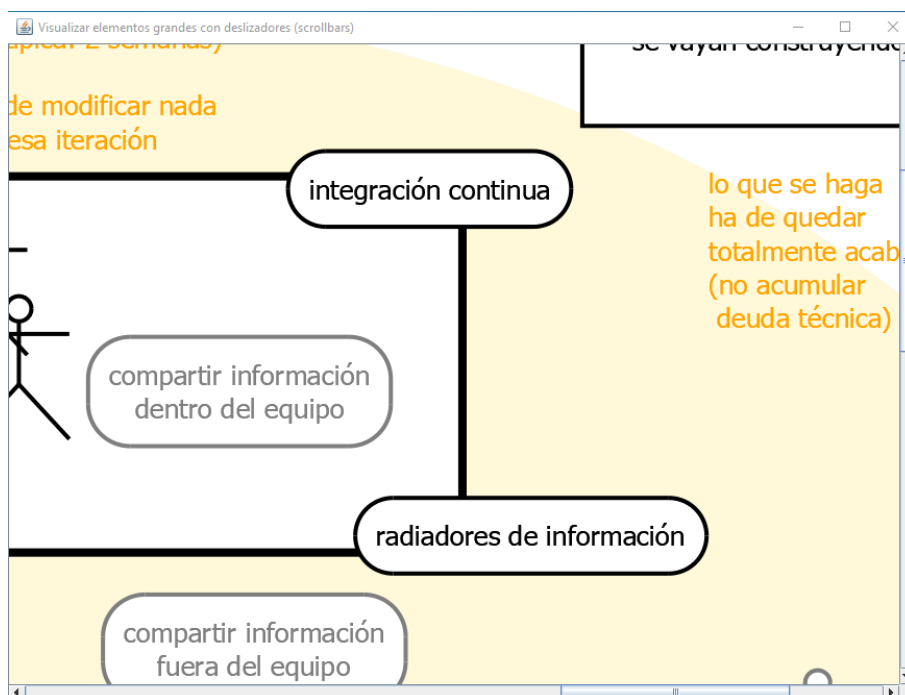
```

    }
}

```

3.5{Scroll}

En lenguaje Java



Listing 17.28: 'module-info.java'

```

module scroll {
    requires java.desktop;
}

```

Listing 17.29: 'InterfazGraficoDeUsuario.java'

```

package scroll;

import java.awt.*;
import javax.swing.*;
import java.awt.image.BufferedImage;

import java.io.File;
import java.io.IOException;

```

```
import java.io.PrintWriter;
import java.io.StringWriter;

import javax.imageio.ImageIO;

public class InterfazGraficoDeUsuario {

    private JFrame cuadroGeneral;

    public InterfazGraficoDeUsuario()
    {
        cuadroGeneral = new JFrame();
        cuadroGeneral.setDefaultCloseOperation(WindowConstants.
            EXIT_ON_CLOSE);
        cuadroGeneral.setPreferredSize(new Dimension(800, 600));
        cuadroGeneral.setTitle("Visualizar elementos grandes con
            deslizadores (scrollbars)");
        cuadroGeneral.getContentPane().setLayout(new BorderLayout(10,
            10));

        BufferedImage imagen = null;
        String pathArchivoImagen = "..\\..\\..\\imagenes\\Esquema general de
            una metodologia agil.png";
        try
        {
            imagen = ImageIO.read(new File(pathArchivoImagen));
        }
        catch(IOException ex)
        {
            StringWriter detallesDelError = new StringWriter();
            PrintWriter paraObtenerLosDetalles = new PrintWriter(
                detallesDelError);
            ex.printStackTrace(new PrintWriter(paraObtenerLosDetalles));
            JOptionPane.showMessageDialog(null, "Problemas para leer la
                imagen del archivo " + pathArchivoImagen
                    + System.lineSeparator() +
                    System.lineSeparator() +
                    detallesDelError.toString
                        ());
        }
        ComponenteParaMostarUnaImagen paraMostrarLaImagen = new
            ComponenteParaMostarUnaImagen(imagen);
        paraMostrarLaImagen.setPreferredSize(new Dimension(imagen.
            getWidth(), imagen.getHeight()));

        JScrollPane panelImagen = new JScrollPane(paraMostrarLaImagen);
        panelImagen.setHorizontalScrollBarPolicy(JScrollPane.
            HORIZONTAL_SCROLLBAR_AS_NEEDED);
        panelImagen.setVerticalScrollBarPolicy(JScrollPane.
            VERTICAL_SCROLLBAR_AS_NEEDED);

        cuadroGeneral.add(panelImagen, BorderLayout.CENTER);

        cuadroGeneral.pack();
    }
}
```

```
    public void mostrarInterface()
    {
        cuadroGeneral.setVisible(true);
    }
}
```

Listing 17.30: 'ComponenteParaMostarUnaImagen.java'

```
package scroll;

import java.awt.Graphics;
import java.awt.image.BufferedImage;

import javax.swing.JPanel;

public class ComponenteParaMostarUnaImagen extends JPanel {

    private static final long serialVersionUID = 5661127745289954328L;

    BufferedImage imagen;

    public ComponenteParaMostarUnaImagen(BufferedImage imagen) {
        this.imagen = imagen;
    }

    @Override
    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        g.drawImage(imagen, 0, 0, null);
    }

}
```

Listing 17.31: 'Main.java'

```
package scroll;

import scroll.InterfazGraficoDeUsuario;

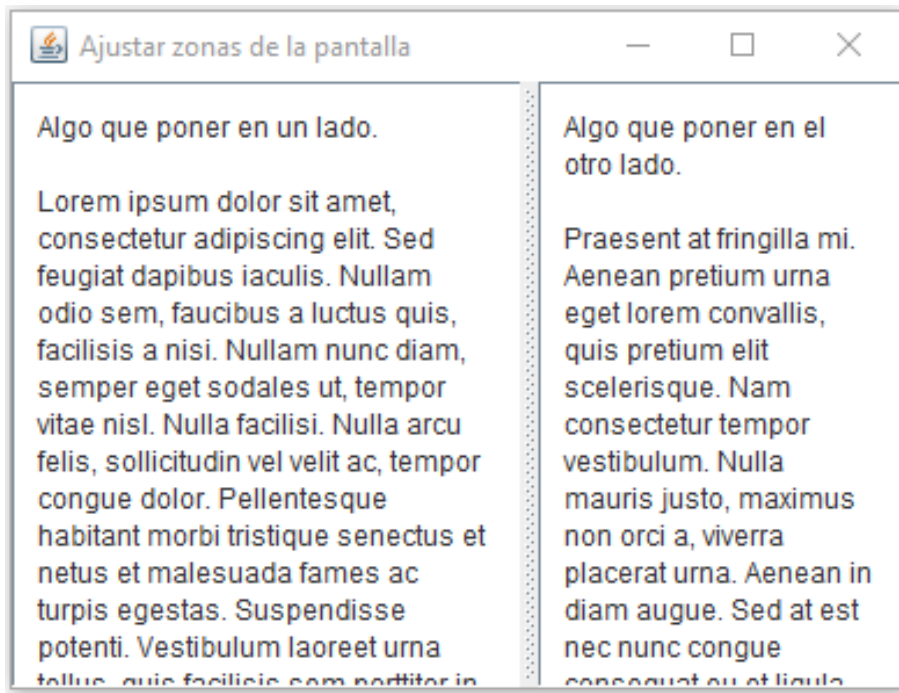
public class Main {

    public static void main(String[] args) {
        InterfazGraficoDeUsuario interfaz = new InterfazGraficoDeUsuario
            ();
        interfaz.mostrarInterface();
    }

}
```

3.6 {Split}

En lenguaje Java



Listing 17.32: 'module-info.java'

```
module split {
    requires java.desktop;
}
```

Listing 17.33: 'InterfazGraficoDeUsuario.java'

```
package split;

import java.awt.*;
import javax.swing.*;
import javax.swing.border.EmptyBorder;

public class InterfazGraficoDeUsuario {

    private JFrame cuadroGeneral;

    public InterfazGraficoDeUsuario()
    {
        cuadroGeneral = new JFrame();
        cuadroGeneral.setDefaultCloseOperation(WindowConstants.
            EXIT_ON_CLOSE);
        cuadroGeneral.setPreferredSize(new Dimension(400, 300));
        cuadroGeneral.setTitle("Ajustar zonas de la pantalla");
        cuadroGeneral.getContentPane().setLayout(new BorderLayout(10,
            10));
    }
}
```



```

JTextArea unLado = new JTextArea();
unLado.setMinimumSize(new Dimension(25, 0));
unLado.setBorder(new EmptyBorder(10,10, 10, 10));
unLado.setEditable(false);
unLado.setLineWrap(true);
unLado.setWrapStyleWord(true);
unLado.setText("Algo que poner en un lado."
+ System.lineSeparator() + System.lineSeparator()
+ "Lorem ipsum dolor sit amet, consectetur
  adipiscing elit. Sed feugiat dapibus iaculis.
  Nullam odio sem, faucibus a luctus quis,
  facilisis a nisi. Nullam nunc diam, semper
  eget sodales ut, tempor vitae nisl. Nulla
  facilisi. Nulla arcu felis, sollicitudin vel
  velit ac, tempor congue dolor. Pellentesque
  habitant morbi tristique senectus et netus et
  malesuada fames ac turpis egestas.
  Suspendisse potenti. Vestibulum laoreet urna
  tellus, quis facilisis sem porttitor in. Ut
  finibus ac turpis vitae tristique. Cras eu
  felis diam. Cras hendrerit odio efficitur
  massa eleifend, ut egestas ipsum commodo.
  Etiam velit lectus, sodales nec mi vitae,
  consequat lacinia dui.");

JTextArea otroLado = new JTextArea();
otroLado.setMinimumSize(new Dimension(25, 0));
otroLado.setBorder(new EmptyBorder(10,10, 10, 10));
otroLado.setEditable(false);
otroLado.setLineWrap(true);
otroLado.setWrapStyleWord(true);
otroLado.setText("Algo que poner en el otro lado."
+ System.lineSeparator() + System.
  lineSeparator()
+ "Praesent at fringilla mi. Aenean pretium
  urna eget lorem convallis, quis
  pretium elit scelerisque. Nam
  consectetur tempor vestibulum. Nulla
  mauris justo, maximus non orci a,
  viverra placerat urna. Aenean in diam
  augue. Sed at est nec nunc congue
  consequat eu et ligula. Donec maximus
  ipsum at finibus facilisis. Fusce
  volutpat leo vitae nisi cursus
  venenatis in eu ipsum. Ut tempus arcu
  ut metus scelerisque mattis. Etiam
  egestas arcu nec libero pellentesque,
  ac euismod sapien convallis. Proin
  sollicitudin magna ac velit aliquam
  convallis. Nunc scelerisque nisi orci,
  nec bibendum massa maximus non. ");

JSplitPane panelGeneral = new JSplitPane(JSplitPane.
  HORIZONTAL_SPLIT,
  unLado, otroLado);
panelGeneral.setResizeWeight(0.5);

```

```
        cuadroGeneral.add(panelGeneral, BorderLayout.CENTER);

        cuadroGeneral.pack();
    }

    public void mostrarInterface()
    {
        cuadroGeneral.setVisible(true);
    }
}
```

split

Listing 17.34: 'Main.java'

```
package split;

import split.InterfazGraficoDeUsuario;

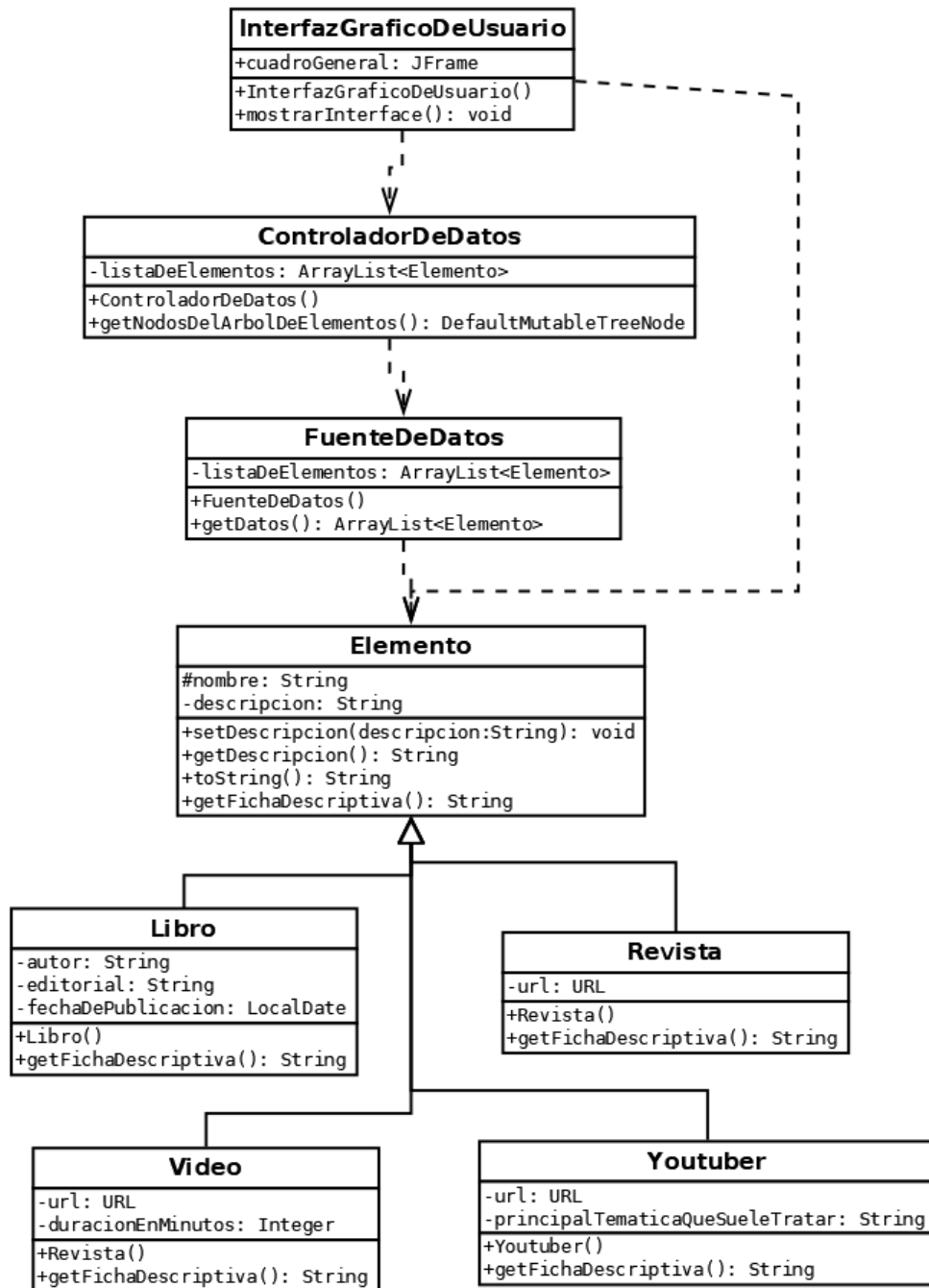
public class Main {

    public static void main(String[] args) {
        InterfazGraficoDeUsuario interfaz = new InterfazGraficoDeUsuario
            ();
        interfaz.mostrarInterface();
    }
}
```

3.8{Selección arbórea}

En lenguaje Java

+



Seleccionar en una lista

- Lecturas y entretenimientos interesantes
 - Libros
 - Clean Code: A Handbook of Agile Software Craftsmanship
 - Refactoring: Improving the Design of Existing Code**
 - Mastering Regular Expressions
 - Semantic Web for the Working Ontologist: Effective modeling in RDFS and OWL
 - Critical Chain Project Management
 - Revistas
 - Java Magazine
 - IEEE Spectrum
 - Wired
 - Make:
 - Videos
 - Uncle Bob Martin - The Future of Programming
 - El abuelo Amu crea un puente de arco de madera
 - Gothic Cathedrals
 - F-15C Grim Reapers, Low Level Mach-Loop

'Refactoring: Improving the Design of Existing Code'
Martin Fowler
(2019) Addison-Wesley

Un clásico que contribuyó a popularizar el término "code smell" (pequeños indicios en el código que nos van indicando posibles futuros problemas graves en el proyecto). La principal aportación de este libro es hacer ver la importancia de refactorizar con asiduidad, para mantener continuamente la buena salud del código. Y la importancia de los test, como red de seguridad para facilitar esa refactorización.

Listing 17.35: 'module-info.java'

```

module seleccionArborea {
    requires java.desktop;
}

```

Listing 17.36: 'InterfazGraficoDeUsuario.java'

```

package seleccionArborea;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeSelectionModel;

public class InterfazGraficoDeUsuario {

    private JFrame cuadroGeneral;

    public InterfazGraficoDeUsuario()
    {
        cuadroGeneral = new JFrame();
        cuadroGeneral.setDefaultCloseOperation(WindowConstants.
            EXIT_ON_CLOSE);
        cuadroGeneral.setPreferredSize(new Dimension(800, 600));
        cuadroGeneral.setTitle("Seleccionar en una lista");
        cuadroGeneral.getContentPane().setLayout(new BorderLayout(10,
            10));

        JTree listaEnArbol = new JTree(new ControladorDeDatos().
            getNodosDelArbolDeElementos());
        listaEnArbol.setBorder(new EmptyBorder(10, 10, 10, 10));
        listaEnArbol.getSelectionModel().setSelectionMode(
            TreeSelectionModel.SINGLE_TREE_SELECTION);
        JScrollPane panelParaLaListaEnArbol = new JScrollPane(
            listaEnArbol);
        cuadroGeneral.add(panelParaLaListaEnArbol, BorderLayout.WEST);

        JTextArea resultado = new JTextArea();
        resultado.setBorder(new EmptyBorder(10, 10, 10, 10));
        resultado.setEditable(false);
        resultado.setLineWrap(true);
        resultado.setWrapStyleWord(true);
        JScrollPane panelParaElResultado = new JScrollPane(resultado);
        cuadroGeneral.add(panelParaElResultado, BorderLayout.CENTER);

        listaEnArbol.addTreeSelectionListener(new TreeSelectionListener
            ()
        {
            @Override
            public void valueChanged(TreeSelectionEvent e) {
                DefaultMutableTreeNode nodoSeleccionado = (

```

```

        DefaultMutableTreeNode) listaEnArbol.
        getLastSelectedPathComponent();
    if (nodoSeleccionado != null)
    {
        if (nodoSeleccionado.isLeaf())
        {
            Elemento elementoSeleccionado = (Elemento)
                nodoSeleccionado.getUserObject();
            resultado.setText(elementoSeleccionado.
                getFichaDescriptiva());
        }
    }
}
});

cuadroGeneral.pack();
}

public void mostrarInterface()
{
    cuadroGeneral.setVisible(true);
}

}

```

Listing 17.37: 'ControladorDeDatos.java'

```

package seleccionArborea;

import java.net.MalformedURLException;
import java.util.ArrayList;
import javax.swing.tree.DefaultMutableTreeNode;

public class ControladorDeDatos {

    private ArrayList<Elemento> listaDeElementos;

    public ControladorDeDatos()
    {
        this.listaDeElementos = new FuenteDeDatos().getDatos();
    }

    public DefaultMutableTreeNode getNodosDelArbolDeElementos()
    {
        DefaultMutableTreeNode raiz = new DefaultMutableTreeNode("
            Lecturas y entretenimientos interesantes");

        DefaultMutableTreeNode libros = new DefaultMutableTreeNode("
            Libros");
        DefaultMutableTreeNode revistas = new DefaultMutableTreeNode("
            Revistas");
        DefaultMutableTreeNode videos = new DefaultMutableTreeNode("
            Videos");
        DefaultMutableTreeNode youtubers = new DefaultMutableTreeNode("
            Youtubers");
    }
}

```

```

    for (Elemento elemento : listaDeElementos)
    {
        if (elemento instanceof Libro)
        {
            DefaultMutableTreeNode unLibro = new DefaultMutableTreeNode(
                elemento.nombre);
            unLibro.setUserObject((Libro) elemento);
            libros.add(unLibro);
        }
        if (elemento instanceof Revista)
        {
            DefaultMutableTreeNode unaRevista = new
                DefaultMutableTreeNode(elemento.nombre);
            unaRevista.setUserObject((Revista) elemento);
            revistas.add(unaRevista);
        }
        if (elemento instanceof Video)
        {
            DefaultMutableTreeNode unVideo = new DefaultMutableTreeNode(
                elemento.nombre);
            unVideo.setUserObject((Video) elemento);
            videos.add(unVideo);
        }
        if (elemento instanceof Youtuber)
        {
            DefaultMutableTreeNode unYoutuber = new
                DefaultMutableTreeNode(elemento.nombre);
            unYoutuber.setUserObject((Youtuber) elemento);
            youtubers.add(unYoutuber);
        }
    }

    raiz.add(libros);
    raiz.add(revistas);
    raiz.add(videos);
    raiz.add(youtubers);
    return raiz;
}

}

```

Listing 17.38: 'FuenteDeDatos.java'

```

package seleccionArborea;

import java.net.MalformedURLException;
import java.net.URL;
import java.time.LocalDate;
import java.util.ArrayList;

public class FuenteDeDatos {

    private final ArrayList<Elemento> listaDeElementos;

    public FuenteDeDatos()

```

```
{
    listaDeElementos = new ArrayList<Elemento>();

    Libro libro001 = new Libro("Clean Code: A Handbook of Agile
        Software Craftsmanship",
        "Robert C. Martin (aka 'uncle Bob')",
        "Prentice Hall", LocalDate.of(2008 , 1,
            1));
    libro001.setDescripcion("La importancia de escribir código
        sencillo de leer, sencillo de entender y sencillo de
        modificar.");
    listaDeElementos.add(libro001);

    Libro libro002 = new Libro("Refactoring: Improving the Design of
        Existing Code",
        "Martin Fowler ",
        "Addison-Wesley", LocalDate.of
            (2019 , 1, 1));
    libro002.setDescripcion(" Un clásico que contribuyó a
        popularizar el término 'code smell' (pequeños indicios
        en el código que nos van indicando posibles futuros
        problemas graves en el proyecto). La principal aportació
        n de este libro es hacer ver la importancia de
        refactorizar con asiduidad, para mantener continuamente
        la buena salud del código. Y la importancia de los test,
        como red de seguridad para facilitar esa refactorizació
        n.");
    listaDeElementos.add(libro002);

    Libro libro003 = new Libro("Mastering Regular Expressions",
        "Jeffrey E.F. Friedl.",
        "O'Reilly", LocalDate.of(2006 ,
            1, 1));
    listaDeElementos.add(libro003);

    Libro libro004 = new Libro("Semantic Web for the Working
        Ontologist: Effective modeling in RDFS and OWL",
        "Dean Allemang and James Hendler ",
        "Morgan Kaufmann", LocalDate.of(2011
            , 05, 20));
    libro004.setDescripcion("Otra mirada acerca del tratamiento de
        datos; no todo en esta vida son datos tabulares...");
    listaDeElementos.add(libro004);

    Libro libro005 = new Libro("Critical Chain Project Management",
        "Lawrence P. Leach ",
        "Artech House Publishers", LocalDate.
            of(2014 , 2, 28));
    libro005.setDescripcion("Herramientas para aplicar de forma prá
        ctica las ideas expuestas por Goldratt en su libro homónimo.
        Son herramientas y conceptos muy útiles cuando se necesita
        realmente completar un proyecto dentro de unos parámetros
        concretos preestablecidos. En proyectos software, pueden
        servir para hacer algo útil cuando alguien se empeña en
        pedirte un cronograma, un Gant, un Pert o algo similar (
        todas esas herramientas son prácticamente inútiles en un
```

[illegible]


```

        yLRfog"),
        "En Guédelon están construyendo
        un castillo siguiendo té
        cnicas ancestrales.");

    listaDeElementos.add(youtub002);

    Youtuber youtub003 = new Youtuber("Protolabs Manufacturing
        Services",
        new URL("https://www.youtube.com
            /channel/
            UCvLZI6FixOkVVBxY2C2g2Cg"),
        "Diversas técnicas de fabricació
            n para obtener series cortas
            de piezas.");

    listaDeElementos.add(youtub003);

    Youtuber youtub004 = new Youtuber("Javier Garcia",
        new URL("https://www.youtube.com
            /channel/
            UCYOv9HwOFwK01Y2dUQ1ZSpg"),
        "Charlas sobre diversos temas de
            física.");

    listaDeElementos.add(youtub004);

}
catch (MalformedURLException e)
{
    // Por ahora las ignoramos; pero las excepciones, como mínimo,
    // se han de registrar en algún sitio donde se puedan
    // consultar luego.)
}

}

public ArrayList<Elemento> getDatos()
{
    return listaDeElementos;
}

}

```

Listing 17.39: 'Elemento.java'

```

package seleccionArborea;

public abstract class Elemento {

    protected String nombre;
    private String descripcion;

    public void setDescripcion(String descripcion)
    {
        this.descripcion = descripcion;
    }

    public String getDescripcion()
    {

```

```
        return descripcion;
    }

    @Override
    public String toString()
    {
        return nombre;
    }

    public abstract String getFichaDescriptiva();
}
```

Listing 17.40: 'Libro.java'

```
package seleccionArborea;

import java.time.LocalDate;

public class Libro extends Elemento {

    private String autor;
    private String editorial;
    private LocalDate fechaDePublicacion;

    public Libro(String titulo, String autor, String editorial,
        LocalDate fechaDePublicacion)
    {
        this.nombre = titulo;
        this.autor = autor;
        this.editorial = editorial;
        this.fechaDePublicacion = fechaDePublicacion;
    }

    @Override
    public String getFichaDescriptiva() {
        return "'" + nombre + "'" + System.lineSeparator()
            + autor + System.lineSeparator()
            + "(" + fechaDePublicacion.getYear() + ") " + editorial +
                System.lineSeparator()
            + System.lineSeparator() + System.lineSeparator()
            + getDescripcion();
    }
}
```

Listing 17.41: 'Revista.java'

```
package seleccionArborea;

import java.net.URL;

public class Revista extends Elemento {

    private URL url;
```

```

    public Revista(String titulo, URL url)
    {
        this.nombre = titulo;
        this.url = url;
    }

    @Override
    public String getFichaDescriptiva() {
        return "/" + nombre + "/" + System.lineSeparator()
            + url.toString() + System.lineSeparator()
            + System.lineSeparator() + System.lineSeparator()
            + getDescripcion();
    }
}

```

Listing 17.42: 'Video.java'

```

package seleccionArborea;

import java.net.URL;

public class Video extends Elemento {

    private URL url;
    private Integer duracionEnMinutos;

    public Video(String titulo, URL url, Integer duracionEnMinutos)
    {
        this.nombre = titulo;
        this.url = url;
        this.duracionEnMinutos = duracionEnMinutos;
    }

    @Override
    public String getFichaDescriptiva() {
        return nombre + System.lineSeparator()
            + url.toString() + System.lineSeparator()
            + "(" + duracionEnMinutos + " min)" + System.
                lineSeparator()
            + System.lineSeparator() + System.lineSeparator()
            + getDescripcion();
    }
}

```

Listing 17.43: 'Youtuber.java'

```

package seleccionArborea;

import java.net.URL;

```

```
public class Youtuber extends Elemento {

    private URL url;
    private String principalTematicaQueSueleTratar;

    public Youtuber(String titulo, URL url, String
        principalTematicaQueSueleTratar)
    {
        this.nombre = titulo;
        this.url = url;
        this.principalTematicaQueSueleTratar =
            principalTematicaQueSueleTratar;
    }

    @Override
    public String getFichaDescriptiva() {
        return nombre + System.lineSeparator()
            + url.toString() + System.lineSeparator()
            + System.lineSeparator() + System.lineSeparator()
            + getDescripcion();
    }

}
```

Listing 17.44: 'Main.java'

```
package seleccionArborea;

import seleccionArborea.InterfazGraficoDeUsuario;

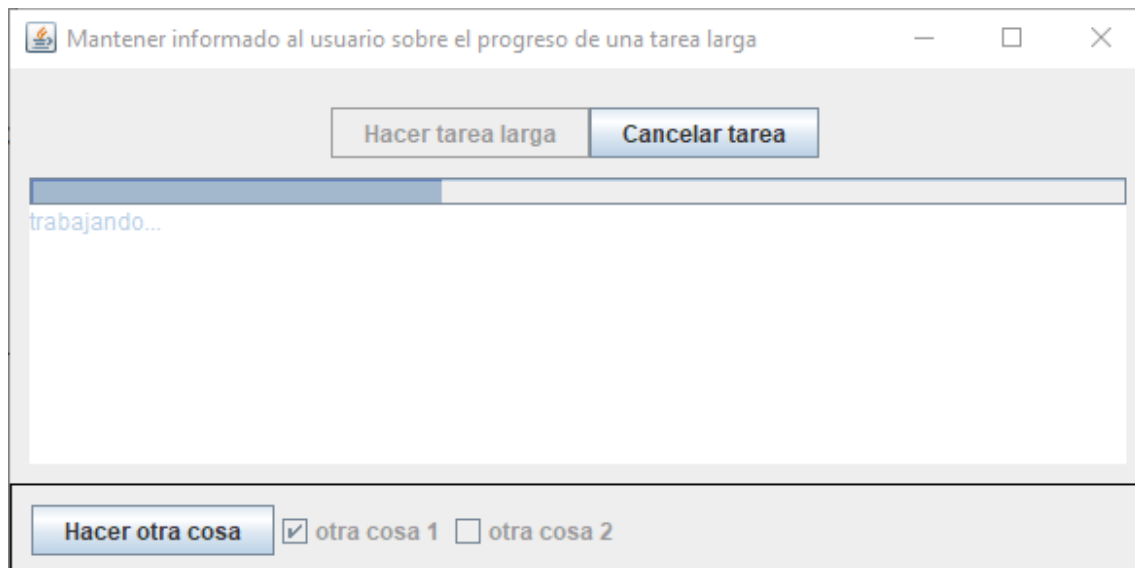
public class Main {

    public static void main(String[] args) {
        InterfazGraficoDeUsuario interfaz = new InterfazGraficoDeUsuario
            ();
        interfaz.mostrarInterface();
    }

}
```

3.9{Barra de progreso}

En lenguaje Java



Listing 17.45: 'module-info.java'

```
module BarraDeProgreso {
    requires java.desktop;
}
```

Listing 17.46: 'SimulacionDeTareaLarga.java'

```
package barraDeProgreso;

import javax.swing.SwingWorker;

public class SimulacionDeTareaLarga extends SwingWorker<String,
    Integer> {

    private int numeroDePasos;

    public SimulacionDeTareaLarga(Integer numeroDePasos) {
        if (numeroDePasos > 0) {
            this.numeroDePasos = numeroDePasos;
        } else {
            throw new IllegalArgumentException("La tarea ha de tener un nú
                mero positivo de pasos.");
        }
    }

    @Override
    protected String doInBackground() throws InterruptedException {
        long avancePrevio = 0;
        Integer pasoEnCurso = 0;
        for (; pasoEnCurso <= numeroDePasos; pasoEnCurso++) {
```

```

        if(this.isCancelled()) {
            return null;
        }
        long avance = Math.round(pasoEnCurso*1.0/numeroDePasos*100);
        // el progreso ha de ir en el rango de 0 a 100
        if (avance > avancePrevio) {
            if(avance > 0 && avance < 101 ) {
                setProgress((int) avance);
            }
            avancePrevio = avance;
        }

        Thread.sleep(10);
    }
    return "La respuesta es... 42";
}

@Override
protected void done() {
    //aquí va lo que sea necesario hacer al terminar o al cancelarse
    el trabajo.
}

}

```

Listing 17.47: 'InterfazGraficoDeUsuario.java'

```

package barraDeProgreso;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.util.concurrent.ExecutionException;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.JTextArea;
import javax.swing.SwingWorker.StateValue;
import javax.swing.WindowConstants;

public class InterfazGraficoDeUsuario {

    private JFrame cuadroGeneral;

```

```

private Integer NUMERODEPASOSENLATAREALARGA = 300;
Boolean seHaMandadoCancelarLaTarea;

public InterfazGraficoDeUsuario() {
    cuadroGeneral = new JFrame();
    cuadroGeneral.setDefaultCloseOperation(WindowConstants.
        EXIT_ON_CLOSE);
    cuadroGeneral.setTitle("Mantener informado al usuario sobre el
        progreso de una tarea larga");
    cuadroGeneral.getContentPane().setLayout(new BorderLayout());
    cuadroGeneral.setSize(600, 300);

    JProgressBar barraDeProgreso = new JProgressBar();
    JTextArea txtResultadoDeLaTarea = new JTextArea();
    txtResultadoDeLaTarea.setEnabled(false);

    JButton btnCancelarTarea = new JButton("Cancelar tarea");

    JButton btnArrancarTarea = new JButton("Hacer tarea larga");
    btnArrancarTarea.setSize(200, 50);
    btnArrancarTarea.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            seHaMandadoCancelarLaTarea = false;
            barraDeProgreso.setValue(0);
            btnArrancarTarea.setEnabled(false);
            btnCancelarTarea.setEnabled(true);
            txtResultadoDeLaTarea.setText("trabajando...");
            //nota: Los SwingWorker son de un solo uso,
            //      para poderlo lanzar cada vez que se clic el boton
            //      se ha de crear uno nuevo y ejecutarlo dentro de la
            //      misma funcion.
            SimulacionDeTareaLarga tarea = new SimulacionDeTareaLarga(
                NUMERODEPASOSENLATAREALARGA);
            tarea.addPropertyChangeListener(new PropertyChangeListener()
                {
                    @Override
                    public void propertyChange(PropertyChangeEvent evt) {
                        if (seHaMandadoCancelarLaTarea) {
                            tarea.cancel(false); //false = cancelar sin
                                interrupciones bruscas ; true = cancelar
                                interrumpiendo el trabajo
                        }
                        if ("progress" == evt.getPropertyName()) {
                            barraDeProgreso.setValue((Integer) evt.getNewValue());
                        }
                        if ("state" == evt.getPropertyName()) {
                            Object estado = evt.getNewValue();
                            if (estado.equals(StateValue.DONE)) {
                                try {
                                    txtResultadoDeLaTarea.setText("La tarea ha dicho:
                                        [" + tarea.get() + "]);
                                } catch (InterruptedException|java.util.concurrent.
```

```

        CancellationExcepcion e) {
            txtResultadoDeLaTarea.setText("La tarea ha tenido
                una excepcion que la ha interrumpido.");
        } catch (ExecutionException e) {
            txtResultadoDeLaTarea.setText("La tarea ha tenido
                algún error de ejecución.");
        }
        barraDeProgreso.setValue(0);
        btnArrancarTarea.setEnabled(true);
        btnCancelarTarea.setEnabled(false);
    }
}
});
tarea.execute();
}

});

btnCancelarTarea.setSize(200, 50);
btnCancelarTarea.setEnabled(false);
btnCancelarTarea.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        seHaMandadoCancelarLaTarea = true;
        txtResultadoDeLaTarea.setText("La tarea se ha interrumpido."
            );
        btnArrancarTarea.setEnabled(true);
        btnCancelarTarea.setEnabled(false);
    }
});

JPanel cuadroBotonesTarea = new JPanel();
cuadroBotonesTarea.setLayout(new BoxLayout(cuadroBotonesTarea,
    BoxLayout.X_AXIS));
cuadroBotonesTarea.setBorder(BorderFactory.createEmptyBorder(10,
    10, 10, 10));
cuadroBotonesTarea.add(btnArrancarTarea);
cuadroBotonesTarea.add(btnCancelarTarea);

JPanel cuadroCentral = new JPanel();
cuadroCentral.setLayout(new BoxLayout(cuadroCentral, BoxLayout.
    Y_AXIS));
cuadroCentral.setBorder(BorderFactory.createEmptyBorder(10, 10,
    10, 10));
cuadroCentral.add(cuadroBotonesTarea);
cuadroCentral.add(barraDeProgreso);
cuadroCentral.add(txtResultadoDeLaTarea);
cuadroGeneral.add(cuadroCentral, BorderLayout.CENTER);

JPanel cuadroHacerOtraCosa = new JPanel();
cuadroHacerOtraCosa.setLayout(new BoxLayout(cuadroHacerOtraCosa,
    BoxLayout.X_AXIS));

```



```

        cuadroHacerOtraCosa.setBorder(BorderFactory.createCompoundBorder
            (BorderFactory.createLineBorder(Color.BLACK), BorderFactory.
                createEmptyBorder(10, 10, 10, 10)));
        JCheckBox otracosa1 = new JCheckBox("otra cosa 1");
        otracosa1.setEnabled(false);
        JCheckBox otracosa2 = new JCheckBox("otra cosa 2");
        otracosa2.setEnabled(false);
        JButton btnHacerOtraCosa = new JButton("Hacer otra cosa");
        btnHacerOtraCosa.setSize(200, 50);
        btnHacerOtraCosa.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(otracosa1.isSelected() && otracosa2.isSelected()) {
                    otracosa1.setSelected(false);
                    otracosa2.setSelected(false);
                }
                else if(otracosa1.isSelected()) {
                    otracosa2.setSelected(true);
                }
                else {
                    otracosa1.setSelected(true);
                }
            }
        });
        cuadroHacerOtraCosa.add(btnHacerOtraCosa);
        cuadroHacerOtraCosa.add(otracosa1);
        cuadroHacerOtraCosa.add(otracosa2);
        cuadroGeneral.add(cuadroHacerOtraCosa, BorderLayout.SOUTH);
    }

    public void mostrarInterface() {
        cuadroGeneral.setVisible(true);
    }
}

```

Listing 17.48: 'Main.java'

```

package barraDeProgreso;

import javax.swing.SwingUtilities;

import barraDeProgreso.InterfazGraficoDeUsuario;

public class Main {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                InterfazGraficoDeUsuario interfaz = new
                    InterfazGraficoDeUsuario();
                interfaz.mostrarInterface();
            }
        });
    }
}

```

```
        });  
    }  
  
}
```

17.2. ...a los ejercicios del capítulo 5

((una disculpa del autor: este capítulo esta “en construcción...” y sigo trabajando en él; queda mucho trabajo para completarlo.))

5.2 {Ascensores}

+

Ascensor
+plantaMasBaja: int +plantaMasAlta: int +plantaDondeEstaAhora: int
+Ascensor(cantidadDePlantasPorEncimaDeCero: Integer, cantidadDePlantasPorDebajoDeCero: Integer) +IrALaPlanta(planta: Integer) +getPlantaDondeEstaAhora() +toString()

Edificio
+ascensores: java.util.ArrayList<Ascensor>
+java.util.ArrayList<Ascensor>(cantidadDePisosPorEncimaDeLaPlantaBaja:int, cantidadDeSotanosPorDebajoDeLaPlantaBaja:int, cantidadDeAscensores:int)
+EnviarUnAscensorAlPiso(piso: Integer) +MoverAscensor(ascensor: Integer, piso: Integer) +getSituacionDeLosAscensores()

InterfaceDeUsuarioTextual
+pisos: Integer +sotanos: Integer +ascensores: Integer +miCasa: Edificio
+run() +getDescripcionDeLaSituacionDeLosAscensores()

+

En lenguaje Java

Listing 17.49: 'module-info.java'

```
module ascensores {  
    requires java.desktop;  
    requires junit;  
}
```

Listing 17.50: 'Ascensor.java'

```
package ascensores;  
  
public class Ascensor {  
  
    private int plantaMasBaja;  
    private int plantaMasAlta;  
    private int plantaDondeEstaAhora;  
  
    public Ascensor(Integer cantidadDePlantasPorEncimaDeCero,  
                    Integer cantidadDePlantasPorDebajoDeCero)  
    {  
        this.plantaDondeEstaAhora = 0;  
        if (cantidadDePlantasPorEncimaDeCero > 0  
            && cantidadDePlantasPorDebajoDeCero >= 0)  
        {  
            this.plantaMasBaja = cantidadDePlantasPorDebajoDeCero * -1;  
            this.plantaMasAlta = cantidadDePlantasPorEncimaDeCero;  
        }  
        else  
        {  
            throw new java.lang.IllegalArgumentException();  
        }  
    }  
  
    public void IrALaPlanta(Integer planta)  
    {  
        if (planta >= this.plantaMasBaja && planta <= this.plantaMasAlta  
            )  
        {  
            this.plantaDondeEstaAhora = planta;  
        }  
    }  
  
    public Integer getPlantaDondeEstaAhora()  
    {  
        return this.plantaDondeEstaAhora;  
    }  
  
    public String toString()  
    {  
        return "Este ascensor puede ir desde la planta " + this.  
            plantaMasBaja  
            + " hasta la planta " + this.plantaMasAlta  
            + System.lineSeparator()  
            + "Ahora esta en la planta " + this.getPlantaDondeEstaAhora  
            ();  
    }  
}
```

```
}  
  
}
```

Listing 17.51: 'AscensorTests.java'

```
package ascensores;  
  
import org.junit.Test;  
import static org.junit.Assert.assertEquals;  
import static org.junit.Assert.assertThrows;  
  
public class AscensorTests {  
  
    @Test  
    void Comprobar_situaciones_excepcionales()  
    {  
        assertThrows(java.lang.IllegalArgumentException.class,  
            () -> {Ascensor prueba = new Ascensor(-2,  
                2);} );  
        assertThrows(java.lang.IllegalArgumentException.class,  
            () -> {Ascensor prueba = new Ascensor(2,  
                -2);} );  
    }  
  
    @Test  
    void Comprobar_operaciones_normales()  
    {  
        Ascensor ascensorDePruebas = new Ascensor(2,2);  
        assertEquals((Integer) 0, ascensorDePruebas.  
            getPlantaDondeEstaAhora());  
  
        ascensorDePruebas.IrALaPlanta(1);  
        assertEquals((Integer) 1, ascensorDePruebas.  
            getPlantaDondeEstaAhora());  
  
        ascensorDePruebas.IrALaPlanta(-1);  
        assertEquals((Integer) (-1), ascensorDePruebas.  
            getPlantaDondeEstaAhora());  
    }  
  
    @Test  
    void Comprobar_operaciones_anomalas()  
    {  
        Ascensor ascensorDePruebas = new Ascensor(2,2);  
        ascensorDePruebas.IrALaPlanta(333);  
        assertEquals((Integer) 0, ascensorDePruebas.  
            getPlantaDondeEstaAhora());  
    }  
  
}
```

Listing 17.52: 'Edificio.java'

```

package ascensores;

public class Edificio {

    private java.util.ArrayList<Ascensor> ascensores;

    public Edificio(int cantidadDePisosPorEncimaDeLaPlantaBaja,
                    int cantidadDeSotanosPorDebajoDeLaPlantaBaja,
                    int cantidadDeAscensores)
    {
        if(cantidadDeAscensores > 0)
        {
            ascensores = new java.util.ArrayList<Ascensor>()
                ;
            for (int i = 0; i < cantidadDeAscensores ; i++)
            {
                ascensores.add(new Ascensor(
                    cantidadDePisosPorEncimaDeLaPlantaBaja,
                    cantidadDeSotanosPorDebajoDeLaPlantaBaja,
                    i));
            }
        }
        else
        {
            throw new java.lang.IllegalArgumentException();
        }
    }

    public void EnviarUnAscensorAlPiso(Integer piso)
    {
        Ascensor ascensorMasCercano = ascensores.get(0);
        Integer menorDistanciaEncontradaHastaAhora =
            Math.abs(piso - ascensorMasCercano.
                getPlantaDondeEstaAhora());
        for (Ascensor ascensor : ascensores)
        {
            if(Math.abs(piso - ascensor.
                getPlantaDondeEstaAhora()) <
                menorDistanciaEncontradaHastaAhora)
            {
                menorDistanciaEncontradaHastaAhora =
                    Math.abs(piso - ascensor.
                        getPlantaDondeEstaAhora());
                ascensorMasCercano = ascensor;
            }
        }
        ascensorMasCercano.IrALaPlanta(piso);
    }

    public void MoverAscensor(Integer ascensor, Integer piso)
    {
        if(ascensor < ascensores.size())
    
```

```

        {
            ascensores.get(ascensor).IrALaPlanta(piso);
        }
    }

    public java.util.ArrayList<Integer> getSituacionDeLosAscensores
        ()
    {
        java.util.ArrayList<Integer> posiciones = new java.util.
            ArrayList<Integer>();
        for (Ascensor ascensor : ascensores)
        {
            posiciones.add(ascensor.getPlantaDondeEstaAhora
                ());
        }
        return posiciones;
    }
}

```

Listing 17.53: 'EdificioTests.java'

```

package ascensores;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertThrows;

import org.junit.Test;

public class EdificioTests {

    @Test
    void Comprobar_situaciones_excepcionales()
    {
        assertThrows(java.lang.IllegalArgumentException.class,
            () -> {Edificio prueba = new
                Edificio(2, 2, 0);} );
        assertThrows(java.lang.IllegalArgumentException.class,
            () -> {Edificio prueba = new Edificio
                (2, 2, -2);} );
    }

    @Test
    void Comprobar_operaciones_normales()
    {
        Edificio casa = new Edificio(2, 2, 2);
        java.util.ArrayList<Integer> posiciones00 = casa.
            getSituacionDeLosAscensores();
        assertEquals((Integer) 0, posiciones00.get(0));
        assertEquals((Integer) 0, posiciones00.get(1));

        casa.MoverAscensor(1, 1);
        java.util.ArrayList<Integer> posiciones01 = casa.
            getSituacionDeLosAscensores();
        assertEquals((Integer) 0, posiciones01.get(0));
    }
}

```

```

        assertEquals((Integer) (1), posiciones01.get(1));

        casa.EnviaUnAscensorAlPiso(-1);
        java.util.ArrayList<Integer> posiciones02 = casa.
            getSituacionDeLosAscensores();
        assertEquals((Integer) (-1), posiciones02.get(0));
        assertEquals((Integer) (1), posiciones02.get(1));
    }

    @Test
    void Comprobar_operaciones_anomalias()
    {
        Edificio casa = new Edificio(2, 2, 2);
        casa.MoverAscensor(9, 1);
        java.util.ArrayList<Integer> posiciones03 = casa.
            getSituacionDeLosAscensores();
        assertEquals((Integer) (0), posiciones03.get(0));
        assertEquals((Integer) (0), posiciones03.get(1));

        casa.MoverAscensor(1, 9);
        java.util.ArrayList<Integer> posiciones04 = casa.
            getSituacionDeLosAscensores();
        assertEquals((Integer) (0), posiciones04.get(0));
        assertEquals((Integer) (0), posiciones04.get(1));

        casa.EnviaUnAscensorAlPiso(9);
        java.util.ArrayList<Integer> posiciones05 = casa.
            getSituacionDeLosAscensores();
        assertEquals((Integer) (0), posiciones05.get(0));
        assertEquals((Integer) (0), posiciones05.get(1));
    }
}

```

```

Tenemos un edificio con 3 ascensores
Que pueden ir desde la planta -2 a la planta +4

Situacion actual de los ascensores: [0] [0] [0]

¿Que desea hacer?
-0- Terminar y salir.
-1- Mover un ascensor concreto a un piso concreto.
-2- Enviar un ascensor a un piso.
1
¿Que ascensor quieres mover? (1...3)
2
¿A que piso? (-2...+4)
4

Tenemos un edificio con 3 ascensores
Que pueden ir desde la planta -2 a la planta +4

Situacion actual de los ascensores: [0] [4] [0]

¿Que desea hacer?
-0- Terminar y salir

```

```

Tenemos un edificio con 3 ascensores
Que pueden ir desde la planta -2 a la planta +4

Situacion actual de los ascensores: [0] [4] [0]

¿Que desea hacer?
-0- Terminar y salir.
-1- Mover un ascensor concreto a un piso concreto.
-2- Enviar un ascensor a un piso.
2
¿A que piso? (-2...+4)
1

Tenemos un edificio con 3 ascensores
Que pueden ir desde la planta -2 a la planta +4

Situacion actual de los ascensores: [1] [4] [0]

¿Que desea hacer?
-0- Terminar y salir

```

Listing 17.54: 'InterfazTextualDeUsuario.java'

```

package ascensores;

public class InterfazTextualDeUsuario {

```

```
private Integer pisos;
private Integer sotanos;
private Integer ascensores;
private Edificio miCasa;

public InterfazTextualDeUsuario()
{
    pisos = 4;
    sotanos = 2;
    ascensores = 3;
    miCasa = new Edificio(pisos, sotanos, ascensores);
}

public void run()
{
    java.util.Scanner lectorDeTeclado = new java.util.
        Scanner(System.in);
    while(true)
    {
        System.out.println();
        System.out.println("Tenemos un edificio con " +
            ascensores + " ascensores");
        System.out.println("Que pueden ir desde la
            planta -" + sotanos + " a la planta +" +
            pisos);
        System.out.println();
        System.out.println(
            getDescripcionDeLaSituacionDeLosAscensores(
                miCasa.getSituacionDeLosAscensores()));
        System.out.println();
        System.out.println("?Que desea hacer?");
        System.out.println("-0- Terminar y salir.");
        System.out.println("-1- Mover un ascensor
            concreto a un piso concreto.");
        System.out.println("-2- Enviar un ascensor a un
            piso.");
        Integer accion = lectorDeTeclado.nextInt();

        switch(accion)
        {
            case 0:
                lectorDeTeclado.close();
                System.exit(0);
                break;
            case 1:
                System.out.println("?Que ascensor
                    quieres mover? (1..." + ascensores +
                    ")");
                Integer ascensor = lectorDeTeclado.
                    nextInt();
                ascensor = ascensor - 1;
                System.out.println("?A que piso? (-" +
                    sotanos + "...+" + pisos + ")");
                Integer plantaDestino = lectorDeTeclado.
                    nextInt();
```

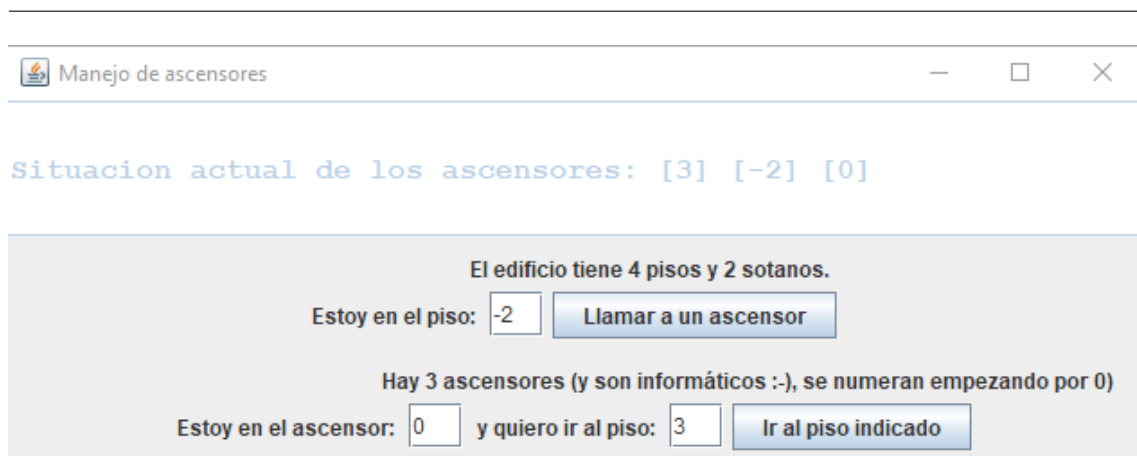


```

        miCasa.MoverAscensor(ascensor,
            plantaDestino);
        break;
    case 2:
        System.out.println("?A que piso? (-" +
            sotanos + "...+" + pisos + ")");
        Integer plantaLlamada = lectorDeTeclado.
            nextInt();
        miCasa.EnviaUnAscensorAlPiso(
            plantaLlamada);
        break;
    }
}

String getDescripcionDeLaSituacionDeLosAscensores(java.util.
    ArrayList<Integer> posicionesDeLosAscensores)
{
    String texto = "Situacion actual de los ascensores: ";
    for (Integer posicion : posicionesDeLosAscensores)
    {
        texto = texto + "[" + posicion + "] ";
    }
    return texto;
}
}

```



Listing 17.55: 'InterfazGraficoDeUsuario.java'

```

package ascensores;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class InterfazGraficoDeUsuario {

    private JFrame cuadroGeneral;

```

```

private int pisos = 4;
private int sotanos = 2;
private int ascensores = 3;
private Edificio miCasa = new Edificio(pisos, sotanos, ascensores)
    ;

public InterfazGraficoDeUsuario()
{
    cuadroGeneral = new JFrame();
    cuadroGeneral.setDefaultCloseOperation(WindowConstants.
        EXIT_ON_CLOSE);
    cuadroGeneral.setTitle("Manejo de ascensores");
    cuadroGeneral.getContentPane().setLayout(new BorderLayout(10,10)
        );

    JTextField estadoDeLosAscensores = new JTextField();
    estadoDeLosAscensores.setPreferredSize(new Dimension(200, 75));
    estadoDeLosAscensores.setEnabled(false);
    estadoDeLosAscensores.setFont(new Font("Monospaced", Font.BOLD,
        16));
    estadoDeLosAscensores.setText(
        getDescripcionDeLaSituacionDeLosAscensores(miCasa.
            getSituacionDeLosAscensores()));
    cuadroGeneral.add(estadoDeLosAscensores, BorderLayout.NORTH);

    JPanel cuadroBotoneraDePiso = new JPanel();
    cuadroBotoneraDePiso.setLayout(new BoxLayout(
        cuadroBotoneraDePiso, BoxLayout.PAGE_AXIS));

    JLabel descripcionDelNumeroDePisos = new JLabel("El edificio
        tiene " + pisos + " pisos y " + sotanos + " sotanos.");
    cuadroBotoneraDePiso.add(descripcionDelNumeroDePisos);

    JPanel subCuadroBotoneraDePiso = new JPanel();

    JLabel pisoEnElQueEstas_lbl = new JLabel();
    pisoEnElQueEstas_lbl.setText("Estoy en el piso: ");
    JTextField pisoEnQueEstas_txt = new JTextField();
    pisoEnQueEstas_txt.setPreferredSize(new Dimension(30,25));

    JButton llamarAlAscensor = new JButton("Llamar a un ascensor");
    llamarAlAscensor.setSize(50, 50);
    llamarAlAscensor.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            miCasa.EnviarUnAscensorAlPiso(Integer.parseInt(
                pisoEnQueEstas_txt.getText()));
            estadoDeLosAscensores.setText(
                getDescripcionDeLaSituacionDeLosAscensores(miCasa.
                    getSituacionDeLosAscensores()));
        }
    });

    subCuadroBotoneraDePiso.add(pisoEnElQueEstas_lbl);

```

```

subCuadroBotoneraDePiso.add(pisoEnQueEstas_txt);
subCuadroBotoneraDePiso.add(llamarAlAscensor);
cuadroBotoneraDePiso.add(subCuadroBotoneraDePiso);
cuadroGeneral.add(cuadroBotoneraDePiso, BorderLayout.CENTER);

JPanel cuadroBotoneraDeAscensor = new JPanel();
cuadroBotoneraDeAscensor.setLayout(new BoxLayout(
    cuadroBotoneraDeAscensor, BoxLayout.PAGE_AXIS));

JLabel descripcionDelNumeroDeAscensores = new JLabel("Hay " +
    ascensores + " ascensores (y son informáticos :-), se
    numeran empezando por 0)");
cuadroBotoneraDeAscensor.add(descripcionDelNumeroDeAscensores);

JPanel subCuadroBotoneraDeAscensor = new JPanel();

JLabel ascensorEnElQueEstas_lbl = new JLabel();
ascensorEnElQueEstas_lbl.setText("Estoy en el ascensor: ");
JTextField ascensorEnElQueEstas_txt = new JTextField();
ascensorEnElQueEstas_txt.setPreferredSize(new Dimension(30,25));

JLabel pisoAlQueQuieresIr_lbl = new JLabel();
pisoAlQueQuieresIr_lbl.setText(" y quiero ir al piso: ");
JTextField pisoAlQueQuieresIr_txt = new JTextField();
pisoAlQueQuieresIr_txt.setPreferredSize(new Dimension(30,25));

JButton irAlPisoIndicado = new JButton("Ir al piso indicado");
irAlPisoIndicado.setSize(50, 50);
irAlPisoIndicado.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        miCasa.MoverAscensor(Integer.parseInt(
            ascensorEnElQueEstas_txt.getText()), Integer.
            parseInt(pisoAlQueQuieresIr_txt.getText()));
        estadoDeLosAscensores.setText(
            getDescripcionDeLaSituacionDeLosAscensores(miCasa.
            getSituacionDeLosAscensores()));
    }
});

subCuadroBotoneraDeAscensor.add(ascensorEnElQueEstas_lbl);
subCuadroBotoneraDeAscensor.add(ascensorEnElQueEstas_txt);
subCuadroBotoneraDeAscensor.add(pisoAlQueQuieresIr_lbl);
subCuadroBotoneraDeAscensor.add(pisoAlQueQuieresIr_txt);
subCuadroBotoneraDeAscensor.add(irAlPisoIndicado);
cuadroBotoneraDeAscensor.add(subCuadroBotoneraDeAscensor);
cuadroGeneral.add(cuadroBotoneraDeAscensor, BorderLayout.SOUTH);

cuadroGeneral.pack();
}

public void run()
{

```

```
        cuadroGeneral.setVisible(true);
    }

    private String getDescripcionDeLaSituacionDeLosAscensores(java.
        util.ArrayList<Integer> posicionesDeLosAscensores)
    {
        String texto = "Situacion actual de los ascensores: ";
        for (Integer posicion : posicionesDeLosAscensores)
        {
            texto = texto + "[" + posicion + "]" + " ";
        }
        return texto;
    }
}
```

Listing 17.56: 'Main.java'

```
package ascensores;

public class Main {

    public static void main(String[] args) {

        //InterfazTextualDeUsuario interfaz = new
        //InterfazTextualDeUsuario();
        InterfazGraficoDeUsuario interfaz = new InterfazGraficoDeUsuario
        ();

        interfaz.run();
    }

}
```

17.3. ...a los ejercicios del capítulo 8

((una disculpa del autor: este capítulo no lo he abordado aún; el libro esta “en construcción...” y sigo trabajando en él; queda mucho trabajo para completarlo.))

17.4. ...a los ejercicios del capítulo 11

((una disculpa del autor: este capítulo no lo he abordado aún; el libro esta “en construcción...” y sigo trabajando en él; queda mucho trabajo para completarlo.))

17.5. ...a los ejercicios del capítulo 13

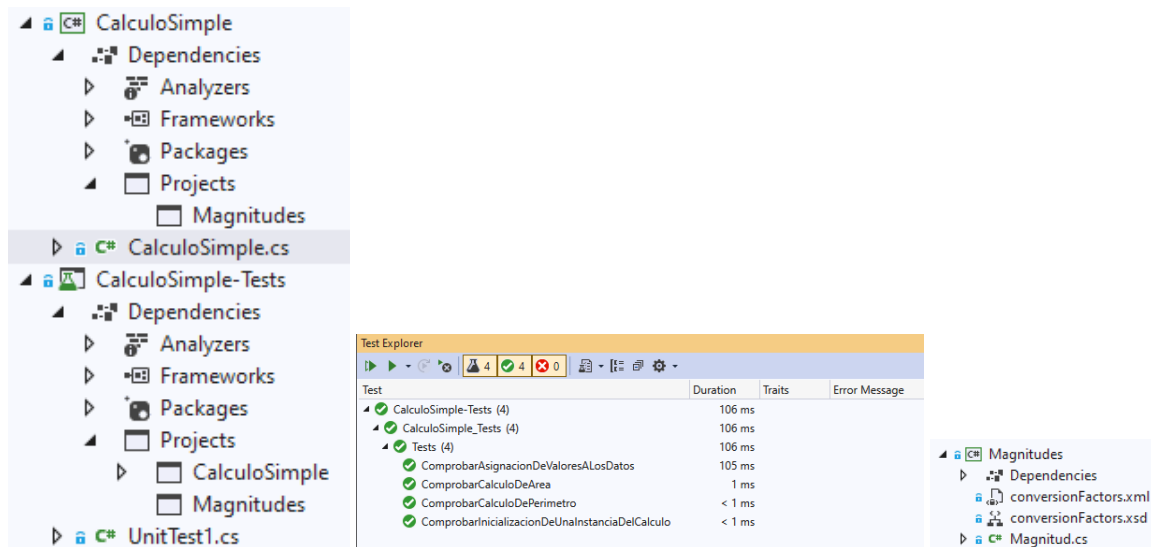
((una disculpa del autor: este capítulo no lo he abordado aún; el libro esta “en construcción...” y sigo trabajando en él; queda mucho trabajo para completarlo.))

17.6. ...a los ejercicios del capítulo 15

((una disculpa del autor: este capítulo no lo he abordado aún; el libro esta “en construcción...” y sigo trabajando en él; queda mucho trabajo para completarlo.))

15.1 {Multiinterface}

El programa base, que se va a utilizar desde diversos interfaces de usuario...



Listing 17.57: 'CalculoSimple.cs'

```
using System;

namespace CalculoSimple
{
    public class CalculoSimple
    {
        private Magnitudes.Magnitud unlado;
        private Magnitudes.Magnitud otrolado;

        private Magnitudes.Magnitud perimetro;
        private Magnitudes.Magnitud area;
        //private System.Drawing.Bitmap representacion;

        public CalculoSimple()
        {
            unlado = new Magnitudes.Magnitud();
            otrolado = new Magnitudes.Magnitud();

            perimetro = new Magnitudes.Magnitud();
            area = new Magnitudes.Magnitud();
            //representacion = Properties.Resources.NoRepresentado;
```

```
}

private void ReCalcular()
{
    if (!double.IsNaN(unlado.valor) && !double.IsNaN(
        otrolado.valor))
    {
        perimetro.valor = 2.0 * unlado.valor + 2.0 *
            otrolado.getCopiaConvertidaA(unlado.
                unidaddemedida).valor;
        perimetro.unidaddemedida = unlado.unidaddemedida;

        area.valor = unlado.valor * otrolado.
            getCopiaConvertidaA(unlado.unidaddemedida).valor
            ;
        area.unidaddemedida = unlado.unidaddemedida + "2";

        //representacion = DibujarElParalelepipedo();
    }
}

public void setDato_unlado(Magnitudes.Magnitud dato)
{
    unlado = dato;
    ReCalcular();
}

public void setDato_otrolado(Magnitudes.Magnitud dato)
{
    otrolado = dato;
    ReCalcular();
}

public Magnitudes.Magnitud getResultado_perimetro()
{
    return perimetro;
}

public Magnitudes.Magnitud getResultado_area()
{
    return area;
}

public Magnitudes.Magnitud getDato_unlado()
{
    return unlado;
}

public Magnitudes.Magnitud getDato_otrolado()
{
    return otrolado;
}

private System.Drawing.Bitmap DibujarElParalelepipedo()
```

```

        {
            String _pendiente_DibujarRealmenteElParalelepipedo;
            return new System.Drawing.Bitmap(100, 100);
        }

        //public System.Drawing.Bitmap getResultado_representacion()
        //{
        //    return representacion;
        //}

    }
}

```

Listing 17.58: 'UnitTest1.cs'

```

using NUnit.Framework;

namespace CalculoSimple_Tests
{
    public class Tests
    {
        [SetUp]
        public void Setup()
        {
        }

        [Test]
        public void ComprobarInicializacionDeUnaInstanciaDelCalculo
            ()
        {
            CalculoSimple.CalculoSimple calculadora = new
                CalculoSimple.CalculoSimple();

            Assert.IsTrue(double.IsNaN(calculadora.getDato_unlado().
                valor));
            Assert.AreEqual(Magnitudes.Magnitud.DESCONOCIDA,
                calculadora.getDato_unlado().unidadmedida);

            Assert.IsTrue(double.IsNaN(calculadora.getDato_otrolado
                ().valor));
            Assert.AreEqual(Magnitudes.Magnitud.DESCONOCIDA,
                calculadora.getDato_otrolado().unidadmedida);
        }

        [Test]
        public void ComprobarAsignacionDeValoresALosDatos()
        {
            CalculoSimple.CalculoSimple calculadora = new
                CalculoSimple.CalculoSimple();
        }
    }
}

```

```

        calculadora.setDato_unlado(new Magnitudes.Magnitud(valor
            : 2, unidadDeMedida: "m"));
        Assert.AreEqual(2.0, calculadora.getDato_unlado().valor)
        ;
        Assert.AreEqual("m", calculadora.getDato_unlado().
            unidaddemedida);

        calculadora.setDato_otrolado(new Magnitudes.Magnitud(
            valor: 300, unidadDeMedida: "cm"));
        Assert.AreEqual(300.0, calculadora.getDato_otrolado().
            valor);
        Assert.AreEqual("cm", calculadora.getDato_otrolado().
            unidaddemedida);
    }

```

```

[Test]
public void ComprobarCalculoDePerimetro()
{
    CalculoSimple.CalculoSimple calculadora = new
        CalculoSimple.CalculoSimple();

    Assert.IsTrue(double.IsNaN(calculadora.
        getResultado_perimetro().valor));
    Assert.AreEqual(Magnitudes.Magnitud.DESCONOCIDA,
        calculadora.getResultado_perimetro().unidaddemedida)
        ;

    calculadora.setDato_unlado(new Magnitudes.Magnitud(valor
        : 2, unidadDeMedida: "m"));
    Assert.IsTrue(double.IsNaN(calculadora.
        getResultado_perimetro().valor));
    Assert.AreEqual(Magnitudes.Magnitud.DESCONOCIDA,
        calculadora.getResultado_perimetro().unidaddemedida)
        ;

    calculadora.setDato_otrolado(new Magnitudes.Magnitud(
        valor: 300, unidadDeMedida: "cm"));
    Assert.AreEqual(10, calculadora.getResultado_perimetro()
        .valor);
    Assert.AreEqual("m", calculadora.getResultado_perimetro
        ().unidaddemedida);
}

```

```

[Test]
public void ComprobarCalculoDeArea()
{
    CalculoSimple.CalculoSimple calculadora = new
        CalculoSimple.CalculoSimple();

    Assert.IsTrue(double.IsNaN(calculadora.getResultado_area
        ().valor));
    Assert.AreEqual(Magnitudes.Magnitud.DESCONOCIDA,

```



```

        calculadora.getResultado_area().unidaddemedida);

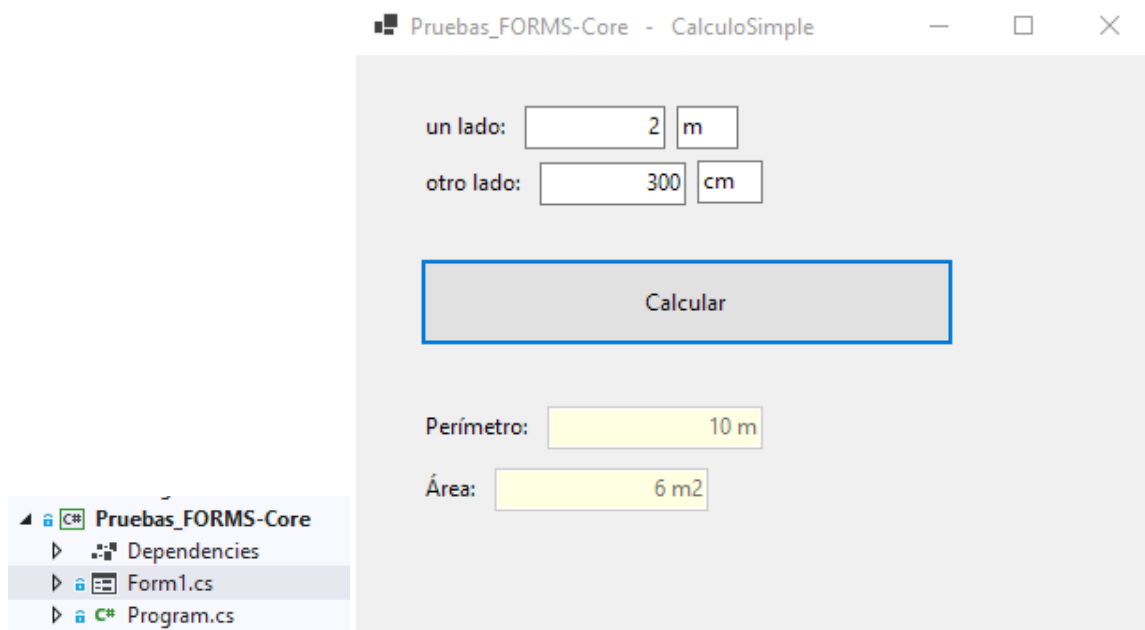
        calculadora.setDato_unlado(new Magnitudes.Magnitud(valor
            : 2, unidadDeMedida: "m"));
        Assert.IsTrue(double.IsNaN(calculadora.getResultado_area()
            .valor));
        Assert.AreEqual(Magnitudes.Magnitud.DESCONOCIDA,
            calculadora.getResultado_area().unidaddemedida);

        calculadora.setDato_otrolado(new Magnitudes.Magnitud(
            valor: 300, unidadDeMedida: "cm"));
        Assert.AreEqual(6, calculadora.getResultado_area().valor
            );
        Assert.AreEqual("m2", calculadora.getResultado_area().
            unidaddemedida);
    }

}

```

Utilizado desde un formulario Windows Forms



Listing 17.59: 'Form1.cs'

```

using System;
using System.Windows.Forms;

namespace Pruebas_FORMS_Core
{
    public partial class Form1 : Form
    {

```

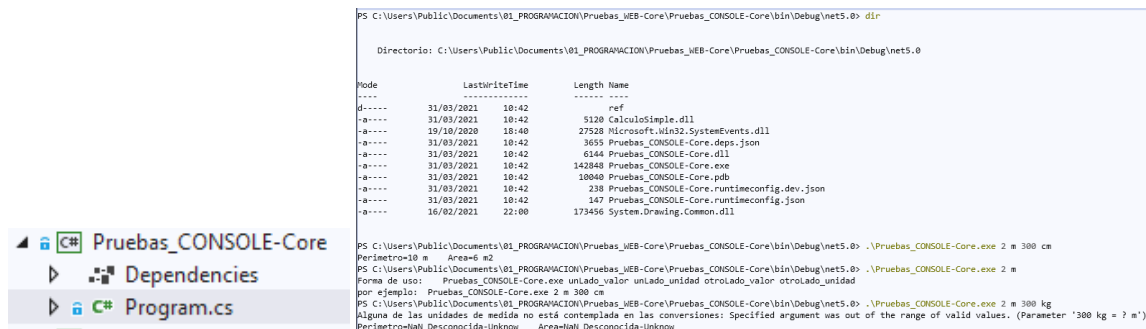
```
public Form1()
{
    InitializeComponent();
}

private void btnCalcular_Click(object sender, EventArgs e)
{
    CalculoSimple.CalculoSimple calculadora = new
        CalculoSimple.CalculoSimple();

    try
    {
        calculadora.setDato_unlado(new Magnitudes.Magnitud(
            Double.Parse(txtUnLado_valor.Text),
            txtUnLado_unidadmedida.Text));
        calculadora.setDato_otrolado(new Magnitudes.Magnitud(
            Double.Parse(txtOtroLado_valor.Text),
            txtOtroLado_unidadmedida.Text));
    }
    catch (ArgumentOutOfRangeException ex)
    {
        MessageBox.Show("Alguna de las unidades de medida no
            está contemplada en las conversiones: " + ex.
            Message);
    }
    catch (System.IO.FileNotFoundException ex)
    {
        MessageBox.Show("No se ha encontrado el archivo con
            la lista de conversiones entre unidades de
            medida: " + ex.Message);
    }
    catch (ArgumentException ex)
    {
        MessageBox.Show("Problemas leyendo el archivo con la
            lista de conversiones entre unidades de medida:
            " + ex.Message);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Problemas en alguna conversión
            entre unidades de medida: " + ex.Message);
    }

    txtPerimetro.Text = calculadora.getResultado_perimetro()
        .ToString();
    txtArea.Text = calculadora.getResultado_area().ToString
        ();
}
}
```

Utilizado desde linea de comandos



Listing 17.60: 'Program.cs'

```
using System;

namespace Pruebas_CONSOLE_Core
{
    class Program
    {
        static void Main(string[] args)
        {
            if (args.Length != 4)
            {
                Console.WriteLine("Forma de uso:    Pruebas_CONSOLE-
Core.exe unLado_valor unLado_unidad
otroLado_valor otroLado_unidad");
                Console.WriteLine("por ejemplo:  Pruebas_CONSOLE-
Core.exe 2 m 300 cm");
            }
            else
            {
                CalculoSimple.CalculoSimple calculadora = new
                CalculoSimple.CalculoSimple();

                try
                {
                    calculadora.setDato_unlado(new Magnitudes.
                    Magnitud(Double.Parse(args[0]), args[1]));
                    calculadora.setDato_otrolado(new Magnitudes.
                    Magnitud(Double.Parse(args[2]), args[3]));
                }
                catch (ArgumentOutOfRangeException ex)
                {
                    Console.WriteLine("Alguna de las unidades de
                    medida no está contemplada en las
                    conversiones: " + ex.Message);
                }
                catch (System.IO.FileNotFoundException ex)
                {
                    Console.WriteLine("No se ha encontrado el
                    archivo con la lista de conversiones entre
                    unidades de medida: " + ex.Message);
                }
                catch (ArgumentException ex)
                {

```

```

        Console.WriteLine("Problemas leyendo el archivo
                           con la lista de conversiones entre unidades
                           de medida: " + ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Problemas en alguna conversi
                           ón entre unidades de medida: " + ex.Message);
    }

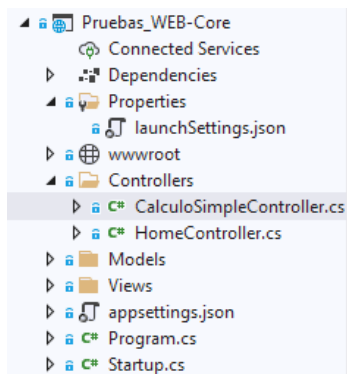
    Console.WriteLine("Perimetro=" + calculadora.
        getResultado_perimetro().ToString()
        + "      Area=" + calculadora.
        getResultado_area().ToString()
        );

    }

    }
}

```

Utilizado para dar unos servicios web RESTfull



Listing 17.61: 'CalculoSimpleController.cs'

```

using Microsoft.AspNetCore.Mvc;

namespace Pruebas_WEB_Core.Controllers
{
    public class ParametrosDelCalculoSimple
    {
        public Magnitudes.Magnitud unLado { get; set; }
        public Magnitudes.Magnitud otroLado { get; set; }
    }

    public class ResultadosDelCalculoSimple
    {
        public Magnitudes.Magnitud perimetro { get; set; }
        public Magnitudes.Magnitud area { get; set; }
    }
}

```

```

}

[ApiController]
[Route("api/[controller]")]
public class CalculoSimpleController : ControllerBase
{
    // nota: por URL, los parametros se pasan algo así como:
    // https://localhost:44310/api/CalculoSimple?unLado_valor=2&
    // unLado_unidaddemedida=m&otroLado_valor=300&
    // otroLado_unidaddemedida=cm

    [HttpGet]
    public ResultadosDelCalculoSimple Get(double unLado_valor,
        string unLado_unidaddemedida, double otroLado_valor,
        string otroLado_unidaddemedida)
    {
        CalculoSimple.CalculoSimple calculadora = new
            CalculoSimple.CalculoSimple();

        calculadora.setDato_unlado(new Magnitudes.Magnitud(
            unLado_valor, unLado_unidaddemedida));
        calculadora.setDato_otrolado(new Magnitudes.Magnitud(
            otroLado_valor, otroLado_unidaddemedida));

        ResultadosDelCalculoSimple resultados = new
            ResultadosDelCalculoSimple();
        resultados.perimetro = calculadora.
            getResultado_perimetro();
        resultados.area = calculadora.getResultado_area();
        return resultados;
    }

    // nota: en JSON el resultado se devuelve algo así como:
    // {
    //     "perimetro": {
    //         "valor": xxxxxx
    //         , "unidaddemedida": "xxxxxxx"
    //     }
    //     , "area": {
    //         "valor": xxxxxx
    //         , "unidaddemedida": "xxxxxxx"
    //     }
    // }
    //Escribiendolo en forma compacta: { "perimetro":{ "valor
    //":10,"unidaddemedida":"m"},"area":{ "valor":6,"
    //unidaddemedida":"m2"} }

    // nota: en JSON hay que pasar los parametros algo asi como:
    // {
    //     "unLado": {
    //         "valor": xxxxxx
    //         , "unidaddemedida":"xxxxxxx"

```

```
//
//      }
//      , "otroLado": {
//          "valor": xxxxxx
//          , "unidaddemedida": "xxxxxxxx"
//      }
//  }
//Escribiendolo en forma compacta: {"unLado":{"valor":2,"
//    unidaddemedida":"m"},"otroLado":{"valor":300,"
//    unidaddemedida":"cm"}}

[HttpPost]
public ResultadosDelCalculoSimple Post([FromBody]
    ParametrosDelCalculoSimple parametros)
{
    CalculoSimple.CalculoSimple calculadora = new
        CalculoSimple.CalculoSimple();

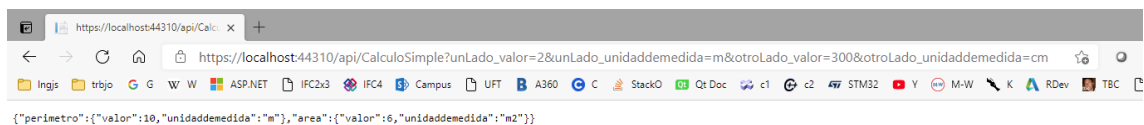
    calculadora.setDato_unlado(parametros.unLado);
    calculadora.setDato_otrolado(parametros.otroLado);

    ResultadosDelCalculoSimple resultados = new
        ResultadosDelCalculoSimple();
    resultados.perimetro = calculadora.
        getResultado_perimetro();
    resultados.area = calculadora.getResultado_area();
    return resultados;
}

}

}
```

Utilizando el servicio HttpGet a través de una url



Utilizando el servicio HttpPut a través de un programilla de línea de comando

```
C:\Users\Public\Documents\01_PROGRAMACION\Pruebas_WEB-Core\zz-paraProbarServicioWebPost\bin\Debug\net5.0>
zz-paraProbarServicioWebPost.exe 2 m 300 cm
PARAMETROS: {"unLado":{"valor":2,"unidaddemedida":"m"},"otroLado":{"valor":300,"unidaddemedida":"cm"}}
RESULTADO: {"perimetro":{"valor":10,"unidaddemedida":"m"},"area":{"valor":6,"unidaddemedida":"m2"}}
Perimetro = 10 m
Area = 6 m2
```

Listing 17.62: 'Program.cs'

```

using System;

namespace zz_paraProbarServicioWebPost
{
    public class Magnitud
    {
        public double valor { get; set; }
        public string unidadmedida { get; set; }
    }
    public class ParametrosDelCalculoSimple
    {
        public Magnitud unLado { get; set; }
        public Magnitud otroLado { get; set; }
    }
    public class ResultadosDelCalculoSimple
    {
        public Magnitud perimetro { get; set; }
        public Magnitud area { get; set; }
    }

    //      zz-paraProbarServicioWebPost.exe 2 m 300 cm
    //      PARAMETROS: {"unLado":{"valor":2,"unidadmedida":"m"},"
    //      otroLado":{"valor":300,"unidadmedida":"cm"}}
    //      RESULTADO: {"perimetro":{"valor":10,"unidadmedida":"m
    //      "},"area":{"valor":6,"unidadmedida":"m2"}}
    //      Perimetro = 10 m
    //      Area = 6 m2

    class Program
    {
        static async System.Threading.Tasks.Task Main(string[] args)
        {
            System.Net.Http.HttpClient clienteWeb = new System.Net.
                HttpClient();

            if (args.Length != 4)
            {
                Console.WriteLine("Forma de uso:      zz-
                    paraProbarServicioWebPost.exe unLado_valor
                    unLado_unidad otroLado_valor otroLado_unidad");
                Console.WriteLine("por ejemplo:
                    paraProbarServicioWebPost.exe 2 m 300 cm");
            }
            else
            {
                Magnitud unLado = new Magnitud();
                unLado.valor = double.Parse(args[0]);
                unLado.unidadmedida = args[1];
                Magnitud otroLado = new Magnitud();
                otroLado.valor = double.Parse(args[2]);
                otroLado.unidadmedida = args[3];

                ParametrosDelCalculoSimple parametros = new
                    ParametrosDelCalculoSimple();
            }
        }
    }
}

```

```

        parametros.unLado = unLado;
        parametros.otroLado = otroLado;
        string parametrosJson = System.Text.Json.
            JsonSerializer.Serialize(parametros);

        Console.WriteLine("PARAMETROS: " + parametrosJson);

    try
    {
        System.Net.Http.HttpContent
            parametrosCodificados = new System.Net.Http.
                StringContent(content: parametrosJson,
                    encoding: System.Text.Encoding.UTF8,
                    mediaType: "application/json");
        System.Net.Http.HttpResponseMessage respuesta =
            await clienteWeb.PostAsync("https://
                localhost:44310/api/CalculoSimple",
                parametrosCodificados);
        respuesta.EnsureSuccessStatusCode();
        string resultadoEnBruto = await respuesta.
            Content.ReadAsStringAsync();

        Console.WriteLine("RESULTADO: " +
            resultadoEnBruto);

        ResultadosDelCalculoSimple resultados = (
            ResultadosDelCalculoSimple)System.Text.Json.
                JsonSerializer.Deserialize(resultadoEnBruto,
                    typeof(ResultadosDelCalculoSimple));

        Console.WriteLine("Perimetro = " + resultados.
            perimetro.valor.ToString() + " " +
            resultados.perimetro.unidaddemedida);
        Console.WriteLine("    Area = " + resultados.
            area.valor.ToString() + " " + resultados.
            area.unidaddemedida);

    }
    catch (Exception ex)
    {
        Console.WriteLine("ERROR: " + ex.Message);
    }

}

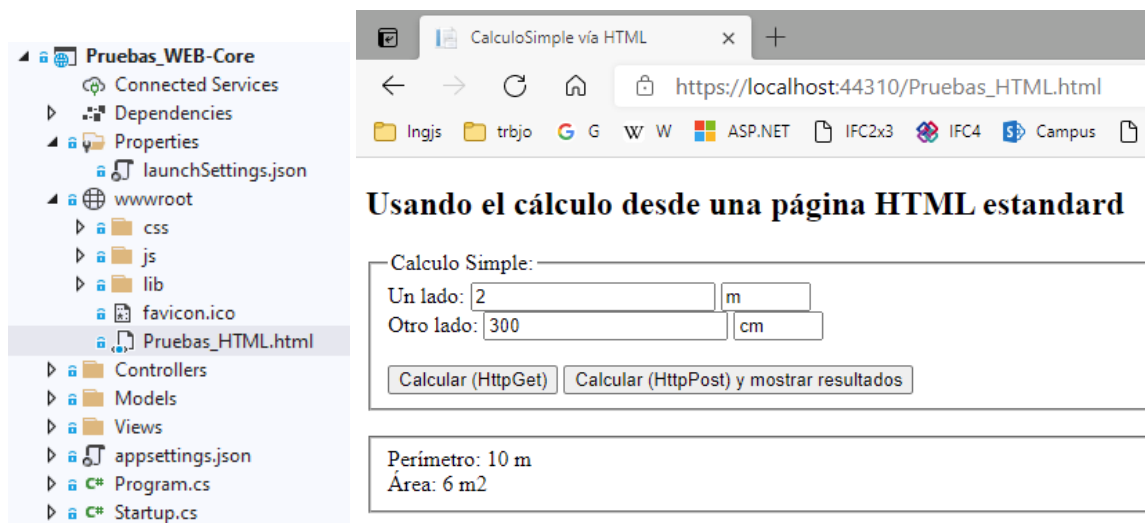
}

}

}

```

Utilizando el servicio HttpPut desde una página web HTML/JavaScript



Listing 17.63: 'Pruebas_HTML.html'

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>CalculoSimple vía HTML</title>
</head>
<body>
  <h2>Usando el cálculo desde una página HTML estandard</h2>

  <div>
    <form name="frmCalculoSimple" action="/api/CalculoSimple"
      target="_self">
      <fieldset>
        <legend>Calculo Simple:</legend>
        <label for="unLado_valor unLado_unidaddemedida">Un
          lado: </label>
        <input type="number" id="unLado_valor" name="
          unLado_valor" required value="2" />
        <input type="text" id="unLado_unidaddemedida" name="
          unLado_unidaddemedida" size="4" placeholder="cm"
          required value="m" />
        <br />
        <label for="otroLado_valor otroLado_unidaddemedida">
          Otro lado: </label>
        <input type="number" id="otroLado_valor" name="
          otroLado_valor" required value="300" />
        <input type="text" id="otroLado_unidaddemedida" name=
          "otroLado_unidaddemedida" size="4" placeholder=
          "cm" required value="cm" />
        <br />
        <br />
        <input type="submit" value="Calcular (HttpGet)"
          formmethod="get" />
        <button type="button" onclick="
          CalcularYMostrarResultados()">Calcular (HttpPost
          ) y mostrar resultados</button>
      </fieldset>
    </div>

    <div>
      Perímetro: 10 m
      Área: 6 m2
    </div>
  </div>
</body>
</html>
```

```

</fieldset>
<br />
<fieldset>
  <label for="perimetro">Perímetro: </label>
  <output id="perimetro"></output>
  <br />
  <label for="area">Área: </label>
  <output id="area"></output>
</fieldset>
</form>
</div>

<script>
  function CalcularYMostrarResultados() {

    var solicitud = new XMLHttpRequest();

    solicitud.onload = function () {
      var respuesta = solicitud.responseText;
      alert("RESPUESTA: " + respuesta);

      if (solicitud.statusText === "OK") {
        var resultados = JSON.parse(respuesta)
        document.getElementById("perimetro").value =
          resultados.perimetro.valor + " " +
          resultados.perimetro.unidaddemedida;
        document.getElementById("area").value =
          resultados.area.valor + " " + resultados.
            area.unidaddemedida;
      }
    }

    var esAsincrono = true;
    solicitud.open("POST", "/api/CalculoSimple", esAsincrono
    );

    solicitud.setRequestHeader("Content-Type", "application/
      json;charset=UTF-8");

    var parametros = JSON.stringify(
      {
        "unLado": {
          "valor": document.getElementById("
            unLado_valor").value
          , "unidaddemedida": document.getElementById(
            "unLado_unidaddemedida").value
        }
        , "otroLado": {
          "valor": document.getElementById("
            otroLado_valor").value
          , "unidaddemedida": document.getElementById(
            "otroLado_unidaddemedida").value
        }
      }
    )
    alert("PARAMETROS: " + parametros);
  }

```

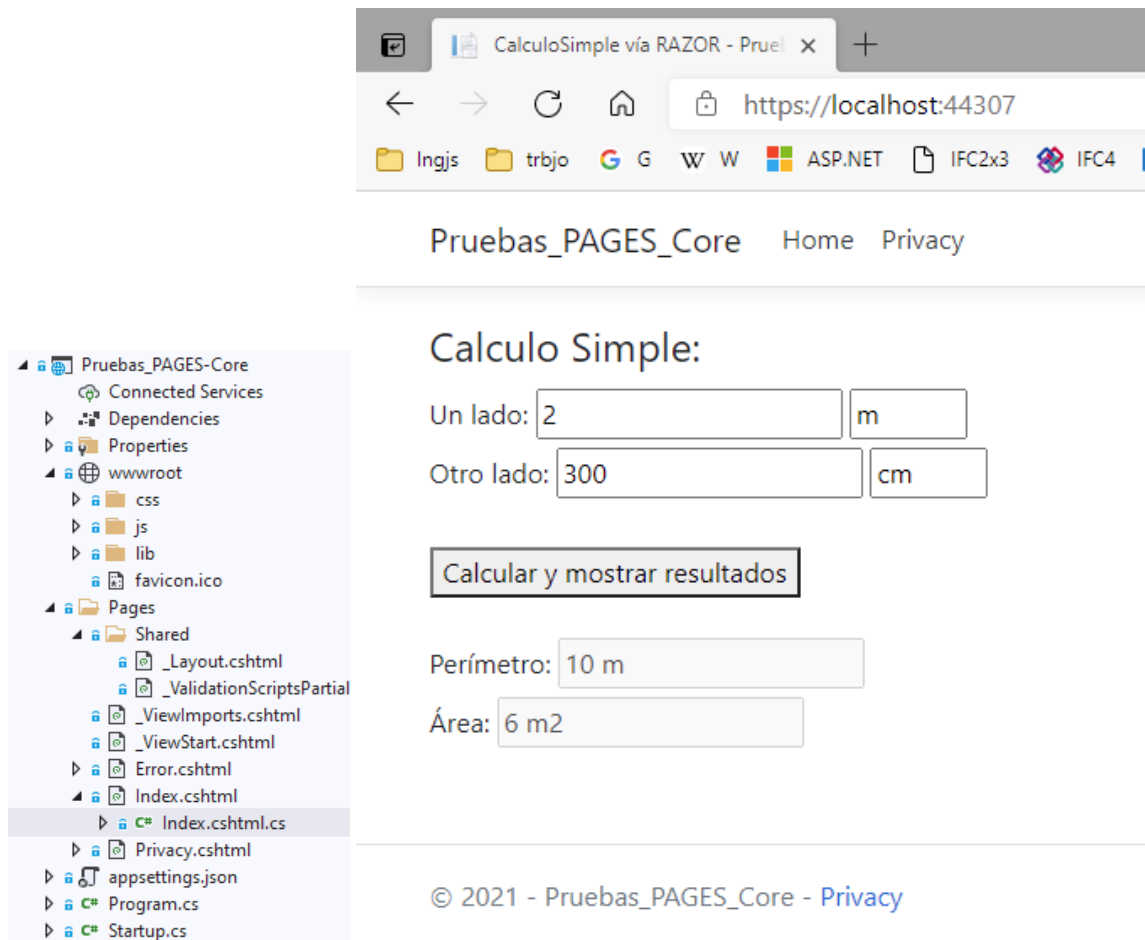
```

        solicitud.send(parametros);
    }
</script>

</body>
</html>

```

Utilizando el servicio HttpPut desde una página web RAZOR/C#



Listing 17.64: 'Index.cshtml'

```

@page
@model IndexModel
@{
    ViewData["Title"] = "CalculoSimple vía RAZOR";
}

<div>
    <form method="post">
        <fieldset>
            <legend>Calculo Simple:</legend>

```

```

        <label for="unLado_valor unLado_unidaddemedida">Un lado:
        </label>
        <input type="number" asp-for="unLado_valor" required />
        <input type="text" asp-for="unLado_unidaddemedida" size=
        "4" placeholder="cm" required />
        <br />
        <label for="otroLado_valor otroLado_unidaddemedida">Otro
        lado: </label>
        <input type="number" asp-for="otroLado_valor" required
        />
        <input type="text" asp-for="otroLado_unidaddemedida"
        size="4" placeholder="cm" required />
        <br />
        <br />
        <button type="submit">Calcular y mostrar resultados</
        button>
    </fieldset>
    <br />
    <fieldset>
        <label for="perimetro">Perímetro: </label>
        <input asp-for="perimetro" disabled />
        <br />
        <label for="area">Área: </label>
        <input asp-for="area" disabled />
    </fieldset>
</form>
</div>

```

Listing 17.65: 'Index.cshtml.cs'

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;
using System;

namespace Pruebas_PAGES_Core.Pages
{
    public class IndexModel : PageModel
    {
        [BindProperty]
        public double unLado_valor { get; set; }
        [BindProperty]
        public string unLado_unidaddemedida { get; set; }
        [BindProperty]
        public double otroLado_valor { get; set; }
        [BindProperty]
        public string otroLado_unidaddemedida { get; set; }

        [BindProperty]
        public string perimetro { get; private set; }
        [BindProperty]
        public string area { get; private set; }

        private readonly ILogger<IndexModel> _logger;
    }
}

```

```
public IndexModel (ILogger<IndexModel> logger)
{
    _logger = logger;
}

public IActionResult OnGet ()
{
    unLado_valor = 2;
    unLado_unidadmedida = "m";
    otroLado_valor = 300;
    otroLado_unidadmedida = "cm";

    return Page();
}

public IActionResult OnPost ()
{
    CalculoSimple.CalculoSimple calculadora = new
        CalculoSimple.CalculoSimple();

    try
    {
        calculadora.setDato_unlado(new Magnitudes.Magnitud(
            unLado_valor, unLado_unidadmedida));
        calculadora.setDato_otrolado(new Magnitudes.Magnitud(
            otroLado_valor, otroLado_unidadmedida));
    }
    catch (ArgumentOutOfRangeException ex)
    {
        //MessageBox.Show("Alguna de las unidades de medida
            no está contemplada en las conversiones: " + ex.
            Message);
    }
    catch (System.IO.FileNotFoundException ex)
    {
        //MessageBox.Show("No se ha encontrado el archivo
            con la lista de conversiones entre unidades de
            medida: " + ex.Message);
    }
    catch (ArgumentException ex)
    {
        //MessageBox.Show("Problemas leyendo el archivo con
            la lista de conversiones entre unidades de
            medida: " + ex.Message);
    }
    catch (Exception ex)
    {
        //MessageBox.Show("Problemas en alguna conversión
            entre unidades de medida: " + ex.Message);
    }

    perimetro = calculadora.getResultado_perimetro().
        ToString();
    area = calculadora.getResultado_area().ToString();
}
```

```

        return Page();
    }

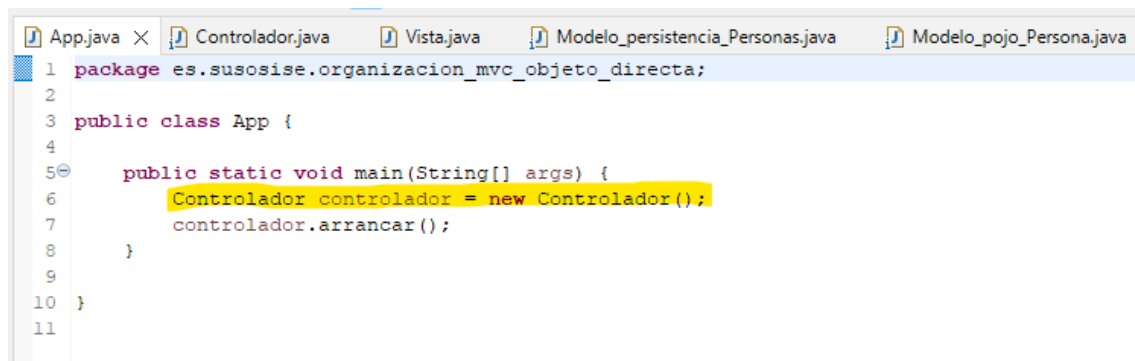
}

```

15.2 {Arquitectura en tres capas, MVC}

Dependencias directas:

Cada objeto se instancia él mismo los otros objetos que necesita.

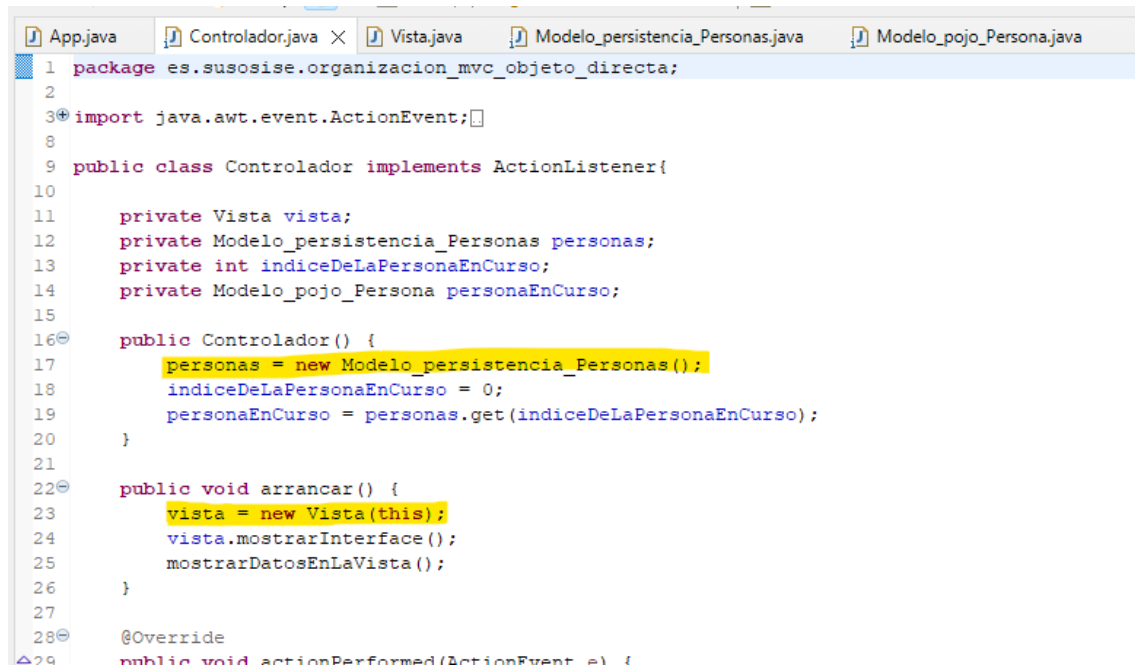


```

App.java × Controlador.java Vista.java Modelo_persistencia_Personas.java Modelo_pojo_Persona.java
1 package es.susosise.organizacion_mvc_objeto_directa;
2
3 public class App {
4
5     public static void main(String[] args) {
6         Controlador controlador = new Controlador();
7         controlador.arrancar();
8     }
9
10 }
11

```

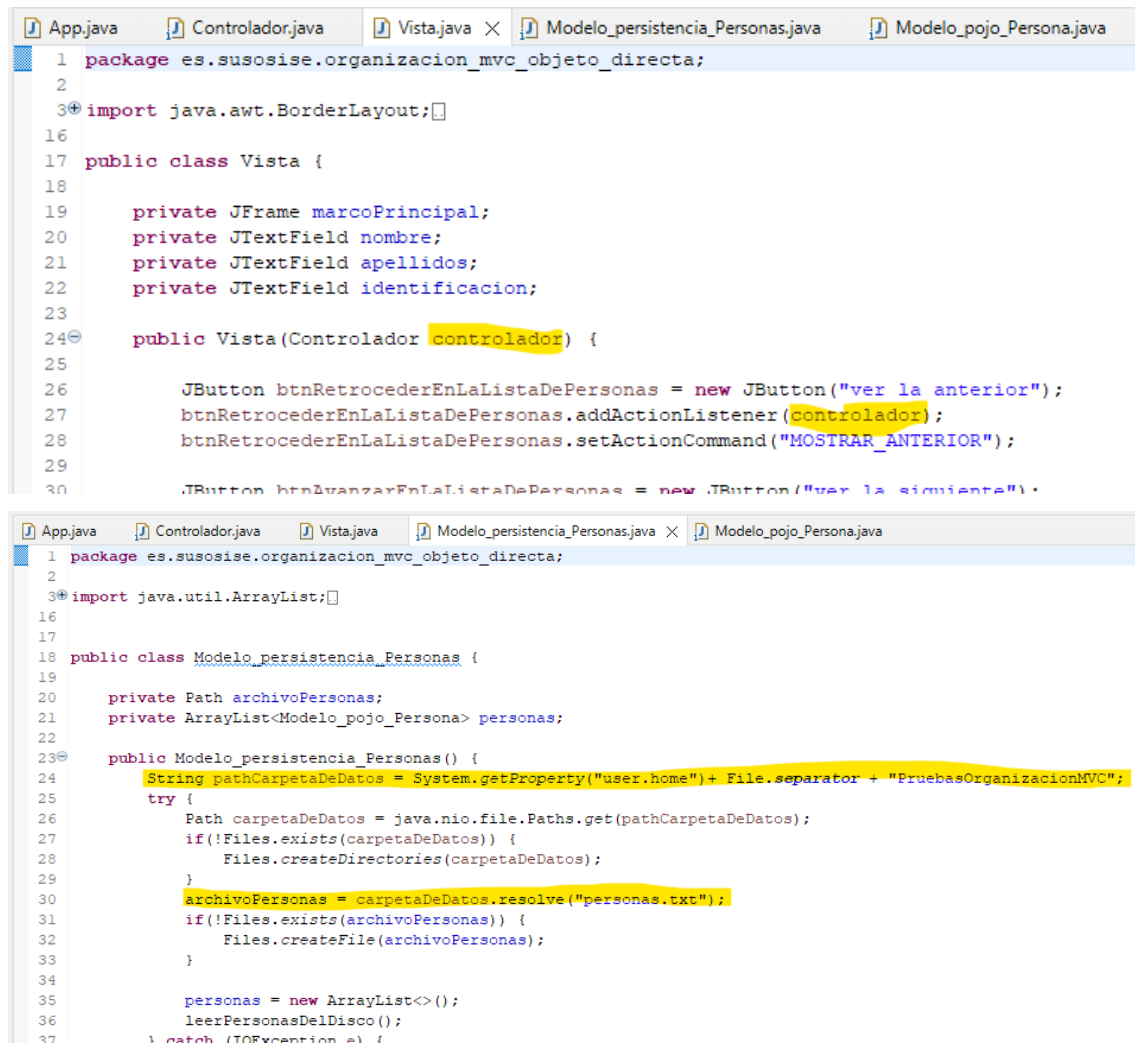
nota: Como la vista y el controlador tienen dependencias mutuamente cíclicas, al final siempre uno ha de crear al otro e inyectarle su dependencia... En este caso, el controlador crea la vista y se pasa a sí mismo en el constructor de esta.



```

App.java Controlador.java × Vista.java Modelo_persistencia_Personas.java Modelo_pojo_Persona.java
1 package es.susosise.organizacion_mvc_objeto_directa;
2
3 import java.awt.event.ActionEvent;
4
5
6
7
8
9 public class Controlador implements ActionListener{
10
11     private Vista vista;
12     private Modelo_persistencia_Personas personas;
13     private int indiceDeLaPersonaEnCurso;
14     private Modelo_pojo_Persona personaEnCurso;
15
16     public Controlador() {
17         personas = new Modelo_persistencia_Personas();
18         indiceDeLaPersonaEnCurso = 0;
19         personaEnCurso = personas.get(indiceDeLaPersonaEnCurso);
20     }
21
22     public void arrancar() {
23         vista = new Vista(this);
24         vista.mostrarInterface();
25         mostrarDatosEnLaVista();
26     }
27
28     @Override
29     public void actionPerformed(ActionEvent e) {

```



Listing 17.66: 'App.java'

```

package es.susosise.mvc_directa;

public class App {

    public static void main(String[] args) {
        Controlador controlador = new Controlador();
        controlador.arrancar();
    }

}

```

Listing 17.67: 'Controlador.java'

```

package es.susosise.mvc_directa;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;

import javax.swing.JOptionPane;

public class Controlador implements ActionListener{

```

```
private Vista vista;
private Modelo_persistencia_Personas personas;
private int indiceDeLaPersonaEnCurso;
private Modelo_pojo_Persona personaEnCurso;

public Controlador() {
    personas = new Modelo_persistencia_Personas();
    indiceDeLaPersonaEnCurso = 0;
    personaEnCurso = personas.get(indiceDeLaPersonaEnCurso);
}

public void arrancar() {
    vista = new Vista(this);
    vista.mostrarInterface();
    mostrarDatosEnLaVista();
}

@Override
public void actionPerformed(ActionEvent e) {
    String comando = e.getActionCommand();
    if (comando.equals("MOSTRAR_ANTERIOR")) {
        indiceDeLaPersonaEnCurso = indiceDeLaPersonaEnCurso - 1;
        if (indiceDeLaPersonaEnCurso < 0 ) {
            indiceDeLaPersonaEnCurso = personas.getCuántasHay() -
                1;
        }
        personaEnCurso = personas.get(indiceDeLaPersonaEnCurso);
        if(personaEnCurso == null) {
            JOptionPane.showMessageDialog(null, "No hay ninguna
                persona en la lista");
        }
        mostrarDatosEnLaVista();
    }
    else if (comando.equals("MOSTAR_SIGUIENTE")) {
        indiceDeLaPersonaEnCurso = indiceDeLaPersonaEnCurso + 1;
        if (indiceDeLaPersonaEnCurso > (personas.getCuántasHay()
            - 1 )) {
            indiceDeLaPersonaEnCurso = 0;
        }
        personaEnCurso = personas.get(indiceDeLaPersonaEnCurso);
        if(personaEnCurso == null) {
            JOptionPane.showMessageDialog(null, "No hay ninguna
                persona en la lista");
        }
        mostrarDatosEnLaVista();
    }
    else if (comando.equals("NUEVA_EN_BLANCO")) {
        personaEnCurso = null;
        mostrarDatosEnLaVista();
    }
    else if (comando.equals("GUARDAR_PERSONA")) {
        if (personaEnCurso == null) {
            if (!vista.getNombre().isBlank() || !vista.
                getApellidos().isBlank()) {
                personaEnCurso = Modelo_pojo_Persona.
```



```

        crearNuevaPersona(vista.getNombre(), vista.
            getApellidos());
        personaEnCurso.setDni_cedula_pasaporte_o_similar(
            vista.getIdentificacion());
    }
}
else {
    personaEnCurso.setNombre(vista.getNombre());
    personaEnCurso.setApellidos(vista.getApellidos());
    personaEnCurso.setDni_cedula_pasaporte_o_similar(
        vista.getIdentificacion());
}

try {
    personas.guardar(personaEnCurso);
} catch (IOException ex) {
    JOptionPane.showMessageDialog(null, "Se ha producido
        un error al guardar." + ex.getMessage());
    ex.printStackTrace();
}
}

private void mostrarDatosEnLaVista() {
    if (personaEnCurso == null) {
        vista.setNombre("");
        vista.setApellidos("");
        vista.setIdentificacion("");
    }
    else {
        vista.setNombre(personaEnCurso.getNombre());
        vista.setApellidos(personaEnCurso.getApellidos());
        vista.setIdentificacion(personaEnCurso.
            getDni_cedula_pasaporte_o_similar());
    }
}
}
}

```

Listing 17.68: 'Vista.java'

```

package es.susosise.mvc_directa;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.FlowLayout;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

```

```
import javax.swing.JTextField;
import javax.swing.WindowConstants;

public class Vista {

    private JFrame marcoPrincipal;
    private JTextField nombre;
    private JTextField apellidos;
    private JTextField identificacion;

    public Vista(Controlador controlador) {

        JButton btnRetrocederEnLaListaDePersonas = new JButton("ver la
            anterior");
        btnRetrocederEnLaListaDePersonas.addActionListener(controlador);
        btnRetrocederEnLaListaDePersonas.setActionCommand("
            MOSTRAR_ANTERIOR");

        JButton btnAvanzarEnLaListaDePersonas = new JButton("ver la
            siguiente");
        btnAvanzarEnLaListaDePersonas.addActionListener(controlador);
        ;
        btnAvanzarEnLaListaDePersonas.setActionCommand("
            MOSTAR_SIGUIENTE");

        JButton btnUnaNuevaEnBlanco = new JButton("una NUEVA en
            blanco");
        btnUnaNuevaEnBlanco.addActionListener(controlador);
        btnUnaNuevaEnBlanco.setActionCommand("NUEVA_EN_BLANCO");

        JPanel botoneraSuperior = new JPanel();
        botoneraSuperior.setLayout(new FlowLayout());
        botoneraSuperior.setAlignmentX(FlowLayout.LEFT);
        botoneraSuperior.add(btnRetrocederEnLaListaDePersonas);
        botoneraSuperior.add(btnAvanzarEnLaListaDePersonas);
        botoneraSuperior.add(btnUnaNuevaEnBlanco);

        JButton btnGuardarPersona = new JButton("GUARDAR");
        btnGuardarPersona.addActionListener(controlador);
        btnGuardarPersona.setActionCommand("GUARDAR_PERSONA");

        JPanel botoneraInferior = new JPanel();
        botoneraInferior.setLayout(new FlowLayout());
        botoneraInferior.setAlignmentX(FlowLayout.LEFT);
        botoneraInferior.add(btnGuardarPersona);

        Box marcoNombre = new Box(BoxLayout.X_AXIS);
        marcoNombre.setAlignmentX(Component.LEFT_ALIGNMENT);
        JLabel lblNombre = new JLabel("Nombre:");
        nombre = new JTextField();
        nombre.setPreferredSize(new Dimension(75, 25));
        marcoNombre.add(lblNombre);
        marcoNombre.add(Box.createRigidArea(new Dimension(5,0)));
        marcoNombre.add(nombre);
```

```

Box marcoApellidos = new Box(BoxLayout.X_AXIS);
marcoApellidos.setAlignmentX(Component.LEFT_ALIGNMENT);
JLabel lblApellidos = new JLabel("Apellidos:");
apellidos = new JTextField();
apellidos.setPreferredSize(new Dimension(250, 25));
marcoApellidos.add(lblApellidos);
marcoApellidos.add(Box.createRigidArea(new Dimension(5, 0)));
marcoApellidos.add(apellidos);

Box marcoIdentificacion = new Box(BoxLayout.X_AXIS);
marcoIdentificacion.setAlignmentX(Component.LEFT_ALIGNMENT);
JLabel lblIdentificacion = new JLabel("Identificacion:");
identificacion = new JTextField();
identificacion.setPreferredSize(new Dimension(75, 25));
marcoIdentificacion.add(lblIdentificacion);
marcoIdentificacion.add(Box.createRigidArea(new Dimension
    (5, 0)));
marcoIdentificacion.add(identificacion);

Box formulario = new Box(BoxLayout.Y_AXIS);
formulario.setAlignmentX(Component.LEFT_ALIGNMENT);
formulario.setBorder(BorderFactory.createEmptyBorder
    (10, 10, 10, 10));
formulario.add(marcoNombre);
formulario.add(Box.createRigidArea(new Dimension(0, 5)));
formulario.add(marcoApellidos);
formulario.add(Box.createRigidArea(new Dimension(0, 5)));
formulario.add(marcoIdentificacion);

marcoPrincipal = new JFrame();
marcoPrincipal.setDefaultCloseOperation(WindowConstants.
    EXIT_ON_CLOSE);
marcoPrincipal.setTitle("Personas");
marcoPrincipal.getContentPane().setLayout(new BorderLayout());
marcoPrincipal.add(botoneraSuperior, BorderLayout.NORTH);
marcoPrincipal.add(formulario, BorderLayout.CENTER);
marcoPrincipal.add(botoneraInferior, BorderLayout.SOUTH);
marcoPrincipal.pack();
}

protected void mostrarInterface() {
    marcoPrincipal.setVisible(true);
}

protected String getNombre() {
    return nombre.getText();
}

protected void setNombre(String texto) {
    nombre.setText(texto);
}

protected String getApellidos() {
    return apellidos.getText();
}

```

```

    }
    protected void setApellidos(String texto) {
        apellidos.setText(texto);
    }

    protected String getIdentificacion() {
        return identificacion.getText();
    }
    protected void setIdentificacion(String texto) {
        identificacion.setText(texto);
    }
}

```

Listing 17.69: 'Modelo_persistencia_Personas.java'

```

package es.susosise.mvc_directa;

import java.util.ArrayList;
import java.util.UUID;

import javax.swing.JOptionPane;

import java.nio.file.Path;
import java.nio.file.Files;
import java.io.IOException;
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;

public class Modelo_persistencia_Personas {

    private Path archivoPersonas;
    private ArrayList<Modelo_pojo_Persona> personas;

    public Modelo_persistencia_Personas() {
        String pathCarpetaDeDatos = System.getProperty("user.home") +
            File.separator + "PruebasOrganizacionMVC";
        try {
            Path carpetaDeDatos = java.nio.file.Paths.get(
                pathCarpetaDeDatos);
            if(!Files.exists(carpetaDeDatos)) {
                Files.createDirectories(carpetaDeDatos);
            }
            archivoPersonas = carpetaDeDatos.resolve("personas.txt");
            if(!Files.exists(archivoPersonas)) {
                Files.createFile(archivoPersonas);
            }

            personas = new ArrayList<>();
            leerPersonasDelDisco();
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null,

```

```

        "Problemas al acceder al archivo de personas en
        la carpeta indicada: [" + pathCarpetaDeDatos
        + "]"
        + System.lineSeparator() + System.lineSeparator
        ()
        + e.getMessage());
    e.printStackTrace();
}

}

public Modelo_pojo_Persona get(UUID idInterno) {
    for(Modelo_pojo_Persona persona : personas) {
        if (persona.getIdInterno().equals(idInterno)) {
            return persona;
        }
    }
    return null;
}

public Modelo_pojo_Persona get(String nombreyapellidos) {
    for(Modelo_pojo_Persona persona : personas) {
        String texto = persona.getNombre() + " " + persona.
            getApellidos();
        if (texto.equals(nombreyapellidos)) {
            return persona;
        }
    }
    return null;
}

public int getCuantasHay() {
    return personas.size();
}

public Modelo_pojo_Persona get(int indice) {
    if (!personas.isEmpty() && indice > -1 && indice < personas.
        size()) {
        return personas.get(indice);
    }
    else {
        return null;
    }
}

public void guardar(Modelo_pojo_Persona persona) throws
    IOException {
    Modelo_pojo_Persona existente = get(persona.getIdInterno());
    if (existente == null) {
        personas.add(persona);
    }
    else {
        existente.actualizarDatos(persona);
    }
}

```

```

        guardarPersonasEnElDisco();
    }

    private void leerPersonasDelDisco() throws IOException {

        personas.clear();
        try(BufferedReader lector = new BufferedReader(new FileReader(
            archivoPersonas.toFile()))){
            String linea = lector.readLine();
            while(linea != null) {
                personas.add(Modelo_pojo_Persona.deserializarDesdeTexto(
                    linea, " : "));
                linea = lector.readLine();
            }
        }
    }

    private void guardarPersonasEnElDisco() throws IOException {

        try(BufferedWriter escritorPersonas = new BufferedWriter(new
            FileWriter(archivoPersonas.toFile()))){
            for(Modelo_pojo_Persona persona : personas) {
                escritorPersonas.write(persona.serializarATexto(" : "));
            }
        }
    }
}

```

Listing 17.70: 'Modelo_pojo_Persona.java'

```

package es.susosise.mvc_directa;

import java.util.Objects;
import java.util.UUID;

public class Modelo_pojo_Persona {
    private UUID idInterno;
    private String dni_cedula_pasaporte_o_similar;
    private String nombre;
    private String apellidos;

    public Modelo_pojo_Persona(UUID idInterno, String pasaporte,
        String nombre, String apellidos) {
        this.idInterno = idInterno;
        this.dni_cedula_pasaporte_o_similar = pasaporte;
        this.nombre = nombre;
        this.apellidos = apellidos;
    }

    public static Modelo_pojo_Persona crearNuevaPersona(String nombre,
        String apellidos) {
        return new Modelo_pojo_Persona(UUID.randomUUID(), null,

```

```
        nombre, apellidos);
    }

    protected UUID getIdInterno() {
        return idInterno;
    }
    protected void setIdInterno(UUID id) {
        this.idInterno = id;
    }

    public String getDni_cedula_pasaporte_o_similar() {
        return dni_cedula_pasaporte_o_similar;
    }

    public void setDni_cedula_pasaporte_o_similar(String id) {
        this.dni_cedula_pasaporte_o_similar = id;
    }

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellidos() {
        return apellidos;
    }
    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    public void actualizarDatos(Modelo_pojo_Persona personaActualizada
    ) {
        this.dni_cedula_pasaporte_o_similar = personaActualizada.
            getDni_cedula_pasaporte_o_similar();
        this.nombre = personaActualizada.getNombre();
        this.apellidos = personaActualizada.getApellidos();
    }

    @Override
    public int hashCode() {
        return Objects.hash(this.idInterno,
            this.dni_cedula_pasaporte_o_similar,
            this.nombre,
            this.apellidos);
    }

    @Override
    public boolean equals(Object unObjeto) {
        if (this == unObjeto) {
            return true;
        }
        if (unObjeto == null) {
            return false;
        }
    }
}
```

```
    }
    if (getClass() != unObjeto.getClass()) {
        return false;
    }
    Modelo_pojo_Persona unaPersona = (Modelo_pojo_Persona)
        unObjeto;
    return Objects.equals(this.idInterno, unaPersona.idInterno)
        && Objects.equals(this.dni_cedula_pasaporte_o_similar,
            unaPersona.dni_cedula_pasaporte_o_similar)
        && Objects.equals(this.nombre, unaPersona.nombre)
        && Objects.equals(this.apellidos, unaPersona.apellidos);
}

public String serializarATexto(String separador) {
    StringBuilder linea = new StringBuilder();

    linea.append(nombre);
    linea.append(separador);

    linea.append(apellidos);
    linea.append(separador);

    linea.append(dni_cedula_pasaporte_o_similar);
    linea.append(separador);

    linea.append(idInterno.toString());
    linea.append(System.lineSeparator());

    return linea.toString();
}

public static Modelo_pojo_Persona deserializarDesdeTexto(String
    lineaDeDatos, String separador) {
    String[] datos = lineaDeDatos.split(separador);
    if(datos.length == 4) {
        return new Modelo_pojo_Persona(UUID.fromString(datos[3]),
            datos[2], datos[0], datos[1]);
    }
    else {
        throw new IllegalArgumentException(
            "Error: no se puede interpretar correctamente esta linea:"
            + System.lineSeparator() + System.lineSeparator()
            + lineaDeDatos
        );
    }
}

}
```

Dependencias inversas (inversión de dependencias):

Cada objeto recibe desde fuera los otros objetos que necesita (o bien inyectados a través de su constructor o bien inyectados a través un metodo setter ad-hoc).

```

App.java × Controlador.java Vista.java Modelo_persistencia_Personas.java Modelo_pojo_Persona.java
1 package es.susosise.organizacion_mvc_objeto_inversa;
2
3 import java.io.IOException;
4
5
6
7 public class App {
8
9     public static void main(String[] args) {
10         if (args.length != 1) {
11             System.out.println("Se ha de pasar como argumento la ruta a la carpeta donde ubicar los datos.");
12         }
13         else {
14             try {
15
16                 Modelo_persistencia_Personas personas;
17                 personas = new Modelo_persistencia_Personas(args[0]);
18
19                 Controlador controlador = new Controlador(personas);
20
21                 Vista vista = new Vista(controlador);
22
23                 controlador.arrancar(vista);
24
25             } catch (IOException e) {
26                 JOptionPane.showMessageDialog(null,
27                     "Problemas al acceder a los datos en la carpeta indicada: [" + args[0] + "]"
28                     + System.lineSeparator() + System.lineSeparator()
29                     + e.getMessage());
30                 e.printStackTrace();
31             }
32         }
33     }
34 }
35 }
36

```

```

App.java Controlador.java × Vista.java Modelo_persistencia_Personas.java Modelo_pojo_Persona.java
1 package es.susosise.organizacion_mvc_objeto_inversa;
2
3 import java.awt.event.ActionEvent;
4
5
6
7 public class Controlador implements ActionListener{
8
9     private Vista vista;
10    private Modelo_persistencia_Personas personas;
11    private int indiceDeLaPersonaEnCurso;
12    private Modelo_pojo_Persona personaEnCurso;
13
14    public Controlador(Modelo_persistencia_Personas personas) {
15        this.personas = personas;
16        indiceDeLaPersonaEnCurso = 0;
17        personaEnCurso = personas.get(indiceDeLaPersonaEnCurso);
18    }
19
20    public void arrancar(Vista vista) {
21        this.vista = vista;
22        vista.mostrarInterface();
23        mostrarDatosEnLaVista();
24    }
25
26    @Override
27    public void actionPerformed(ActionEvent e) {
28        String comando = e.getActionCommand();
29        if (comando.equals("MOSTRAR_ANTERIOR")) {
30            indiceDeLaPersonaEnCurso = indiceDeLaPersonaEnCurso - 1;
31            if (indiceDeLaPersonaEnCurso < 0 ) {
32                indiceDeLaPersonaEnCurso = personas.getCuántasHay() - 1;
33            }
34            personaEnCurso = personas.get(indiceDeLaPersonaEnCurso);
35        }
36    }
37 }
38

```



Listing 17.71: 'App.java'

```

package es.susosise.mvc_inversa;

import java.io.IOException;

import javax.swing.JOptionPane;

public class App {

    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Se ha de pasar como argumento la ruta
                               a la carpeta donde ubicar los datos.");
        }
        else {
            try {

                Modelo_persistencia_Personas personas;
                personas = new Modelo_persistencia_Personas(args[0]);

```

```

        Controlador controlador = new Controlador(personas);

        Vista vista = new Vista(controlador);

        controlador.arrancar(vista);

    } catch (IOException e) {
        JOptionPane.showMessageDialog(null,
            "Problemas al acceder a los datos en la
            carpeta indicada: [" + args[0] + "]"
            + System.lineSeparator() + System.
            lineSeparator()
            + e.getMessage());
        e.printStackTrace();
    }
}
}
}

```

Listing 17.72: 'Controlador.java'

```

package es.susosise.mvc_inversa;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;

import javax.swing.JOptionPane;

public class Controlador implements ActionListener{

    private Vista vista;
    private Modelo_persistencia_Personas personas;
    private int indiceDeLaPersonaEnCurso;
    private Modelo_pojo_Persona personaEnCurso;

    public Controlador(Modelo_persistencia_Personas personas) {
        this.personas = personas;
        indiceDeLaPersonaEnCurso = 0;
        personaEnCurso = personas.get(indiceDeLaPersonaEnCurso);
    }

    public void arrancar(Vista vista) {
        this.vista = vista;
        vista.mostrarInterface();
        mostrarDatosEnLaVista();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String comando = e.getActionCommand();
        if (comando.equals("MOSTRAR_ANTERIOR")) {
            indiceDeLaPersonaEnCurso = indiceDeLaPersonaEnCurso - 1;
            if (indiceDeLaPersonaEnCurso < 0 ) {

```

```

        indiceDeLaPersonaEnCurso = personas.getCuantasHay() -
            1;
    }
    personaEnCurso = personas.get(indiceDeLaPersonaEnCurso);
    if(personaEnCurso == null) {
        JOptionPane.showMessageDialog(null, "No hay ninguna
            persona en la lista");
    }
    mostrarDatosEnLaVista();
}
else if (comando.equals("MOSTAR_SIGUIENTE")) {
    indiceDeLaPersonaEnCurso = indiceDeLaPersonaEnCurso + 1;
    if (indiceDeLaPersonaEnCurso > (personas.getCuantasHay()
        - 1)) {
        indiceDeLaPersonaEnCurso = 0;
    }
    personaEnCurso = personas.get(indiceDeLaPersonaEnCurso);
    if(personaEnCurso == null) {
        JOptionPane.showMessageDialog(null, "No hay ninguna
            persona en la lista");
    }
    mostrarDatosEnLaVista();
}
else if (comando.equals("NUEVA_EN_BLANCO")) {
    personaEnCurso = null;
    mostrarDatosEnLaVista();
}
else if (comando.equals("GUARDAR_PERSONA")) {
    if (personaEnCurso == null) {
        if (!vista.getNombre().isBlank() || !vista.
            getApellidos().isBlank()) {
            personaEnCurso = Modelo_pojo_Persona.
                crearNuevaPersona(vista.getNombre(), vista.
                    getApellidos());
            personaEnCurso.setDni_cedula_pasaporte_o_similar(
                vista.getIdentificacion());
        }
    }
    else {
        personaEnCurso.setNombre(vista.getNombre());
        personaEnCurso.setApellidos(vista.getApellidos());
        personaEnCurso.setDni_cedula_pasaporte_o_similar(
            vista.getIdentificacion());
    }

    try {
        personas.guardar(personaEnCurso);
    } catch (IOException ex) {
        JOptionPane.showMessageDialog(null, "Se ha producido
            un error al guardar." + ex.getMessage());
        ex.printStackTrace();
    }
}

private void mostrarDatosEnLaVista() {

```

```

        if (personaEnCurso == null) {
            vista.setNombre("");
            vista.setApellidos("");
            vista.setIdentificacion("");
        }
        else {
            vista.setNombre(personaEnCurso.getNombre());
            vista.setApellidos(personaEnCurso.getApellidos());
            vista.setIdentificacion(personaEnCurso.
                getDni_cedula_pasaporte_o_similar());
        }
    }
}

```

Listing 17.73: 'Vista.java'

```

package es.susosise.mvc_inversa;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.FlowLayout;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.WindowConstants;

public class Vista {

    private JFrame marcoPrincipal;
    private JTextField nombre;
    private JTextField apellidos;
    private JTextField identificacion;

    public Vista(Controlador controlador) {

        JButton btnRetrocederEnLaListaDePersonas = new JButton("ver la
            anterior");
        btnRetrocederEnLaListaDePersonas.addActionListener(controlador);
        btnRetrocederEnLaListaDePersonas.setActionCommand("
            MOSTRAR_ANTERIOR");

        JButton btnAvanzarEnLaListaDePersonas = new JButton("ver la
            siguiente");
        btnAvanzarEnLaListaDePersonas.addActionListener(controlador);
        ;
        btnAvanzarEnLaListaDePersonas.setActionCommand("
            MOSTAR_SIGUIENTE");
    }
}

```

```
        JButton btnUnaNuevaEnBlanco = new JButton("una NUEVA en  
            blanco");  
        btnUnaNuevaEnBlanco.addActionListener(controlador);  
        btnUnaNuevaEnBlanco.setActionCommand("NUEVA_EN_BLANCO");  
  
        JPanel botoneraSuperior = new JPanel();  
        botoneraSuperior.setLayout(new FlowLayout());  
        botoneraSuperior.setAlignmentX(FlowLayout.LEFT);  
        botoneraSuperior.add(btnRetrocederEnLaListaDePersonas);  
        botoneraSuperior.add(btnAvanzarEnLaListaDePersonas);  
        botoneraSuperior.add(btnUnaNuevaEnBlanco);  
  
        JButton btnGuardarPersona = new JButton("GUARDAR");  
        btnGuardarPersona.addActionListener(controlador);  
        btnGuardarPersona.setActionCommand("GUARDAR_PERSONA");  
  
        JPanel botoneraInferior = new JPanel();  
        botoneraInferior.setLayout(new FlowLayout());  
        botoneraInferior.setAlignmentX(FlowLayout.LEFT);  
        botoneraInferior.add(btnGuardarPersona);  
  
        Box marcoNombre = new Box(BoxLayout.X_AXIS);  
        marcoNombre.setAlignmentX(Component.LEFT_ALIGNMENT);  
        JLabel lblNombre = new JLabel("Nombre:");  
        nombre = new JTextField();  
        nombre.setPreferredSize(new Dimension(75, 25));  
        marcoNombre.add(lblNombre);  
        marcoNombre.add(Box.createRigidArea(new Dimension(5, 0)));  
        marcoNombre.add(nombre);  
  
        Box marcoApellidos = new Box(BoxLayout.X_AXIS);  
        marcoApellidos.setAlignmentX(Component.LEFT_ALIGNMENT);  
        JLabel lblApellidos = new JLabel("Apellidos:");  
        apellidos = new JTextField();  
        apellidos.setPreferredSize(new Dimension(250, 25));  
        marcoApellidos.add(lblApellidos);  
        marcoApellidos.add(Box.createRigidArea(new Dimension(5, 0)));  
        marcoApellidos.add(apellidos);  
  
        Box marcoIdentificacion = new Box(BoxLayout.X_AXIS);  
        marcoIdentificacion.setAlignmentX(Component.LEFT_ALIGNMENT);  
        JLabel lblIdentificacion = new JLabel("Identificacion:");  
        identificacion = new JTextField();  
        identificacion.setPreferredSize(new Dimension(75, 25));  
        marcoIdentificacion.add(lblIdentificacion);  
        marcoIdentificacion.add(Box.createRigidArea(new Dimension  
            (5, 0)));  
        marcoIdentificacion.add(identificacion);  
  
        Box formulario = new Box(BoxLayout.Y_AXIS);  
        formulario.setAlignmentX(Component.LEFT_ALIGNMENT);  
        formulario.setBorder(BorderFactory.createEmptyBorder  
            (10, 10, 10, 10));  
        formulario.add(marcoNombre);
```

```

        formulario.add(Box.createRigidArea(new Dimension(0, 5)));
        formulario.add(marcoApellidos);
        formulario.add(Box.createRigidArea(new Dimension(0, 5)));
        formulario.add(marcoIdentificacion);

        marcoPrincipal = new JFrame();
        marcoPrincipal.setDefaultCloseOperation(WindowConstants.
            EXIT_ON_CLOSE);
        marcoPrincipal.setTitle("Personas");
        marcoPrincipal.getContentPane().setLayout(new BorderLayout());
        marcoPrincipal.add(botoneraSuperior, BorderLayout.NORTH);
        marcoPrincipal.add(formulario, BorderLayout.CENTER);
        marcoPrincipal.add(botoneraInferior, BorderLayout.SOUTH);
        marcoPrincipal.pack();
    }

    protected void mostrarInterface() {
        marcoPrincipal.setVisible(true);
    }

    protected String getNombre() {
        return nombre.getText();
    }

    protected void setNombre(String texto) {
        nombre.setText(texto);
    }

    protected String getApellidos() {
        return apellidos.getText();
    }

    protected void setApellidos(String texto) {
        apellidos.setText(texto);
    }

    protected String getIdentificacion() {
        return identificacion.getText();
    }

    protected void setIdentificacion(String texto) {
        identificacion.setText(texto);
    }
}

```

Listing 17.74: 'Modelo_persistencia_Personas.java'

```

package es.susosise.mvc_inversa;

import java.util.ArrayList;
import java.util.UUID;

import javax.swing.JOptionPane;

import java.nio.file.Path;
import java.nio.file.Files;

```

```

import java.io.IOException;
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.BufferedReader;

public class Modelo_persistencia_Personas {

    private static final String SEPARADOR = " : ";

    private static final String ARCHIVO_PERSONAS = "personas.txt";
    private Path archivoPersonas;
    private ArrayList<Modelo_pojo_Persona> personas;

    public Modelo_persistencia_Personas(String pathCarpetaDeDatos)
        throws IOException {
        try {
            Path carpetaDeDatos = java.nio.file.Paths.get(
                pathCarpetaDeDatos);
            if(!Files.exists(carpetaDeDatos)) {
                Files.createDirectories(carpetaDeDatos);
            }
            archivoPersonas = carpetaDeDatos.resolve(ARCHIVO_PERSONAS);
            if(!Files.exists(archivoPersonas)) {
                Files.createFile(archivoPersonas);
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null,
                "Problemas al acceder al archivo de personas en  

                la carpeta indicada: [" + pathCarpetaDeDatos  

                + "]"
                + System.lineSeparator() + System.lineSeparator()
                + e.getMessage());
            e.printStackTrace();
        }

        personas = new ArrayList<>();
        leerPersonasDelDisco();
    }

    public Modelo_pojo_Persona get(UUID idInterno) {
        for(Modelo_pojo_Persona persona : personas) {
            if (persona.getIdInterno().equals(idInterno)) {
                return persona;
            }
        }
        return null;
    }

    public Modelo_pojo_Persona get(String nombreyapellidos) {
        for(Modelo_pojo_Persona persona : personas) {
            String texto = persona.getNombre() + " " + persona.

```



```

        getApellidos();
        if (texto.equals(nombreyapellidos)) {
            return persona;
        }
    }
    return null;
}

public int getCuantasHay() {
    return personas.size();
}

public Modelo_pojo_Persona get(int indice) {
    if (!personas.isEmpty() && indice > -1 && indice < personas.size()) {
        return personas.get(indice);
    }
    else {
        return null;
    }
}

public void guardar(Modelo_pojo_Persona persona) throws
    IOException {
    Modelo_pojo_Persona existente = get(persona.getIdInterno());
    if (existente == null) {
        personas.add(persona);
    }
    else {
        existente.actualizarDatos(persona);
    }
    guardarPersonasEnElDisco();
}

private void leerPersonasDelDisco() throws IOException {

    personas.clear();
    try(BufferedReader lector = new BufferedReader(new FileReader(
        archivoPersonas.toFile()))){
        String linea = lector.readLine();
        while(linea != null) {
            personas.add(Modelo_pojo_Persona.deserializarDesdeTexto(
                linea, SEPARADOR));
            linea = lector.readLine();
        }
    }
}

private void guardarPersonasEnElDisco() throws IOException {

    try(BufferedWriter escritorPersonas = new BufferedWriter(new
        FileWriter(archivoPersonas.toFile()))){
        for(Modelo_pojo_Persona persona : personas) {
            escritorPersonas.write(persona.serializarATexto(SEPARADOR));

```

```

    }
}

}

}

```

Listing 17.75: 'Modelo_pojo_Persona.java'

```

package es.susosise.mvc_inversa;

import java.util.Objects;
import java.util.UUID;

public class Modelo_pojo_Persona {
    private UUID idInterno;
    private String dni_cedula_pasaporte_o_similar;
    private String nombre;
    private String apellidos;

    public Modelo_pojo_Persona(UUID idInterno, String pasaporte,
                               String nombre, String apellidos) {
        this.idInterno = idInterno;
        this.dni_cedula_pasaporte_o_similar = pasaporte;
        this.nombre = nombre;
        this.apellidos = apellidos;
    }

    public static Modelo_pojo_Persona crearNuevaPersona(String nombre,
                                                         String apellidos) {
        return new Modelo_pojo_Persona(UUID.randomUUID(), null,
                                         nombre, apellidos);
    }

    protected UUID getIdInterno() {
        return idInterno;
    }

    protected void setIdInterno(UUID id) {
        this.idInterno = id;
    }

    public String getDni_cedula_pasaporte_o_similar() {
        return dni_cedula_pasaporte_o_similar;
    }

    public void setDni_cedula_pasaporte_o_similar(String id) {
        this.dni_cedula_pasaporte_o_similar = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

```

```
}

public String getApellidos() {
    return apellidos;
}

public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}

public void actualizarDatos(Modelo_pojo_Persona personaActualizada) {
    this.dni_cedula_pasaporte_o_similar = personaActualizada.
        getDni_cedula_pasaporte_o_similar();
    this.nombre = personaActualizada.getNombre();
    this.apellidos = personaActualizada.getApellidos();
}

@Override
public int hashCode() {
    return Objects.hash(this.idInterno,
        this.dni_cedula_pasaporte_o_similar,
        this.nombre,
        this.apellidos);
}

@Override
public boolean equals(Object unObjeto) {
    if (this == unObjeto) {
        return true;
    }
    if (unObjeto == null) {
        return false;
    }
    if (getClass() != unObjeto.getClass()) {
        return false;
    }
    Modelo_pojo_Persona unaPersona = (Modelo_pojo_Persona)
        unObjeto;
    return Objects.equals(this.idInterno, unaPersona.idInterno)
        && Objects.equals(this.dni_cedula_pasaporte_o_similar,
            unaPersona.dni_cedula_pasaporte_o_similar)
        && Objects.equals(this.nombre, unaPersona.nombre)
        && Objects.equals(this.apellidos, unaPersona.apellidos);
}

public String serializarATexto(String separador) {
    StringBuilder linea = new StringBuilder();

    linea.append(nombre);
    linea.append(separador);

    linea.append(apellidos);
    linea.append(separador);
}
```

```
        linea.append(dni_cedula_pasaporte_o_similar);
        linea.append(separador);

        linea.append(idInterno.toString());
        linea.append(System.lineSeparator());

        return linea.toString();
    }

    public static Modelo_pojo_Persona deserializarDesdeTexto(String
        lineaDeDatos, String separador) {
        String[] datos = lineaDeDatos.split(separador);
        if(datos.length == 4) {
            return new Modelo_pojo_Persona(UUID.fromString(datos[3]),
                datos[2], datos[0], datos[1]);
        }
        else {
            throw new IllegalArgumentException(
                "Error: no se puede interpretar correctamente esta linea:"
                + System.lineSeparator() + System.lineSeparator()
                + lineaDeDatos
            );
        }
    }
}
```

15.4 {Hotel}

Un ejemplo de posible solución puede verse en

<https://bitbucket.org/susosise/hotel/src/main/>

Aviso: está en construcción (y, por cierto, aún muy verde, solo empezando), así es que te animo a clonarlo y revisar de vez en cuando como va avanzando.