UNIVERSIDAD
POLITÉCNICA
DE MADRID

Recetario y planificador de comidas

Proyecto de la asignatura Taller de Programación
Convocatoria ordinaria, curso 2024-2025

E. T. S. de Ingeniería de Sistemas Informáticos
Universidad Politécnica de Madrid

15 de noviembre de 2024

Licencias

Atribución

El enunciado de esta práctica se ha elaborado con LaTeX, basándose en la plantilla “Sullivan Business Report” de Vel, publicada en `https://www.LaTeXTemplates.com`, que se distribuye bajo una licencia Atribución/Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional de Creative Commons.

Contacto

Para cualquier duda o comentario sobre esta práctica, por favor, contacta con los profesores de la asignatura.

Control de versiones

v1.0	15-11-2024	Versión final del enunciado de la práctica.

Índice

1 Introducción

La organización y planificación de las comidas es una tarea cotidiana que muchas personas enfrentan a diario. ¿Qué pasaría si pudieras simplificar este proceso utilizando tus habilidades de programación? En esta práctica, desarrollarás una aplicación de consola en `Java` para gestionar un libro de recetas y planificar las comidas de una semana, combinando creatividad y programación para resolver un problema real.

Este proyecto te permitirá poner en práctica conceptos fundamentales de programación, como la lectura y escritura desde la consola, la manipulación de arrays, el diseño de funciones y el uso de estructuras de control.

Al finalizar el proyecto, habrás creado una herramienta funcional que permitirá:

1. Registrar y consultar recetas, incluyendo ingredientes y pasos de preparación.
2. Organizar un plan semanal de comidas basado en las recetas disponibles.

Este proyecto no solo fortalecerá tus habilidades técnicas en `Java`, sino que también fomentará tu capacidad de abordar problemas desde una perspectiva lógica y estructurada, así como tu capacidad para resolver problemas con la programación.

1.1 Preparación del entorno de desarrollo

Para el desarrollo de este proyecto, podrás utilizar cualquier entorno de desarrollo integrado (IDE) que prefieras, aunque el soporte completo será ofrecido únicamente para `IntelliJ IDEA`, un potente IDE de `JetBrains` ampliamente utilizado en la industria para el cual, además, tenemos licencia educativa[1] toda la comunidad universitaria.

El esqueleto del proyecto se encuentra disponible en un repositorio de `GitHub`. El repositorio está configurado como plantilla, lo que significa que puedes crear un nuevo repositorio a partir de él y comenzar a trabajar en tu propia copia del proyecto. Para ello, sigue los siguientes pasos:

1. Accede al repositorio del proyecto[2].
2. Haz clic en el botón “Use this template” para crear un

[1] *Licencias educativas gratuitas. Asistencia a la comunidad*. JetBrains. 2024. URL: `https://www.jetbrains.com/es-es/community/education/#students`

[2] *Plantilla de Proyecto de Taller de Programación. Repositorio GitHub*. 2024. URL: `https://github.com/laracabrera/tp-ordinaria-2425`

nuevo repositorio a partir de la plantilla[3].

3. Introduce un nombre para tu repositorio y haz clic en "Create repository from template".
4. Clona tu nuevo repositorio en tu equipo y comienza a trabajar en tu copia del proyecto:
   - Si utilizas `IntelliJ IDEA`, puedes clonar el repositorio directamente desde el IDE. Para ello, selecciona la opción "Clone Repository" en el menú de inicio y pega la URL de tu repositorio.
   - Si vas a trabajar desde la línea de comandos, puedes clonar el repositorio con el comando `git clone` seguido de la URL de tu repositorio.
5. No te olvides de añadir al resto de integrantes como colaboradores en tu repositorio y configurar la visibilidad en **Privado**.

[3] *Crear un repositorio desde una plantilla*. GitHub Docs. 2024. URL: `https://docs.github.com/es/repositories/creating-and-managing-repositories/creating-a-repository-from-a-template`

El proyecto se ha creado con `Maven`[4], un sistema de automatización de compilación y gestión de dependencias. `Maven` se encargará de gestionar las dependencias del proyecto y de compilarlo. Si utilizas `IntelliJ IDEA`, el IDE se encargará de importar el proyecto y configurar `Maven` automáticamente. Si trabajas desde la línea de comandos, puedes compilar el proyecto con el comando `mvn compile`.

[4] Eric Redmond. *Maven in 5 Minutes – Maven*. 2008. URL: `https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html`

En el caso de que quieras gestionar las dependencias del proyecto manualmente, en la Tabla 1 se detallan las dependencias necesarias para el proyecto

Tabla 1: Dependencias del proyecto.

Dep.	Versión
JDK	22
JUnit	5.8.1

2 Descripción y requisitos del proyecto

El objetivo de este proyecto es desarrollar un sistema en Java que permita gestionar un libro de recetas y planificar comidas semanales. Los estudiantes deberán implementar varias clases y métodos para cumplir con los requisitos especificados.

2.1 Descripción del proyecto

La herramienta a desarrollar es lo que se conoce como una aplicación de línea de comandos (CLI, por sus siglas en inglés). La interfaz de usuario será textual, sin gráficos ni interacción con el ratón. El programa se ejecutará en la terminal de

un sistema operativo, lo que se conoce como Terminal User Interface (TUI)[5].

[5] Aunque pueda parecer algo del pasado, las TUI están a la orden del día. Echa un vistazo al siguiente enlace

Para la realización del proyecto se proporciona un esquema de clases y métodos que los estudiantes deberán completar. Se usarán los conceptos de programación ya trabajados en la asignatura de Fundamentos de Programación.

2.2 Componentes a desarrollar

La arquitectura del sistema se divide en cinco componentes, como se puede observar en la Tabla 2. Cada componente tiene una funcionalidad específica y se encarga de gestionar una parte del sistema.

Tabla 2: Componentes a desarrollar.

#	Componente	Descripción
1	`InterfazUsuario`	Gestiona la interacción con el usuario a través de un menú en consola.
2	`LibroDeRecetas`	Almacena un conjunto de recetas. Permite agregar, buscar y eliminar recetas. Permite guardar y cargar recetas desde archivos.
3	`Receta`	Representa una receta con un nombre, ingredientes e instrucciones. Permite agregar ingredientes e instrucciones. Proporciona métodos para obtener información sobre la receta.
4	`PlanificadorSemanal`	Permite planificar recetas para cada día de la semana así como guardar el plan semanal en un archivo.
5	`Utilidades`	Proporciona métodos de utilidad para la entrada de datos por teclado.

A continuación se detallan los requisitos de cada componente.

2.2.1 La receta

El componente `Receta` representa una receta de cocina. Cada receta tiene un nombre, una lista de ingredientes y una lista de instrucciones. El componente debe proporcionar métodos para agregar ingredientes e instrucciones a la receta, así como para obtener información sobre la receta.

El componente `Receta` es fundamental para el funcionamiento del sistema. Asegúrate de implementar correctamente los métodos necesarios.

El número máximo de ingredientes e instrucciones que puede tener una receta se especifican como párametros de ejecución. Si se intenta agregar más ingredientes o instrucciones, el sistema debe mostrar un mensaje de error[6] y no permitir la adición:

[6] Recuerda que los casos de prueba esperan estos mensajes en concreto, por lo que tienes que ser **preciso** con espacios y signos de puntuación.

```
No se pueden añadir más ingredientes.
No se pueden añadir más instrucciones.
```

Listado 1: Mensajes de error al intentar añadir ingredientes e instrucciones, respectivamente, si se supera el máximo.

La funcionalidad que nos permite agregar ingrediente se encapsula en el método

```
public boolean agregarIngrediente(String ingrediente)
```

que recibe como parámetros el nombre del ingrediente y la cantidad. La funcionalidad de agregar instrucción se encapsula en el método

```
public boolean agregarInstruccion(String ingrediente)
```

que recibe como parámetro la instrucción. Ambos métodos devuelven un valor booleano[7] que indica si la operación se ha realizado correctamente.

[7] `true` si se ha realizado la operación, `false` en otro caso

Una funcionalidad adicional que se pide es la de obtener la información de la receta. Esta funcionalidad se encapsula en el método `public String toString()`, que devuelve una cadena de texto con la información de la receta en el siguiente formato[8]:

[8] Ten **muy en cuenta** los espacios y saltos de línea en la salida

```
Receta: nombre de la receta
Ingredientes:
- Ingrediente 1
- Ingrediente n
Instrucciones:
1. Instrucción 1
2. Instrucción m
```

Listado 2: Formato de representación textual de una receta

Recomendamos utilizar `StringBuilder` en lugar de la concatenación de cadenas de caracteres, especialmente cuando manipulas grandes cantidades de texto o realizas múltiples operaciones de inserción. La superior eficiencia y rendimiento del `StringBuilder` hacen que sea el método más óptimo para estas tareas en programación.

Para guardar y cargar las recetas a partir de un fichero de texto se usa una representación textual **simplificada** de las recetas. Esta representación se genera en el método

```
public String toRawString()
```

y devuelve la información de la receta, incluyendo el nombre, los ingredientes y las instrucciones, en el siguiente formato compacto[9]:

[9] Presta atención a las diferencias con la representación textual completa mostrada en el Listado 2. Fíjate también en el separador de recetas al final del texto.

```
Nombre de la receta
Ingrediente 1
Ingrediente n
INSTRUCCIONES
Instrucción 1
Instrucción m
-----
```

Listado 3: Formato de representación textual simplificada y compacta de una receta

Además de los métodos anteriores, el componente `Receta` debe proporcionar los métodos auxiliares que son necesarios para la correcta implementación de los otros componentes y se detallan a continuación:

```
public Receta(String nombre, int maxIngredientes, int maxInstrucciones)
```

Constructor de la clase que recibe como parámetros el nombre de la receta, el número máximo de ingredientes y el número máximo de instrucciones que puede contener la receta.

```
public String getNombre()
```

Devuelve el nombre de la receta.

```
public int getMaxIngredientes()
```

Devuelve el número máximo de ingredientes que puede contener la receta.

```
public int getMaxInstrucciones()
```

Devuelve el número máximo de instrucciones que puede contener la receta.

```
public List<String> getIngredientes()
```

Devuelve la lista de ingredientes de la receta.

```
public List<String> getInstrucciones()
```

Devuelve la lista de instrucciones de la receta.

```
public boolean ingredientesCompletos()
```

```
public boolean instruccionesCompletas()
```

Devuelve `true` si la receta tiene el número máximo de ingredientes e instrucciones, respectivamente, `false` en otro caso.

```
public int numIngredientes()
```

```
public int numInstrucciones()
```

Devuelve el número de ingredientes e instrucciones de la receta, respectivamente.

2.2.2 El libro de recetas

El componente `LibroDeRecetas` es el encargado de almacenar un conjunto de recetas. Debe proporcionar métodos para agregar, buscar y eliminar recetas, así como para guardar y cargar recetas desde archivos. A continuación, nos centramos en cada una de las funcionalidades que debe implementar este componente.

Agregar receta La funcionalidad que nos permite agregar una receta se encapsula en el método

```
public boolean agregarReceta(Receta receta)
```

que recibe como parámetro[10] la receta a añadir. El método devuelve un valor booleano que indica si la operación se ha realizado correctamente. Si se intenta añadir una receta a un libro que ya está completo, el sistema debe mostrar un mensaje de error[11] y no permitir la adición:

```
1 No se pueden añadir más recetas.
```

Listado 4: Mensaje de error al intentar añadir una receta si el libro está completo.

[10] Será tu responsabilidad comprobar si la receta es un valor nulo.

[11] Recuerda que los casos de prueba esperan estos mensajes en concreto, por lo que tienes que ser **preciso** con espacios y signos de puntuación.

En cualquier caso, el método `agregarReceta` debe devolver `true` si la receta se ha añadido correctamente y `false` en otro caso.

El mecanismo de inserción de recetas en el libro no está especificado, por lo que puedes elegir la estructura de datos que consideres más adecuada para almacenar las recetas. Ten en cuenta que el libro de recetas tiene un tamaño fijo, que se especifica como parámetro de ejecución.

Buscar receta La funcionalidad que nos permite buscar una receta se encapsula en el método

```
public Receta[] buscarRecetaPorNombre(String texto)
```

que recibe como parámetro el texto a buscar en el nombre de las recetas. El método devuelve un array con aquellas recetas que contienen el texto de búsqueda como subcadena[12] del nombre. Si no se encuentra ninguna receta que cumpla con el criterio de búsqueda, el método debe devolver un array vacío.

[12] Puedes usar el método `contains` de la clase `String`

Eliminar receta La funcionalidad que nos permite eliminar una receta se encapsula en el método

```
public void eliminarReceta(Receta seleccionada)
```

que recibe como parámetro la receta a eliminar. Si la receta no se encuentra[13] en el libro, el método no realiza ninguna acción.

[13] Encontrar una receta significa que son **el mismo** objeto.

Guardar y cargar recetas Para guardar y cargar las recetas a partir de un fichero de texto se usa una representación textual **simplificada** de las recetas (ver Listado 3).

El proceso de guardado de las recetas se realiza en el método

```
public void guardarRecetasEnArchivo(String nombreArchivo)
```

que recibe como parámetro el nombre del archivo en el que se guardarán las recetas. Hay que usar el método `toRawString` de la clase `Receta` para obtener la representación textual de cada receta, y escribir esta información en el archivo de texto[14].

[14] En el caso de que el fichero a escribir ya exista, se **sobreescribirá** su contenido

El proceso de carga de las recetas se realiza en el método

```
public void cargarRecetasDeArchivo(String nombreArchivo,
                                   int maxIngredientes,
                                   int maxInstrucciones)
```

que recibe como parámetros el nombre del archivo del que se cargarán las recetas y el número máximo de ingredientes e instrucciones que puede tener cada receta. El método debe leer el contenido del archivo de texto, crear las recetas correspondientes a partir de la información leída y añadirlas al libro de recetas.

Es importante tener en cuenta que el método de carga debe ser capaz de cargar recetas de un archivo que haya sido guardado previamente con el método para escribirlas en el fichero. Además, el método debe ser capaz de cargar recetas de un archivo que contenga un número de recetas superior al tamaño del libro de recetas, en cuyo caso se deben cargar únicamente las primeras recetas que quepan en el libro.

Al tratarse de operaciones con ficheros, es necesario manejar las excepciones que puedan surgir durante la lectura y escritura de los archivos. Sin embargo, no es necesario que se capturen las excepciones en el propio método. En su lugar, se deben lanzar las excepciones para que sean manejadas en el método que llama a `guardarRecetasEnArchivo` y `cargarRecetasDeArchivo`.

Métodos auxiliares Además de los métodos anteriores, el componente `LibroDeRecetas` debe proporcionar los métodos auxiliares que son necesarios para la correcta implementación de los otros componentes y se detallan a continuación:

```
public LibroDeRecetas(int maxRecetasEnLibro)
```

Constructor de la clase que recibe como parámetro el número máximo de recetas que puede contener el libro.

```
public int numRecetas()
```

Devuelve el número de recetas que contiene el libro.

```
public boolean recetasCompletas()
```

Devuelve `true` si el libro de recetas está completo, es decir, si contiene el número máximo de recetas, `false` en otro caso.

2.2.3 El planificador semanal

El componente `PlanificadorSemanal` es el encargado de planificar las comidas para cada día de la semana y de guardar el plan semanal en un archivo. A continuación, nos centramos en cada una de las funcionalidades que debe implementar este componente.

Planificar comidas La funcionalidad que nos permite planificar una comida para un día de la semana se encapsula en el método

```
public void agregarComida(int dia, Receta receta)
```

que recibe como parámetros el día de la semana en el que se planifica la comida y la receta que se va a añadir. El día de la semana se representa como un número entero, siendo 0 el lunes y 6 el domingo (ver Tabla 3). Si ya hay una receta planificada para el día especificado, la receta anterior se reemplaza por la nueva.

Tabla 3: Días de la semana representados como números enteros.

#	Día
0	Lunes
1	Martes
2	Miércoles
3	Jueves
4	Viernes
5	Sábado
6	Domingo

Representación textual del plan semanal Para mostrar el plan semanal por pantalla, se debe proporcionar una representación textual del mismo. Esta representación debe mostrar las recetas planificadas para cada día de la semana. El método que genera esta representación textual es

```
public String toString()
```

que devuelve una cadena de texto con el plan semanal en el siguiente formato:

```
1  ------------------------------------------------------------------------------------------------------------------------
2     Lunes              Martes             Miércoles          Jueves             Viernes            Sábado             Domingo
3  ------------------------------------------------------------------------------------------------------------------------
4     Tortilla de Patatas                                      Gazpacho Andaluz                                         Tarta de Manzana
5  ------------------------------------------------------------------------------------------------------------------------
```

Listado 5: Formato de representación textual del plan semanal (ejemplo)

Observa que la representación textual del plan semanal muestra las recetas planificadas para cada día de la semana. Si no hay ninguna receta planificada para un día en concreto, el hueco se deja vacío. La representación textual del plan semanal debe incluir un encabezado con los días de la semana y un separador entre el encabezado y las recetas planificadas. Además, cada receta planificada debe estar alineada con el día correspondiente.

Guardar plan semanal Para guardar el plan semanal en un archivo de texto se usa una representación textual **simplificada** del plan semanal. En este caso no se genera la representación simplificada en un método aparte, sino que se realiza todo en el método de guardado:

```
public void guardarPlanEnArchivo(String nombreArchivo)
```

que recibe como parámetro el nombre del archivo en el que se guardará el plan semanal. El método debe escribir la representación textual del plan semanal en el archivo de texto[15]. El formato es el siguiente:

[15] En el caso de que el fichero a escribir ya exista, se **sobreescribirá** su contenido

```
Lunes: Tortilla de Patatas
Martes: ---
Miércoles: ---
Jueves: Gazpacho Andaluz
Viernes: ---
Sábado: ---
Domingo: Tarta de Manzana
```

Listado 6: Formato de representación textual simplificada y compacta del plan semanal (ejemplo)

Observa que la representación textual del plan semanal muestra las recetas planificadas para cada día de la semana. Si no hay ninguna receta planificada para un día en concreto, se muestran tres guiones `---`. La representación textual del plan semanal debe incluir el nombre del día y la receta planificada, separados por dos puntos.

2.2.4 Utilidades

El componente `Utilidades` proporciona métodos de utilidad para la entrada de datos por teclado. En concreto, se deben implementar los siguientes métodos:

```
public static String leerCadena(Scanner teclado, String s)
```

Método que muestra el mensaje `s` por pantalla y lee una cadena de texto introducida por el usuario. El método devuelve la cadena de texto leída.

```
public static int leerNumero(Scanner teclado,
                             String mensaje,
                             int minimo,
                             int maximo)
```

Método que muestra el mensaje `mensaje` por pantalla y lee un número entero introducido por el usuario. El método debe comprobar que el número introducido está en el rango $[\text{minimo}, \text{maximo}]$ y, si no es así, volver a solicitar el número. El método devuelve el número entero leído.

```
public static int leerDiaDeLaSemana(Scanner teclado,
                                     String mensaje)
```

Método que muestra el mensaje `mensaje` por pantalla y lee un caracter introducido por el usuario que representa un día de la semana. El método debe comprobar que el caracter introducido se corresponde con un día de la semana (ver Tabla 4) y, si no es así, volver a solicitar el caracter. El método devuelve el número entero correspondiente al día que ha escrito el usuario (ver Tabla 3).

```
public static int diaSemanaAPosicion(String dia)
```

Método que recibe como parámetro el caracter de un día de la semana (ver Tabla 4) y devuelve el número entero correspondiente a ese día (ver Tabla 3).

```
public static String posicionADiaSemana(int pos)
```

Método que recibe como parámetro el número entero de un día de la semana (ver Tabla 3) y devuelve **el nombre completo** del día que corresponde a esa posición (ver Tabla 4).

Tabla 4: Días de la semana representados como caracteres.

Char	Día
L	Lunes
M	Martes
X	Miércoles
J	Jueves
V	Viernes
S	Sábado
D	Domingo

2.2.5 La interfaz de usuario

La interfaz de usuario es el componente que se encarga de gestionar la interacción con el usuario. Debe mostrar un menú con las opciones disponibles y permitir al usuario seleccionar una de ellas. Las opciones disponibles son las siguientes:

1. Agregar Receta
2. Consultar/Editar Receta
3. Planificar Comidas
4. Guardar Recetas
5. Cargar Recetas
6. Guardar Plan Semanal
7. Salir

El componente se encargará de llamar a los métodos correspondientes de los otros componentes para realizar las acciones solicitadas por el usuario, además de encargarse de la entrada y salida de los datos necesarios para realizar dichas acciones.

Por ejemplo, si el usuario selecciona la opción de agregar receta, la interfaz de usuario se encargará de solicitar la información de la nueva receta al usuario y llamará al método correspondiente del componente que gestiona el libro de recetas.

Una vez realizada la acción solicitada por el usuario, la interfaz de usuario debe volver a mostrar el menú de opciones. El programa debe seguir ejecutándose hasta que el usuario seleccione la opción de salir. Este bucle de ejecución se conoce como el bucle principal del programa, y se define en el método

```
private void menuPrincipal(Scanner scanner)
```

A continuación se puede ver un ejemplo de cómo se debe mostrar el menú de opciones al usuario (observa que el menú se muestra en la consola, que el usuario debe introducir un número para seleccionar una opción y que se muestra un símbolo[16] para representar los espacios en blanco ␣):

```
---␣Menú␣Principal␣---
1.␣Agregar␣Receta
2.␣Consultar/Editar␣Receta
3.␣Planificar␣Comidas
4.␣Guardar␣Recetas
5.␣Cargar␣Recetas
6.␣Guardar␣Plan␣Semanal
7.␣Salir

Elige␣una␣opción:␣
```

Listado 7: Menu principal y petición de acción al usuario

[16] Se muestran los espacios en blanco de manera explícita usando el símbolo concreto ␣ para que puedas generar una salida exactamente igual.

A continuación, se detalla la funcionalidad que debe implementar la interfaz de usuario para cada una de las opciones del menú.

Agregar Receta La funcionalidad que permite agregar una receta se implementa en el método

```
private void agregarReceta(Scanner scanner)
```

que recibe como parámetro un objeto de la clase `Scanner` para leer la entrada del usuario. El método debe solicitar al usuario el nombre de la receta. A continuación, debe solicitar al usuario que introduzca los ingredientes e instrucciones de la receta. Una vez introducidos los datos, el método debe crear un objeto de la clase `Receta` con la información proporcionada y añadir la receta al libro de recetas. Si la receta se añade correctamente, se debe mostrar un mensaje de éxito[17]:

[17] Recuerda que los casos de prueba esperan estos mensajes en concreto, por lo que tienes que ser **preciso** con espacios y signos de puntuación.

```
Nombre de la receta: Gazpacho
Introduce los ingredientes (una línea por ingrediente, escribe 'fin' para terminar):
```

```
Tomate
Pepino
Aceite
Ajo
Agua
fin
Introduce las instrucciones (una línea por instrucción, escribe 'fin' para terminar):
Echar todos los ingredientes a la batidora
Servir muy frío
fin
¡Receta agregada exitosamente!
```

Listado 8: Ejemplo de funcionalidad Agregar receta

Fíjate en que el usuario debe introducir los ingredientes e instrucciones de la receta, una línea por cada uno, y escribir `fin` para indicar que ha terminado de introducir los datos. En el caso de que la receta no se haya podido añadir al libro, se debe mostrar el error `No se pudo añadir la receta.` por pantalla. En cualquier caso, el método debe volver al menú principal una vez finalizada la acción.

Consultar/Editar Receta La funcionalidad que permite consultar y editar una receta se implementa en el método

```
private void consultarReceta(Scanner scanner)
```

que recibe como parámetro un objeto de la clase `Scanner` para leer la entrada del usuario. El método debe solicitar al usuario el texto a buscar dentro del nombre de las recetas que desea consultar o editar[18]. Una vez realizada la consulta, se le muestra al usuario el listado de recetas resultante y se le pide que seleccione una de ellas. A continuación, se le muestra la información de la receta seleccionada y se le muestra el menú de edición de receta, que permite añadir un ingrediente, una instrucción o eliminar la receta:

[18] Si el usuario introduce `-FIN-` se cancela la búsqueda.

```
Introduce el texto de la receta a buscar (-FIN- para volver): lla
Recetas encontradas:
1. Tortilla de Patatas
2. Paella Valenciana
Elige una receta: 1
Receta: Tortilla de Patatas
Ingredientes:
- Huevos
- Patatas
- Aceite
```

```
- Cebolla
Instrucciones:
1. Pelar y cortar las patatas.
2. Freírlas en aceite.
3. Añadir los huevos batidos.
4. Cocinar a fuego medio.

1. Añadir ingrediente
2. Añadir instrucción
3. Eliminar receta
4. Volver
Elige una opción: 1
Introduce el ingrediente a añadir: Pimienta
```

Listado 9: Ejemplo de funcionalidad Consultar/Editar receta

Como se puede observar, el usuario puede seleccionar una receta de la lista de recetas encontradas y, a continuación, puede añadir un ingrediente, una instrucción o eliminar la receta. Una vez realizada la acción solicitada por el usuario, el método debe volver al menú principal. Las acciones de añadir ingrediente y añadir instrucción deben seguir el mismo procedimiento que en la funcionalidad de agregar receta, incluyendo mensajes de error en caso de no poder realizarse (ver Listado 1).

La funcionalidad de este método está, a su vez, encapsulada en otros métodos auxiliares que se detallan a continuación:

```
private Receta buscarRecetaPorNombre(Scanner scanner)
```

Método que recibe como parámetro un objeto de la clase `Scanner` para leer la entrada del usuario y devuelve la receta seleccionada por el usuario. El método debe solicitar al usuario el texto a buscar dentro del nombre de las recetas que desea consultar o editar y mostrar el listado de recetas resultante[19]. A continuación, debe solicitar al usuario que seleccione una de las recetas encontradas y devolver la receta seleccionada. La selección de las recetas también está encapsulada en el método

[19] Si la búsqueda no arroja resultados, hay que solicitar al usuario un nuevo término de búsuqeda.

```
private Receta seleccionarReceta(Scanner scanner, Receta[] recetas))
```

que recibe como parámetro un objeto de la clase `Scanner` para leer la entrada del usuario y un array de recetas. El método debe mostrar al usuario el listado de recetas y solicitar al usuario que seleccione una de ellas, mediante el número de orden en el listado (ver Listado 9). A continuación, debe de-

volver la receta seleccionada por el usuario.

Por último, el método

```
private void editarReceta(Scanner scanner, Receta seleccionada)
```

que recibe como parámetros un objeto de la clase `Scanner` para leer la entrada del usuario y la receta seleccionada por el usuario. El método debe mostrar la información de la receta seleccionada y el menú de edición de receta, que permite añadir un ingrediente, una instrucción o eliminar la receta (ver Listado 9). En el caso de eliminar una receta, se debe mostrar el mensaje `Receta eliminada.` por pantalla:

```
Introduce el texto de la receta a buscar (-FIN- para volver): tortilla
Recetas encontradas:
1. Tortilla de Patatas
Elige una receta: 1
Receta: Tortilla de Patatas
Ingredientes:
- Huevos
- Patatas
- Aceite
- Cebolla
- Pimienta
Instrucciones:
1. Pelar y cortar las patatas.
2. Freírlas en aceite.
3. Añadir los huevos batidos.
4. Cocinar a fuego medio.

1. Añadir ingrediente
2. Añadir instrucción
3. Eliminar receta
4. Volver
Elige una opción: 3
Receta eliminada.
```

Listado 10: Ejemplo de eliminar una receta

Planificar Comidas La funcionalidad que permite planificar las comidas para cada día de la semana se implementa en el método

```
private void planificarComidas(Scanner scanner)
```

que recibe como parámetro un objeto de la clase `Scanner` para leer la entrada del usuario. El método debe, en primer lugar,

imprimir pon pantalla la planificación actual de la semana. Después, debe solicitar al usuario el día de la semana en el que desea planificar la comida y la receta que desea añadir. Una vez introducidos los datos, el método debe planificar la comida para el día especificado y mostrar un mensaje de éxito:

```
Planificación de comidas para la semana:
------------------------------------------------------------------------------
 Lunes      Martes     Miércoles  Jueves     Viernes    Sábado     Domingo
------------------------------------------------------------------------------

------------------------------------------------------------------------------


Introduce el día de la semana (L, M, X, J, V, S, D): M
Introduce el texto de la receta a buscar (-FIN- para volver): tor
Recetas encontradas:
1. Tortilla de Patatas
Elige una receta: 1
Receta planificada para Martes
```

Listado 11: Ejemplo de funcionalidad Planificar comidas

Para pedir al usuario el día de la semana en el que desea planificar la comida, se debe utilizar el componente `Utilidades` (ver section 2.2.4). En cualquier caso, el método debe volver al menú principal una vez finalizada la acción.

Guardar Recetas La funcionalidad que permite guardar las recetas en un archivo de texto se implementa en el método

```
private void guardarRecetas(Scanner scanner)
```

que recibe como parámetro un objeto de la clase `Scanner` para leer la entrada del usuario. El método debe solicitar al usuario la ruta del archivo en el que desea guardar las recetas y llamar al método correspondiente del componente que gestiona el libro de recetas (ver section 2.2.2). Si las recetas se guardan correctamente, se debe mostrar un mensaje[20] de éxito:

[20] En caso de error, el mensaje a mostrar será: *Error al guardar el archivo.*

```
Introduce el nombre del archivo donde guardar las recetas: asdfaa.txt
Recetas guardadas en asdfaa.txt
```

Listado 12: Ejemplo de funcionalidad Guardar recetas

En cualquier caso, el método debe volver al menú principal una vez finalizada la acción.

Cargar Recetas La funcionalidad que permite cargar las recetas desde un archivo de texto se implementa en el método

```
private void cargarRecetas(Scanner scanner)
```

que recibe como parámetro un objeto de la clase `Scanner` para leer la entrada del usuario. El método debe solicitar al usuario la ruta del archivo del que desea cargar las recetas y llamar al método correspondiente del componente que gestiona el libro de recetas (ver section 2.2.2). Si las recetas se cargan correctamente, se debe mostrar un mensaje[21] de éxito:

[21] En caso de error, el mensaje a mostrar será: *Error al cargar el archivo.*

```
Introduce la ruta del archivo de donde cargar las recetas: aaaa.txt
Recetas cargadas desde aaaa.txt
```

Listado 13: Ejemplo de funcionalidad Cargar recetas

En cualquier caso, el método debe volver al menú principal una vez finalizada la acción.

Guardar Plan Semanal La funcionalidad que permite guardar el plan semanal en un archivo de texto se implementa en el método

```
private void guardarPlanSemanal(Scanner scanner)
```

que recibe como parámetro un objeto de la clase `Scanner` para leer la entrada del usuario. El método debe solicitar al usuario la ruta del archivo en el que desea guardar el plan semanal y llamar al método correspondiente del componente que gestiona el planificador semanal (ver section 2.2.3). Si el plan se guarda correctamente, se debe mostrar un mensaje[22] de éxito:

[22] En caso de error, el mensaje a mostrar será: *Error al guardar el archivo.*

```
Introduce el nombre del archivo donde guardar el plan semanal: plan1.txt
Plan semanal guardado en plan1.txt
```

Listado 14: Ejemplo de funcionalidad Guardar plan semanal

En cualquier caso, el método debe volver al menú principal una vez finalizada la acción.

2.3 La función principal Main

La función principal `Main` es la encargada de inicializar los componentes de la aplicación y de gestionar la ejecución del

programa. También se encarga de recibir los argumentos de la línea de comandos que configuran el funcionamiento de la aplicación. En concreto, la función principal debe recibir los siguientes argumentos:

Tabla 5: Argumentos de la línea de comandos (y su orden)

#	Parámetro	Descripción
1	`maxIngredientesPorReceta`	Número máximo de ingredientes que puede contener una receta.
2	`maxInstruccionesPorReceta`	Número máximo de instrucciones que puede contener una receta.
3	`maxRecetasEnLibro`	Número máximo de recetas que puede contener el libro de recetas.
4	`nombreArchivoRecetas`	*(Opcional)* Nombre del archivo de texto donde se guardarán las recetas.

Importancia de los argumentos en la línea de comandos
Los argumentos proporcionados mediante la línea de comandos permiten a los programas recibir configuraciones o datos de entrada directamente al momento de su ejecución, sin necesidad de modificar el código fuente o interactuar manualmente con el usuario tras iniciarlo. Esto tiene varias ventajas clave:

1. **Flexibilidad y reutilización:** Usar argumentos en la línea de comandos hace que los programas sean más versátiles, ya que pueden adaptarse fácilmente a diferentes escenarios sin necesidad de cambios en el código. Por ejemplo, un programa que gestione recetas puede aceptar el nombre de una receta como argumento para buscarla automáticamente al iniciar.
2. **Automatización:** Los argumentos facilitan la integración de programas en scripts o procesos automatizados. Al permitir que los datos se pasen directamente al programa, se elimina la necesidad de interacción manual, lo cual es ideal para entornos de producción o tareas repetitivas.
3. **Cumplimiento de buenas prácticas:** Siguiendo estándares comunes, como los de herramientas Unix, el uso de argumentos fomenta una interfaz coherente y predecible para los usuarios, lo cual es especialmente importante en software diseñado para otros desarrolladores.

4. **Simulación de escenarios reales:** Enseñar a usar argumentos en programas educativos permite a los estudiantes familiarizarse con prácticas comunes en la industria, donde los sistemas suelen depender de configuraciones y parámetros proporcionados al momento de la ejecución.
5. **Claridad y separación de responsabilidades:** Pasar argumentos en la línea de comandos promueve la separación entre la lógica de la aplicación y su configuración. Esto facilita el desarrollo, el mantenimiento y las pruebas, ya que los valores específicos pueden cambiarse sin necesidad de alterar el código fuente.

Fíjate que el argumento `nombreArchivoRecetas` es opcional, lo que significa que el programa puede ejecutarse sin proporcionar este argumento. En caso de que se proporcione, el programa debe cargar las recetas desde el archivo especificado al inicio. Si no se proporciona, el programa debe comenzar sin recetas cargadas.

La función principal debe inicializar los componentes de la aplicación y ejecutar el bucle principal del programa, que se define en el método `menuPrincipal` de la interfaz de usuario (ver section 2.2.5). La función principal debe gestionar las excepciones que puedan surgir durante la ejecución del programa y mostrar un mensaje de error en caso de que ocurra una excepción no controlada.

2.4 Casos de prueba

Para que puedas comprobar que tu implementación es correcta, se proporcionan una serie de casos de prueba que puedes utilizar para verificar el funcionamiento de tu programa. Además, permite emplear la metodología de desarrollo dirigido por pruebas[23] (*Test-Driven Development, TDD*) para implementar cada funcionalidad de forma incremental y asegurarte de que cada parte de tu programa funciona correctamente antes de pasar a la siguiente.

[23] Dave Astels. *Test Driven development: A Practical Guide*. Prentice Hall Professional Technical Reference. DOI: 10.5555/864016

Aunque implementar y diseñar los casos de prueba queda fuera del alcance de esta asignatura, es importante que conozcas su utilidad y cómo se utilizan. En este caso, los casos de prueba se especifican usando código Java en ficheros que contienen una serie de comandos y mensajes esperados que se pueden utilizar para verificar el correcto funcionamiento de tu programa.

3 Criterios de evaluación

Para llevar a cabo la evaluación de la práctica se tendrán en cuenta varios aspectos complementarios que se detallan a continuación.

3.1 Casos de prueba

Se proporcionan una serie de casos de prueba que se utilizarán para comprobar el correcto funcionamiento de tu programa. Estos casos de prueba se han diseñado para cubrir diferentes situaciones y comprobar que tu implementación es correcta. Es importante que tu programa pase todos los casos de prueba para asegurar que cumple con los requisitos y funcionalidades solicitadas.

! Importante Durante la evaluación, se utilizarán casos de prueba adicionales que no se proporcionan en el enunciado. Estos casos de prueba adicionales se utilizarán para comprobar que tu implementación es correcta y cumple con los requisitos solicitados.

El hecho de que tu programa pase los casos de prueba es condición necesaria, pero no suficiente, para obtener una calificación alta. Además de pasar los casos de prueba, tu implementación se evaluará con el resto de criterios detallados en esta sección.

3.2 Diseño y estructura del código

Se valorará la claridad, organización y estructura del código fuente de tu programa. Es importante que tu implementación sea fácil de leer, comprender y mantener. Se valorará positivamente la correcta nomenclatura de variables y métodos, la división en métodos y clases coherentes y la reutilización de código.

Se valorará el uso de buenas prácticas de programación y el cumplimiento de los principios de diseño y desarrollo de software. Es importante que tu implementación siga las recomendaciones y buenas prácticas de programación en Java, así como que siga los principios de diseño y desarrollo de software.

! Importante Se valorará negativamente:

- Código duplicado o redundante.
- Métodos o clases demasiado largos o complejos.
- Falta de modularidad y reutilización de código.
- Uso incorrecto de estructuras de control: bucles y condicionales.
- Romper la ejecución de un bucle con `break` o `return` en lugar de usar

una condición.
- Uso de más de un `return` en un método.
- Mal uso de excepciones.

3.3 Documentación y comentarios

Se valorará la calidad de la documentación y los comentarios en el código fuente de tu programa. Es importante que tu implementación esté correctamente documentada y que los comentarios sean claros y concisos. Se valorará positivamente la presencia de comentarios que expliquen el propósito de las clases y métodos, así como el significado de variables y parámetros.

Al igual que en el diseño y estructura del código, se valorará negativamente la falta de comentarios y documentación, así como la presencia de comentarios innecesarios o confusos.

3.4 Errores graves

A continuación, se detallan una serie de errores graves que, de estar presentes en tu implementación, **supondrán un suspenso** en la calificación final de la práctica. Es importante que tengas en cuenta estos errores y los evites en tu implementación:

- No compilar: tu programa no compila correctamente y no se puede ejecutar.
- Argumentos incorrectos: tu programa no recibe los argumentos de la línea de comandos correctamente.
- Plagio: tu implementación es sospechosa de contener código copiado de otras fuentes.
- Código malicioso: tu implementación contiene código malicioso o perjudicial.

Referencias

[1] Dave Astels. *Test Driven development: A Practical Guide*. Prentice Hall Professional Technical Reference. DOI: 10.5555/864016.

[2] *Crear un repositorio desde una plantilla*. GitHub Docs. 2024. URL: https://docs.github.com/es/repositories/creating-and-managing-repositories/creating-a-repository-from-a-template.

[3] *Licencias educativas gratuitas. Asistencia a la comunidad*. JetBrains. 2024. URL: https://www.jetbrains.com/es-es/community/education/#students.

[4] *Plantilla de Proyecto de Taller de Programación. Repositorio GitHub*. 2024. URL: https://github.com/laracabrera/tp-ordinaria-2425.

[5] Eric Redmond. *Maven in 5 Minutes – Maven*. 2008. URL: https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html.