

Programación Java SE 7

Guía de actividades

D67238CS20

Edición 2.0

Noviembre de 2011

D81765

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Exención de responsabilidad

Este documento contiene información propiedad de Oracle Corporation y se encuentra protegido por el copyright y otras leyes sobre la propiedad intelectual. Usted solo podrá realizar copias o imprimir este documento para uso exclusivo por usted en los cursos de formación de Oracle. Este documento no podrá ser modificado ni alterado en modo alguno. Salvo que la legislación del copyright lo considere un uso excusable o legal o "fair use", no podrá utilizar, compartir, descargar, cargar, copiar, imprimir, mostrar, representar, reproducir, publicar, conceder licencias, enviar, transmitir ni distribuir este documento total ni parcialmente sin autorización expresa por parte de Oracle.

La información contenida en este documento puede someterse a modificaciones sin previo aviso. Si detecta cualquier problema en el documento, le agradeceremos que nos lo comunique por escrito a: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 EE. UU. No se garantiza que este documento se encuentre libre de errores.

Aviso sobre restricción de derechos

Si este software o la documentación relacionada se entrega al Gobierno de EE. UU. o a cualquier entidad que adquiera licencias en nombre del Gobierno de EE. UU. se aplicará la siguiente disposición:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Disposición de marca comercial registrada

Oracle y Java son marcas comerciales registradas de Oracle y/o sus filiales. Todos los demás nombres pueden ser marcas comerciales de sus respectivos propietarios.

Authors

Michael Williams, Tom McGinn, Matt Heimer

Technical Contributors and Reviewers

Lee Klement, Steve Watts, Brian Earl, Vasily Strelnikov, Andy Smith, Nancy K.A.N, Chris Lamb, Todd Lowry, Ionut Radu, Joe Darcy, Brian Goetz, Alan Bateman y David Holmes

Este libro se ha publicado con la ayuda de: **Oracle Tutor**

Tabla de Contenidos

Prácticas de la lección 1	1-1
Prácticas de la lección1: Visión general	1-2
Práctica 1-1: Verificación de la instalación del software	1-3
Práctica 1-2: Instalación del software	1-5
Práctica 1-3: Configuración de NetBeans 7.0.1 para utilizar JDK 7	1-7
Prácticas de la lección 2: Sintaxis Java y revisión de clases	2-1
Prácticas de la lección 2: Visión general	2-2
Práctica 2-1: Nivel de resumen: Creación de clases Java	2-3
Práctica 2-1: Nivel detallado: Creación de clases Java	2-5
Prácticas de la lección 3: Encapsulación y creación de subclases	3-1
Prácticas de la lección 3: Visión general	3-2
Práctica 3-1: Nivel de resumen: Creación de subclases	3-3
Práctica 3-1: Nivel detallado: Creación de subclases	3-6
(Opcional) Práctica 3-2: Adición de una clase Staff a una clase Manager	3-11
Prácticas de la lección 4: Diseño de clases Java	4-1
Prácticas de la lección 4	4-2
Práctica 4-1: Nivel de resumen: Sustitución de métodos y aplicación de polimorfismo	4-3
Práctica 4-1: Nivel detallado: Sustitución de métodos y aplicación de polimorfismo	4-6
Prácticas de la lección 5: Diseño de clases avanzadas	5-1
Prácticas de la lección 5: Visión general	5-2
Práctica 5-1: Nivel de resumen: Aplicación de la palabra clave abstract	5-3
Práctica 5-1: Nivel detallado: Aplicación de la palabra clave abstract	5-7
Práctica 5-2: Nivel de resumen: Aplicación del patrón de diseño Singleton	5-12
Práctica 5-2: Nivel detallado: Aplicación del patrón de diseño Singleton	5-14
(Opcional) Práctica 5-3: Uso de enumeraciones Java	5-16
(Opcional) Práctica 5-4: Reconocimiento de clases anidadas	5-18
(Opcional) Solución 5-4: Reconocimiento de clases anidadas	5-19
Prácticas de la lección 6: Herencia con interfaces Java	6-1
Prácticas de la lección 6: Visión general	6-2
Práctica 6-1: Nivel de resumen: Implantación de una interfaz	6-3
Práctica 6-1: Nivel detallado: Implantación de una interfaz	6-6
Práctica 6-2: Nivel de resumen: Aplicación del patrón DAO	6-11
Práctica 6-2: Nivel detallado: Aplicación del patrón DAO	6-14
(Opcional) Práctica 6-3: Implantación de la composición	6-19
Prácticas de la lección 7: Elementos genéricos y recopilaciones	7-1
Prácticas de la lección 7: Visión general	7-2
Práctica 7-1: Nivel de resumen: Recuento de números de artículo mediante elementos HashMap	7-3
Práctica 7-1: Nivel detallado: Recuento de números de artículo mediante elementos HashMap	7-5
Práctica 7-2: Nivel de resumen: Coincidencia de paréntesis mediante Deque	7-7
Práctica 7-2: Nivel detallado: Coincidencia de paréntesis mediante Deque	7-8
Práctica 7-3: Nivel de resumen: Recuento de inventario y ordenación con elementos Comparator	7-10
Práctica 7-3: Nivel detallado: Recuento de inventario y ordenación con elementos Comparator	7-13

Prácticas de la lección 8: Procesamiento de cadenas.....	8-1
Prácticas de la lección 8: Visión general.....	8-2
Práctica 8-1: Nivel de resumen: Análisis de texto con <code>split()</code>	8-3
Práctica 8-1: Nivel detallado: Análisis de texto con <code>split()</code>	8-5
Práctica 8-2: Nivel de resumen: Creación de un programa de búsqueda de expresiones regulares	8-8
Práctica 8-2: Nivel detallado: Creación de un programa de búsqueda de expresiones regulares.....	8-10
Práctica 8-3: Nivel de resumen: Transformación de HTML mediante expresiones regulares.....	8-12
Práctica 8-3: Nivel detallado: Transformación de HTML mediante expresiones regulares	8-14
Prácticas de la lección 9: Excepciones y afirmaciones.....	9-1
Prácticas de la lección 9: Visión general.....	9-2
Práctica 9-1: Nivel de resumen: Captura de excepciones	9-3
Práctica 9-1: Nivel detallado: Captura de excepciones.....	9-6
Práctica 9-2: Nivel de resumen: Ampliación de <code>Exception</code>	9-10
Práctica 9-2: Nivel detallado: Ampliación de <code>Exception</code>	9-14
Prácticas de la lección 10: Conceptos fundamentales de E/S en Java.....	10-1
Prácticas de la lección 10: Visión general.....	10-2
Práctica 10-1: Nivel de resumen: Escritura de una aplicación simple de E/S de la consola	10-3
Práctica 10-1: Nivel detallado: Escritura de una aplicación simple de E/S de la consola	10-5
Práctica 10-2: Nivel de resumen: Serialización y anulación de la serialización de un objeto <code>ShoppingCart</code>	10-8
Práctica 10-2: Nivel detallado: Serialización y anulación de la serialización de un objeto <code>ShoppingCart</code>	10-11
Prácticas de la lección 11: E/S de archivos Java (NIO.2)	11-1
Prácticas de la lección 11: Visión general.....	11-2
Práctica 11-1: Nivel de resumen: Escritura de una aplicación de fusión de archivos	11-3
Práctica 11-1: Nivel detallado: Escritura de una aplicación de fusión de archivos.....	11-6
Práctica 11-2: Nivel de resumen: Copia recursiva	11-10
Práctica 11-2: Nivel detallado: Copia recursiva	11-12
(Opcional) Práctica 11-3: Nivel de resumen: Uso de <code>PathMatcher</code> para realizar una supresión recursiva	11-15
(Opcional) Práctica 11-3: Nivel detallado: Uso de <code>PathMatcher</code> para realizar una supresión recursiva	11-17
Prácticas de la lección 12: Thread	12-1
Prácticas de la lección 12: Visión general.....	12-2
Práctica 12-1: Nivel de resumen: Sincronización de acceso a datos compartidos	12-3
Práctica 12-1: Nivel detallado: Sincronización de acceso a datos compartidos	12-6
Práctica 12-2: Nivel de resumen: Implantación de un programa multithread	12-10
Práctica 12-2: Nivel detallado: Implantación de un programa multithread.....	12-12
Prácticas de la lección 13: Simultaneidad.....	13-1
Prácticas de la lección 13: Visión general.....	13-2
(Opcional) Práctica 13-1: Uso del paquete <code>java.util.concurrent</code>	13-3
(Opcional) Práctica 13-2: Uso del marco Fork-Join	13-5
Prácticas de la lección 14: Creación de aplicaciones de base de datos con JDBC	14-1
Prácticas de la lección 14: Visión general.....	14-2
Práctica 14-1: Nivel de resumen: Trabajo con la base de datos Derby y JDBC.....	14-3
Práctica 14-1: Nivel detallado: Trabajo con la base de datos Derby y JDBC	14-5
Práctica 14-2: Nivel de resumen: Uso del patrón de objeto de acceso a datos	14-7
Práctica 14-2: Nivel detallado: Uso del patrón de objeto de acceso a datos.....	14-10

Prácticas de la lección 15: Localización.....	15-1
Prácticas de la lección 15: Visión general.....	15-2
Práctica 15-1: Nivel de resumen: Creación de una aplicación de fecha localizada	15-3
Práctica 15-1: Nivel detallado: Creación de una aplicación de fecha localizada	15-5
Práctica 15-2: Nivel de resumen: Localización de una aplicación JDBC (opcional).....	15-9

Carlos Jaramillo (cajaramillo@gmail.com) has a non-transferable license to use this Student Guide.

Carlos Jaramillo (cajaramillo@gmail.com) has a non-transferable
license to use this Student Guide.

Prácticas de la lección 1

Capítulo 1

Prácticas de la lección 1: Visión general

Visión general de las prácticas

En estas prácticas se aborda la configuración de un entorno de desarrollo para Java SE 7. Estas prácticas no se deben realizar a menos que así se lo haya indicado el instructor.

Práctica 1-1: Verificación de la instalación del software

Visión general

En esta práctica, verificará la instalación del software necesario para realizar un desarrollo de software con Java 7. Si la verificación falla, seguirá con las demás prácticas.

Supuestos

El instructor le ha pedido que realice estos pasos.

Resumen

Se le ha proporcionado un sistema de computadora que se usará para el desarrollo del software con Java SE 7. Debe asegurarse de que esté instalado Java 7 SE Development Kit, NetBeans IDE 7.0.1 y que NetBeans IDE esté correctamente configurado para usar JDK 7.

Tareas

1. Abra una ventana de comandos o de terminal.

Nota: si utiliza Windows, realice los siguientes pasos para abrir una ventana de comandos:

- a. Haga clic en el botón Inicio.
- b. Haga clic en Ejecutar.
- c. Escriba `cmd` en el cuadro de diálogo Ejecutar y haga clic en el botón Aceptar.

2. Ejecute el comando `java -version`. Así se verifica que se haya instalado un *JRE*; de esta forma, no se verifica que esté instalado el *JDK*.

- a. Verifique que la salida del comando `java -version` coincide con la siguiente salida de ejemplo. Para máquinas de 64 bits, la salida debe ser:

```
java version "1.7.0"
Java(TM) SE Runtime Environment (build 1.7.0-b147)
Java HotSpot(TM) 64-Bit Server VM (build 21.0-b17, mixed mode)
```

Para máquinas de 32 bits, la salida debe ser:

```
java version "1.7.0"
Java(TM) SE Runtime Environment (build 1.7.0-b147)
Java HotSpot(TM) Client VM (build 21.0-b17, mixed mode, sharing)
```

- b. Si aparece otro número de versión o un mensaje de error, puede que exista uno de los siguientes problemas:
 - El JRE/JDK no está instalado.
 - El comando `java` no está incluido en la variable `PATH`.
 - Está instalada la versión incorrecta de JRE/JDK.
 - Hay instalados varios JRE/JDK.

Nota: para excluir una variable de entorno `PATH` incorrecta como el posible motivo de una versión de Java incorrecta o no reconocida, puede intentar localizar la ruta de acceso al JDK y ejecutar `java -version` mediante una ruta de acceso totalmente cualificada. Por ejemplo:

```
"C:\Program Files\Java\jdk1.7.0\bin\java.exe" -version
```

3. Ejecute el comando `javac -version`. De esta forma, se verifica que se ha instalado un JDK.

- a. Verifique que la salida del comando `javac -version` coincide con la siguiente salida de ejemplo:

```
javac 1.7.0
```

- b. Si aparece otro número de versión o un mensaje de error, puede que exista uno de los siguientes problemas:

- El JDK no está instalado.
- El comando `javac` no está incluido en la variable `PATH`.
- Está instalada la versión incorrecta de JDK.
- Hay instalados varios JDK.

Nota: es muy habitual que un directorio que contenga `javac` no aparezca en la variable de entorno `PATH`. No tendrá que modificar la variable `PATH` para que la mayoría de los IDE funcionen, sino que tendrá que localizar la ruta de acceso al JDK y ejecutar `javac -version` mediante una ruta de acceso totalmente cualificada para verificar la presencia y la versión del JDK. Por ejemplo:

```
"C:\Program Files\Java\jdk1.7.0\bin\javac.exe" -version
```

4. Inicie NetBeans IDE y verifique el número de versión del JDK que usa el IDE.

- a. Al instalarse NetBeans, se incluye un menú en el menú Inicio. Busque el acceso directo a NetBeans IDE 7.0.1 en el menú Inicio y haga clic en él.
- b. En NetBeans, haga clic en el menú Help y, a continuación, en About.
- c. En el cuadro de diálogo About aparecen los números de versión que se usan tanto de NetBeans como de JDK. Para máquinas de 64 bits, debe aparecer:

```
Product Version: NetBeans IDE 7.0.1 (Build 201107282000)
Java: 1.7.0; Java HotSpot(TM) 64-Bit Server VM 21.0-b17
```

Para máquinas de 32 bits, debe aparecer:

```
Product Version: NetBeans IDE 7.0.1 (Build 201107282000)
Java: 1.7.0; Java HotSpot(TM) Client VM 21.0-b17
```

Nota: incluso aunque se haya detectado JDK7 en un paso anterior, debe verificar que NetBeans esté usando Java 1.7.0.

- d. Haga clic en el botón Close para cerrar el cuadro de diálogo About.

Nota: NetBeans 7.0.1 es la primera versión de NetBeans que cuenta con soporte total de la versión final de JDK 7.

Práctica 1-2: Instalación del software

Visión general

En esta práctica, instalará el software necesario para realizar un desarrollo de Java 7 SE.

Supuestos

El instructor le ha pedido que realice estos pasos.

Resumen

Se le ha proporcionado un sistema de computadora que se usará para el desarrollo del software con Java SE 7. Utilizará la información obtenida en la práctica anterior para determinar el software que se va a instalar y, a continuación, lo instalará.

Hay disponible un grupo conjunto de JDK y NetBeans, que permitirá reducir el número de archivos que es necesario descargar. Descargar el JDK y NetBeans por separado ofrece una mayor flexibilidad a la hora de seleccionar el JDK que se va a usar (de 32 bits o de 64 bits), además de reducir la cantidad de datos que descargar si está instalado el JDK o NetBeans.

Para obtener más información sobre los sistemas operativos soportados de Oracle JDK 7, vaya a <http://www.oracle.com/technetwork/java/javase/config-417990.html>. Puede que a través de otros canales haya disponible soporte de Java 7 para más sistemas operativos.

Tareas

1. Obtenga el software necesario.
 - a. Si necesita tanto el JDK como NetBeans, el método más sencillo consiste en descargar el grupo conjunto “JDK 7 with NetBeans 7.0.1”.
 - 1) Mediante un explorador web, vaya a <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.
 - 2) Busque la tabla de grupos conjuntos de Java SE Development Kit, Java SE Development Kit (JDK) Cobundles.
 - 3) Haga clic en el botón Download de “JDK 7 with NetBeans 7.0.1”.
 - 4) Debe aceptar el acuerdo de licencia del grupo conjunto JDK 7 y NetBeans 7.0.1 para poder descargar el software.
 - 5) Descargue el archivo correspondiente a su sistema operativo. En el momento de redactar esta documentación, había disponibles descargas para Linux, Solaris y Windows.
 - b. Si solo necesita el JDK, descargue Java SE 7 JDK.
 - 1) Mediante un explorador web, vaya a <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.
 - 2) Busque la tabla Java Platform, Standard Edition.
 - 3) Haga clic en el botón Download de Java SE 7 JDK.
Nota: asegúrese de descargar el *JDK* y no el *JRE*.
 - 4) Debe aceptar el acuerdo de licencia del grupo conjunto JDK 7 y NetBeans 7.0.1 para poder descargar el software.
 - 5) Descargue el archivo adecuado a su sistema operativo. En el momento de redactar esta documentación, había disponibles descargas para Linux, Solaris y Windows.

- c. Si solo necesita NetBeans IDE, descargue “NetBeans IDE 7.0.1 for Java SE”.
- 1) Con un explorador web, vaya a <http://download.netbeans.org/netbeans/7.0.1/final/>.
 - 2) Haga clic en el botón Download de Java SE.
 - 3) Si la descarga no se inicia automáticamente, haga clic en el enlace “download it here”.

2. Instale el software necesario.

- Instale el software descargado en el paso anterior.
- El JDK incluye demostraciones opcionales y una base de datos denominada JavaDB. Instale todos los componentes opcionales del JDK si está instalando el JDK.
- Si ha descargado el JDK y NetBeans por separado (no el grupo conjunto), realice la instalación del JDK antes de instalar NetBeans.
- Si tiene varias versiones de JDK instaladas, asegúrese de seleccionar Java SE 7 JDK si se le solicita durante la instalación de NetBeans.
- Si se ha instalado NetBeans 7.0.1 antes de JDK 7 o se ha seleccionado una versión anterior de JDK durante la instalación de NetBeans, realice la práctica 1-3 para volver a configurar NetBeans para usar JDK 7.

3. Verifique la instalación del software.

- Repita la práctica 1-1 para verificar la instalación del software.

Práctica 1-3: Configuración de NetBeans 7.0.1 para utilizar JDK 7

Visión general

En esta práctica, configurará NetBeans 7.0.1 para usar una instancia de JDK 7 instalada localmente.

Supuestos

El instructor le ha pedido que realice estos pasos.

Resumen

Estos pasos solo son necesarios si se han instalado tanto NetBeans 7.0.1 como JDK 7, pero NetBeans se está ejecutando con el JDK incorrecto. Puede verificar el JDK que está usando NetBeans con los pasos descritos en la práctica 1-1, paso 4. Si no corrige la configuración de NetBeans para que use JDK 7, no podrá crear proyectos Java que usen las funciones del lenguaje Java 7.

Tareas

1. Configure NetBeans para que reconozca la plataforma Java 7.
 - a. En NetBeans IDE, seleccione Tools y, a continuación, Java Platforms en el menú Main.
 - b. Haga clic en el botón Add Platform y especifique el directorio que contiene la instalación de JDK 7.
 - c. En el paso Platform Name, verifique que las ubicaciones por defecto del archivo zip Platform Sources y de la documentación API son válidas.
 - d. Haga clic en el botón Finish para cerrar el cuadro de diálogo Add Java Platform.
 - e. Asegúrese de que ha seleccionado JDK 1.7 en la lista Platforms y haga clic en Close.
2. Configure NetBeans para que se inicie con Java SE 7 JDK.
 - a. Abra el directorio que contiene los archivos de configuración de NetBeans, normalmente: C:\Program Files\NetBeans 7.0.1\etc\.
 - b. Utilice un editor de texto para editar el archivo `netbeans.conf`.
 - c. Modifique la propiedad `netbeans_jdkhome` para que incluya un valor de la ubicación de instalación de JDK 7, por ejemplo: `netbeans_jdkhome="C:\Program Files\Java\jdk1.7.0"`.
3. Reinicie NetBeans y verifique el JDK que está usando NetBeans con los pasos señalados en la práctica 1-1, paso 4.

Carlos Jaramillo (cajaramillo@gmail.com) has a non-transferable
license to use this Student Guide.

Prácticas de la lección 2: Sintaxis Java y revisión de clases

Capítulo 2

Prácticas de la lección 2: Visión general

Visión general de las prácticas

En estas prácticas, utilizará NetBeans IDE y creará un proyecto, creará paquetes y una clase principal Java y, a continuación, agregará clases y subclases. También ejecutará el proyecto desde el IDE y aprenderá a transferir argumentos de la línea de comandos a la clase principal.

Nota: hay dos niveles para la mayoría de las prácticas de este curso. Las prácticas marcadas con “Nivel detallado” proporcionan más instrucciones y, como su nombre indica, a un nivel más detallado. Las prácticas marcadas con “Nivel de resumen” ofrecen menos detalles y para realizarlas probablemente se necesite un repaso adicional de los materiales de la guía del alumno. El resultado de las prácticas de nivel “Detallado” y “Resumen” es el mismo, por lo que también puede usar el resultado de la solución como herramienta para guiarle.

Práctica 2-1: Nivel de resumen: Creación de clases Java

Visión general

En esta práctica, mediante NetBeans IDE, creará una clase `Employee`, creará una clase con un método `main` para probar la clase `Employee`, compilará y ejecutará la aplicación e imprimirá los resultados en la salida de la línea de comandos.

Tareas

1. Inicie NetBeans IDE mediante el icono del escritorio.
2. Cree un nuevo proyecto `EmployeePractice` en el directorio `D:\labs\02-Review\practices` con una clase principal `EmployeeTest` en el paquete `com.example`.
3. Defina el formato Source/Binary en JDK 7.
 - a. Haga clic con el botón derecho en el proyecto y seleccione Properties.
 - b. Seleccione JDK 7 en la lista desplegable de Source/Binary Format.
 - c. Haga clic en OK.
4. Cree otro paquete denominado `com.example.domain`.
5. Agregue una clase denominada `Employee` en el paquete `com.example.domain`.
6. Codifique la clase `Employee`.
 - a. Agregue los siguientes campos de datos a la clase `Employee`; utilice su buen criterio para decidir qué nombres asignar a estos campos de la clase. Consulte los materiales de la lección para obtener ideas sobre los nombres de campos y la sintaxis si no está seguro. Utilice `public` como modificador de acceso.

Uso de campo	Tipo de campo recomendado
Employee id	int
Employee name	String
Employee Social Security Number	String
Employee salary	double

7. Cree un constructor `no-arg` para la clase `Employee`.

NetBeans puede dar formato al código en cualquier momento. Haga clic con el botón derecho en cualquier parte de la clase y seleccione Format o pulse la combinación de teclas `Alt-Mayús-F`.

8. Agregue los métodos de acceso o mutadores a cada uno de los campos.

Tenga en cuenta que NetBeans incluye una función para crear los métodos getter. Haga clic en la clase donde desee incluir los métodos, haga clic con el botón derecho y seleccione Insert Code (o pulse las teclas Alt-Insert). Seleccione Getters en el menú Generate y haga clic en los cuadros situados junto a los campos para los que desea generar métodos getter y setter.

9. Escriba código en la clase `EmployeeTest` para probar la clase `Employee`.

- a. Cree una instancia de `Employee`.

- b. Utilice los métodos setter para asignar los siguientes valores a la instancia:

Campo	Valor
Employee id	101
Employee name	Jane Smith
Employee Social Security Number	012-34-4567
Employee salary	120_345.27

- c. En el cuerpo del método `main`, utilice el método `System.out.println` para escribir el valor de los campos de empleado en la salida de la consola.
- d. Resuelva cualquier sentencia de importación que falte.
- e. Guarde la clase `EmployeeTest`.

10. Ejecute el proyecto `EmployeePractice`.

11. (Opcional) Agregue algunas instancias de empleado adicionales a la clase de prueba.

Práctica 2-1: Nivel detallado: Creación de clases Java

Visión general

En esta práctica, mediante NetBeans IDE, creará una clase `Employee`, creará una clase con un método `main` para probar la clase `Employee`, compilará y ejecutará la aplicación e imprimirá los resultados en la salida de la línea de comandos.

Tareas

1. Inicie NetBeans IDE mediante el icono del escritorio.
2. Cree un nuevo proyecto denominado `EmployeePractice` en NetBeans con una clase `EmployeeTest` y un método `main`.
 - a. Haga clic en `File > New Project`.
 - b. Seleccione `Java` en `Categories` y `Java Application` en `Projects`.
 - c. Haga clic en `Next`.
 - d. En la pantalla `New Application`, introduzca los siguientes valores:

Ventana/descripción de la página	Opciones o valores
Project Name:	<code>EmployeePractice</code>
Project Location	<code>D:\labs\02-Review\practices</code>
Use Dedicated Folder for Storing Libraries	Desactivado
Create Main Class	Seleccionado Cambie el nombre por <code>com.example.EmployeeTest</code> . <code>com.example</code> es el nombre del paquete.
Set as Main Project	Seleccionado

- e. Haga clic en `Finish`.

Observará que NetBeans le ha evitado tener que escribir muchos datos al crear una clase denominada `EmployeeTest`, incluido el nombre de paquete `com.example` y al escribir el esqueleto del método `main`.
3. Defina el formato `Source/Binary` en `JDK 7`.
 - a. Haga clic con el botón derecho en el proyecto y seleccione `Properties`.
 - b. Seleccione `JDK 7` en la lista desplegable de `Source/Binary Format`.
 - c. Haga clic en `OK`.
 4. Cree otro paquete denominado `com.example.domain`.
 - a. Haga clic con el botón derecho en el paquete actual `com.example`, en `Source Packages`.
 - b. Seleccione `New > Java Package`. En el cuadro de diálogo `New Java Package` se resalta el nuevo nombre del paquete.
 - c. Introduzca `com.example.domain` en el campo `Package Name` y haga clic en `Finish`.

Observará que el icono que aparece junto al nombre del paquete es de color gris en `Project`; esto se debe a que el paquete aún no incluye clases.

5. Cree una nueva clase denominada `Employee` en el paquete `com.example.domain`.
 - a. Haga clic con el botón derecho en el paquete `com.example.domain` y seleccione **New > Java Class**.
 - b. En el campo **Class Name**, introduzca `Employee`.
 - c. Haga clic en **Finish** para crear la clase.

Observe que NetBeans ha generado una clase con el nombre `Employee` en el paquete `com.example.domain`.

6. Codifique la clase `Employee`.
 - a. Agregue los siguientes campos de datos a la clase `Employee`.

Uso de campo	Acceso	Tipo de campo recomendado	Nombre de campo
Employee id	public	int	empId
Employee name	public	String	name
Employee Social Security Number	public	String	ssn
Employee salary	public	double	salary

- b. Agregue un constructor a la clase `Employee`:

```
public Employee() { }
```

NetBeans puede dar formato al código en cualquier momento. Haga clic con el botón derecho en cualquier parte de la clase y seleccione **Format** o pulse la combinación de teclas **Alt-Mayús-F**.

- c. Cree métodos de acceso o mutadores (getter/setter) para cada uno de los campos.

Tenga en cuenta que NetBeans incluye una función para crear los métodos getter. Haga clic en la clase donde desee incluir los métodos, haga clic con el botón derecho y seleccione **Insert Code** (o pulse las teclas **Alt-Insert**). Seleccione **"Getter and Setter"** en el menú **Generate** y haga clic en los cuadros situados junto a los campos para los que desea generar métodos getter y setter. También puede hacer clic en el nombre de clase (`Employee`) para seleccionar todos los campos. Haga clic en **Generate** para insertar el código.

El comprobador de sintaxis automatizada incorporado de NetBeans le debe haber proporcionado las indicaciones en caso de errores de sintaxis o errores de código. Guarde la clase.

7. Escriba código en la clase principal `EmployeeTest` para probar la clase `Employee`.

- a. Agregue una sentencia de importación a la clase del objeto `Employee`:

```
import com.example.domain.Employee;
```

- b. En el método **main** de `EmployeeTest`, cree una instancia de la clase `Employee`, como esta:

```
Employee emp = new Employee();
```

- c. Con la instancia del objeto de empleado, agregue datos al objeto mediante los métodos setter. Por ejemplo:

```
emp.setEmpId(101);
emp.setName("Jane Smith");
emp.setSsn ("012-34-5678");
emp.setSalary(120_345.27);
```

Tenga en cuenta que después de escribir `emp.`, NetBeans le ofrece los nombres de campo sugeridos (en color verde) y los nombres de método (en color negro) a los que se puede acceder mediante la referencia `emp` que ha introducido. Puede usar esta función para reducir la cantidad de datos que debe introducir. Después de escribir el punto que aparece después de `emp`, utilice las teclas de flecha o el mouse para seleccionar el método adecuado en la lista. Para reducir la lista, siga escribiendo algunas de las primeras letras del nombre del método. Por ejemplo, escribir `set` limitará la lista a los nombres de método que empiecen por `set`. Haga clic dos veces en el método para seleccionarlo.

- d. En el cuerpo del método `main`, utilice el método `System.out.println` para escribir mensajes en la salida de la consola.

```
System.out.println ("Employee id:      " + emp.getEmpId());
System.out.println ("Employee name:    " + emp.getName());
System.out.println ("Employee Soc Sec #: " + emp.getSsn());
System.out.println ("Employee salary:   " + emp.getSalary());
```

La clase `System` está en el paquete `java.lang`, que es el motivo por el no tiene que importarla (por defecto, siempre obtendrá `java.lang`). Dispone de más información sobre el funcionamiento de esta notación de puntos múltiple, pero, por ahora, basta con que sepa que este método adopta un argumento string y escribe esa cadena en la salida de la consola.

- e. Guarde la clase `EmployeeTest`.

8. Examine las propiedades del proyecto.

- Haga clic con el botón derecho en el proyecto y seleccione `Properties`.
- Amplíe `Build`, si es necesario, y seleccione `Compiling`. La opción de la parte superior, "Compile on Save", está seleccionada por defecto. Esto significa que tan pronto como guarde el elemento `Employee` y las clases `EmployeeTest`, se compilarán.
- Seleccione `Run`. Verá que el valor de `Main Class` es `com.example.EmployeeTest`. Se trata de la clase que ejecutará el intérprete Java. El siguiente campo es `Arguments`, que se usa para transferir argumentos al método `main`. Utilizará los argumentos en una lección posterior.
- Haga clic en `Cancel` para cerrar `Project Properties`.

9. Ejecute el proyecto `EmployeePractice`.

- Para ejecutar el proyecto `EmployeePractice`, haga clic con el botón derecho en el proyecto y seleccione `Run`, o bien haga clic en el icono `Run Main Project` (triángulo de color verde) o pulse `F6`.

- b. Si las clases no tienen errores, debe aparecer la siguiente salida en la ventana Output:

```
run:
Employee id:          101
Employee name:        Jane Smith
Employee Soc Sec #:   012-34-5678
Employee salary:      120345.27
BUILD SUCCESSFUL (total time: 1 second)
```

10. (Opcional) Agregue algunas instancias de empleado adicionales a la clase de prueba.

Prácticas de la lección 3: Encapsulación y creación de subclases

Capítulo 3

Prácticas de la lección 3: Visión general

Visión general de las prácticas

En estas prácticas, ampliará la clase Employee existente para crear nuevas clases para los elementos Engineer, Admin, Manager y Director.

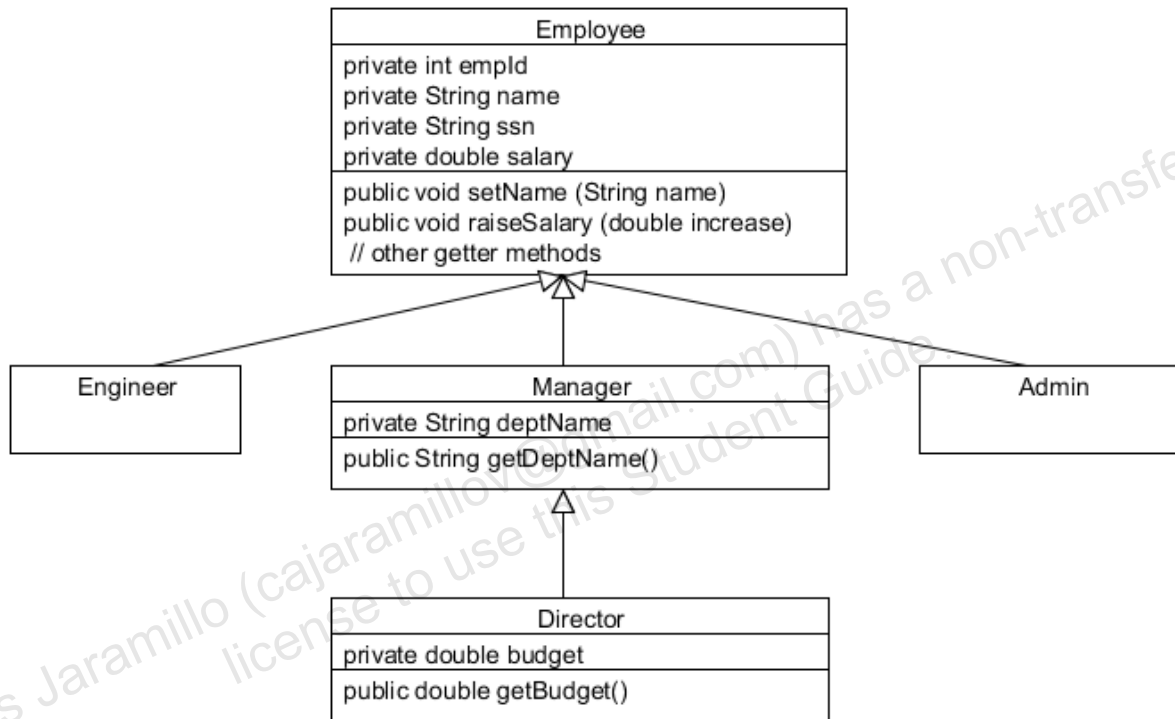
Práctica 3-1: Nivel de resumen: Creación de subclases

Visión general

En esta práctica, creará subclases de `Employee`, incluidas `Manager`, `Engineer` y `Administrative assistant (Admin)`. Creará una subclase de `Manager` denominada `Director` y una clase de prueba con un método `main` para probar las nuevas clases.

Supuestos

Utilice este diagrama de clases Java como guía durante esta práctica.



Tareas

1. Abra el proyecto `EmployeePractice` en el directorio `practices`.
2. Aplique la encapsulación a la clase `Employee`.
 - a. Convierta los campos de la clase `Employee` en privados.
 - b. Sustituya al constructor no-arg de `Employee` por un constructor que tome `empId`, `name`, `ssn` y `salary`.
 - c. Elimine todos los métodos setter, excepto `setName`.
 - d. Agregue un método denominado `raiseSalary` con un parámetro del tipo `double` denominado `increase` para aumentar el salario.
 - e. Guarde `Employee.java`.

3. Cree una subclase de `Employee` denominada `Manager` en el mismo paquete.
 - a. Agregue un campo `String` privado para almacenar el nombre del departamento en un campo denominado `deptName`.
 - b. Cree un constructor que incluya todos los parámetros necesarios para `Employee` y `deptName`.
 - c. Agregue un método `getter` para `deptName`.
4. Cree subclases de `Employee`: `Engineer` y `Admin` en el paquete `com.example.domain`. Estas subclases no necesitan campos o métodos en este momento.
5. Cree una subclase de `Manager` denominada `Director` en el paquete `com.example.domain`.
 - a. Agregue un campo privado para almacenar un valor `double` `budget`.
 - b. Cree un constructor para `Director` que incluya los parámetros necesarios para `Manager` y el parámetro `budget`.
 - c. Cree un método `getter` para este campo.
6. Guarde todas las clases.
7. Pruebe las subclases modificando la clase `EmployeeTest`. Haga que el código realice las siguientes acciones:
 - a. Eliminar el código que crea una instancia de la empleada "Jane Smith".
 - b. Crear una instancia de un elemento `Engineer` con la siguiente información:

Campo	Opciones o valores
ID	101
Name	Jane Smith
SSN	012-34-5678
Salary	120_345.27

Probablemente verá un error junto a la línea que ha agregado para crear un elemento `Engineer`. Esto se debe a que NetBeans no puede resolver el elemento `Engineer` con las sentencias de importación existentes en la clase. La forma más rápida de corregir las sentencias de importación es permitir que NetBeans las rellene. Haga clic con el botón derecho en la clase y seleccione `Fix Imports` o pulse la combinación de teclas `Ctrl + Mayús + I`. NetBeans agregará automáticamente la sentencia `import` para el elemento `Engineer` en el lugar adecuado de la clase y el error desaparecerá.

- c. Cree una instancia de un elemento `Manager` con la siguiente información:

Campo	Opciones o valores
ID	207
Name	Barbara Johnson
SSN	054-12-2367
Salary	109_501.36
Department	US Marketing

- d. Utilice el método `setDeptName` en la instancia de `manager` para definir el campo del departamento en “US Marketing”.
- e. Cree una instancia de un elemento `Admin` con la siguiente información:

Campo	Opciones o valores
ID	304
Name	Bill Monroe
SSN	108-23-6509
Salary	75_002.34

- f. Cree una instancia de un elemento `Director`:

Campo	Opciones o valores
ID	12
Name	Susan Wheeler
SSN	099-45-2340
Salary	120_567.36
Department	Global Marketing
Budget	1_000_000.00

- g. Utilice el método `setDeptName` para definir el campo de nombre de departamento en “Global Marketing”.
- h. Utilice el método `setBudget` de `Director` para definir el presupuesto en 1 000 000.
- i. Guarde `EmployeeTest` y corrija los errores de sintaxis.
8. Agregue un método `printEmployee` a `EmployeeTest` para imprimir un objeto `Employee` con formato (sus campos de datos). El método `printEmployee` debe tomar un objeto `Employee` como parámetro.
9. (Opcional) Utilice los métodos `raiseSalary` y `setName` en algunos de sus objetos para asegurarse de que esos métodos funcionan.
10. Guarde la clase `EmployeeTest` y pruebe el trabajo.
11. (Opcional) Mejore el aspecto de la salida impresa del salario con la clase `NumberFormat`.
 - a. Utilice el siguiente código para obtener una instancia de una clase `java.text.NumberFormat` estática que pueda usar para aplicar formato al salario para que tenga un aspecto de divisa de dólares estadounidenses estándar:

```
NumberFormat.getCurrencyInstance().format((double)emp.getSalary());
```

En la lección sobre clases abstractas, verá cómo usar una fábrica abstracta, como `NumberFormat.getCurrencyInstance()`.
12. (Opcional) Agregue lógica de negocio adicional (validación de datos) a la clase `Employee`.
 - a. Procure que el método `raiseSalary` no tenga un valor negativo.
 - b. Procure que el método `setName` no tenga un valor nulo o vacío.

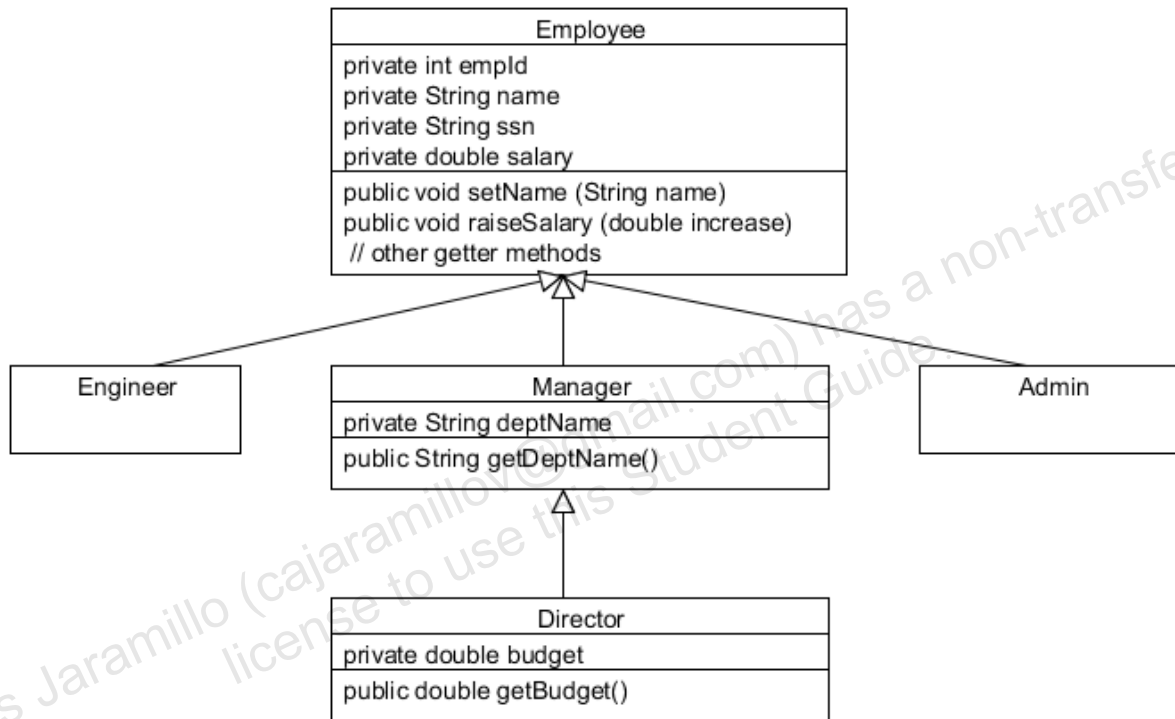
Práctica 3-1: Nivel detallado: Creación de subclases

Visión general

En esta práctica, creará subclases de `Employee`, incluidas `Manager`, `Engineer` y `Administrative assistant (Admin)`. Creará una subclase de `Manager` denominada `Director` y una clase de prueba con un método `main` para probar las nuevas clases.

Supuestos

Utilice este diagrama de clases Java como guía durante esta práctica.



Tareas

1. Abra el proyecto `EmployeePractice` en el directorio `practices`.
 - a. Seleccione `File > Open Project`.
 - b. Vaya a `D:\labs\03-Encapsulation\practices`.
 - c. Seleccione `EmployeePractice`.
 - d. Haga clic en `Open Project`.
2. Aplique la encapsulación a la clase `Employee`.
 - a. Abra la clase `Employee` en el editor.
 - b. Convierta los campos de la clase `Employee` en privados.

- c. Sustituya al constructor no-arg de `Employee` por un constructor que tome `empId`, `name`, `ssn` y `salary`.

```
public Employee(int empId, String name, String ssn, double salary) {
    this.empId = empId;
    this.name = name;
    this.ssn = ssn;
    this.salary = salary;
}
```

- d. Elimine todos los métodos setter, excepto `setName`.
e. Agregue un método denominado `raiseSalary` con un parámetro del tipo `double` denominado `increase` para aumentar el salario.

```
public void raiseSalary(double increase) {
    salary += increase;
}
```

- f. Guarde `Employee.java`.

3. Cree una subclase de `Employee` denominada `Manager`.

- a. Haga clic con el botón derecho en el paquete `com.example.domain` y seleccione `New > Java Class`.
b. Introduzca el nombre de clase `Manager` y haga clic en `Finish`.
c. Modifique la clase por la subclase `Employee`.

Observe que la declaración de clase ahora tiene una marca de error de NetBeans. No olvide que los constructores no se heredan de la clase principal, por lo que necesitará agregar un constructor que defina el valor de los campos heredados de la clase principal. La forma más sencilla de realizar esto es escribir un constructor que llame al constructor principal con la palabra clave `super`.

- d. Agregue un campo `String` privado para almacenar el nombre del departamento en un campo denominado `deptName`.
e. Agregue un constructor que tome los valores `empId`, `name`, `ssn`, `salary` y `deptName` del tipo `String`. El constructor `Manager` debe llamar al constructor `Employee` con la palabra clave `super` y, a continuación, definir el valor de `deptName`.

```
public Manager(int empId, String name, String ssn, double salary, String deptName) {
    super (empId, name, ssn, salary);
    this.deptName = deptName;
}
```

- f. Agregue un método getter para `deptName`.
g. Guarde la clase `Manager`.

4. Cree dos subclases de `Employee`: `Engineer` y `Admin` en el paquete `com.example.domain`. Estas subclases no necesitan campos o métodos en este momento.
 - a. Como los elementos `Engineer` y `Admin` son de tipo `Employee`, agregue un constructor para cada una de las clases que crearán la clase como una instancia de `Employee`.
Indicación: utilice la palabra clave `super` como hizo en la clase `Manager`.
 - b. Guarde las clases.
5. Cree una subclase de `Manager` denominada `Director` en el paquete `com.example.domain`.
 - a. Agregue un campo privado para almacenar un valor `double budget`.
 - b. Agregue los constructores adecuados para `Director`. Utilice la palabra clave `super` para crear una instancia `Manager` y definir el valor de `budget`.
 - c. Cree un método getter para `budget`.
6. Guarde la clase.
7. Pruebe las subclases modificando la clase `EmployeeTest`. Haga que el código realice las siguientes acciones:
 - a. Eliminar el código que crea una instancia de la empleada "Jane Smith".
 - b. Crear una instancia de un elemento `Engineer` con la siguiente información:

Campo	Opciones o valores
ID	101
Name	Jane Smith
SSN	012-34-5678
Salary	120_345.27

Probablemente verá un error junto a la línea que ha agregado para crear un elemento `Engineer`. Esto se debe a que NetBeans no puede resolver el elemento `Engineer` con las sentencias de importación existentes en la clase. La forma más rápida de corregir las sentencias de importación es permitir que NetBeans las rellene. Haga clic con el botón derecho en la clase y seleccione `Fix Imports` o pulse la combinación de teclas `Ctrl + Mayús + I`. NetBeans agregará automáticamente la sentencia `import` para el elemento `Engineer` en el lugar adecuado de la clase y el error desaparecerá.

- c. Cree una instancia de un elemento `Manager` con la siguiente información:

Campo	Opciones o valores
ID	207
Name	Barbara Johnson
SSN	054-12-2367
Salary	109_501.36
Department	US Marketing

d. Cree una instancia de un elemento `Admin` con la siguiente información:

Campo	Opciones o valores
ID	304
Name	Bill Monroe
SSN	108-23-6509
Salary	75_002.34

e. Cree una instancia de un elemento `Director`:

Campo	Opciones o valores
ID	12
Name	Susan Wheeler
SSN	099-45-2340
Salary	120_567.36
Department	Global Marketing
Budget	1_000_000.00

f. Guarde `EmployeeTest` y corrija los errores de sintaxis.

8. Agregue un método `printEmployee` a `EmployeeTest`.

a. Al agregar métodos `System.out.println` después de cada una de las instancias creadas, se va a obtener mucho código redundante. En su lugar, utilizará un método que toma un objeto `Employee` como parámetro:

```
public static void printEmployee (Employee emp) {  
    System.out.println(); // Print a blank line as a separator  
    // Print out the data in this Employee object  
    System.out.println ("Employee id:         " + emp.getEmpId());  
    System.out.println ("Employee name:       " + emp.getName());  
    System.out.println ("Employee Soc Sec #:  " + emp.getSsn());  
    System.out.println ("Employee salary:    " + emp.getSalary());  
}
```

Tenga en cuenta que todas las instancias de objeto que está creando son objetos `Employee`, por lo que, con independencia de la subclase que cree, el método `printEmployee` funcionará. Sin embargo, la clase `Employee` no puede saber nada sobre la especialización de sus subclases. En la siguiente lección verá cómo solucionar esto.

9. Utilice el método `printEmployee` para imprimir información sobre las clases. Por ejemplo:

```
printEmployee (eng);  
printEmployee (man);  
printEmployee (adm);  
printEmployee (dir);
```

10. (Opcional) Utilice los métodos `raiseSalary` y `setName` en algunos de sus objetos para asegurarse de que esos métodos funcionen. Por ejemplo:

```
mgr.setName ("Barbara Johnson-Smythe");  
mgr.raiseSalary(10_000.00);  
printEmployee(mgr);
```

11. Guarde la clase `EmployeeTest` y pruebe el trabajo.
12. (Opcional) Mejore el aspecto de la salida impresa del salario con la clase `NumberFormat`.
- Utilice el siguiente código para obtener una instancia de una clase `java.text.NumberFormat` estática que pueda usar para aplicar formato al salario para que tenga un aspecto de divisa de dólares estadounidenses estándar. Sustituya el valor `emp.getSalary()` por lo siguiente:

```
NumberFormat.getCurrencyInstance().format((double)emp.getSalary())
```

En la lección sobre clases abstractas, verá cómo usar una fábrica abstracta, como `NumberFormat.getCurrencyInstance()`.
13. (Opcional) Agregue lógica de negocio adicional (validación de datos) a la clase `Employee`.
- Procure que el método `raiseSalary` no tenga un valor negativo.
 - Procure que el método `setName` no tenga un valor nulo o vacío.

(Opcional) Práctica 3-2: Adición de una clase Staff a una clase Manager

Visión general

En esta práctica, modificará la clase `Manager` para agregar una matriz de objetos `Employee` (elemento staff) a la clase `manager` y creará métodos para agregar y eliminar empleados del elemento `Manager`. Por último, agregará un método a `Manager` para imprimir los nombres y los ID del personal.

Supuestos

Comience por el proyecto finalizado de la práctica 3-1 (resumen o detallado) o con la solución del directorio `solutions\practice1`.

Tareas

1. Agregue campos a la clase `Manager` para mantener los objetos de empleados.
 - a. Declare un campo privado denominado `staff` que se declare como una matriz de objetos `Employee`.
 - b. Tendrá que realizar un seguimiento del número de empleados de `staff`, por lo que deberá crear un campo de enteros privado `employeeCount` para poder mantener un recuento del número de empleados. Inicialice el recuento de empleados con 0.
 - c. En el constructor, inicialice la matriz `staff` con un máximo de 20 empleados.
2. Agregue un método denominado `findEmployee`. Este método explora la matriz `Employee` del personal actual para ver si existe coincidencia entre algún miembro de `staff` y el elemento `Employee` transferido.
 - a. Devuelve `-1` si no hay coincidencias y el número de índice del empleado si hay una coincidencia.
3. Agregue un método denominado `addEmployee`. Este método agrega el elemento `Employee` transferido como parámetro al final de la matriz.
 - a. Este método debe devolver un valor `boolean` y tomar un objeto `Employee` como parámetro. El método debe devolver `true` si el empleado se ha agregado correctamente y `false` si el empleado ya existe como miembro del personal.
 - b. Llame al método `findEmployee` para determinar si el empleado ya es miembro del personal. Devuelve `false` si existe alguna coincidencia.
 - c. Agregue el objeto de empleado a la matriz de `staff`. (Indicación: utilice `employeeCount` como índice del elemento de matriz al que asignar el parámetro `employee`).
 - d. Aumente el valor de `employeeCount` y devuelva `true`.

4. Agregue un método denominado `removeEmployee`. Este método es un poco más complicado. Al eliminar un elemento de la matriz, debe cambiar los otros elementos de la matriz para que no haya elementos vacíos. La forma más sencilla de realizar esta acción es crear una nueva matriz y asignarle una copia de cada uno de los elementos `staff`, excepto para la coincidencia. De esta forma, se elimina eficazmente la coincidencia de la matriz.
 - a. Declare una variable local `boolean` inicializada como `false` para que se devuelva el estado del método.
 - b. Declare una matriz temporal de los objetos `Employee` para copiarle la matriz de personal revisada.
 - c. Declare un contador de enteros del número de empleados copiado en la matriz temporal.
 - d. Utilice un bucle `FOR` para desplazarse por la matriz `staff` e intentar hacer coincidir el ID de empleado de cada uno de los elementos de la matriz `staff` con el ID de empleado transferido en el método como parámetro.
 - e. Si los ID de empleado no coinciden, copie la referencia de empleado de la matriz `staff` en la matriz temporal del paso b y aumente el recuento de los empleados en la matriz temporal.
 - f. Si hay una coincidencia, “omite” este empleado. Para ello, vaya al siguiente elemento de la matriz `staff` y defina la variable local `boolean` del paso a en `true`.
 - g. Si hay una coincidencia (el valor `boolean` local es `true`), sustituya la matriz `staff` actual por la matriz temporal y el recuento de los empleados con el contador temporal del paso c.
 - h. Devuelva la variable local `boolean`.
5. Agregue un método denominado `printStaffDetails`. Con este método se imprime el nombre del superior y, a continuación, cada uno de los elementos de `staff` a su vez.

Prácticas de la lección 4: Diseño de clases Java

Capítulo 4

Prácticas de la lección 4

Visión general de las prácticas

En estas prácticas, sustituirá métodos, incluido el método `toString` de la clase `Object`.

También creará un método en una clase que use el operador `instanceof` para determinar el objeto que se ha transferido al método.

Práctica 4-1: Nivel de resumen: Sustitución de métodos y aplicación de polimorfismo

Visión general

En esta práctica, sustituirá el método `toString` de la clase `Object` en la clase `Employee` y en la clase `Manager`. Creará una clase `EmployeeStockPlan` con un método `grantStock` que use el operador `instanceof` para determinar la cantidad de acciones que otorgar en función del tipo de empleado de que se trate.

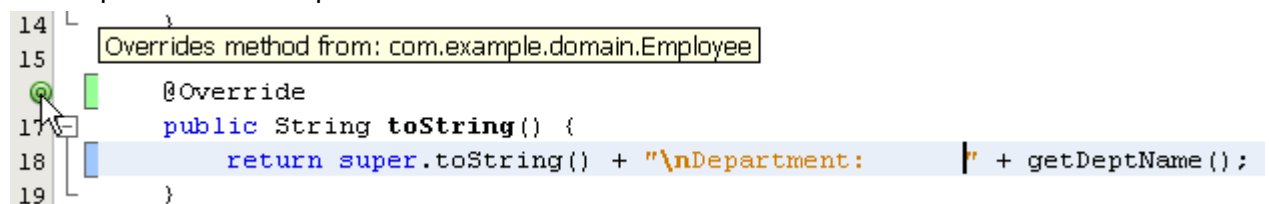
Supuestos

Tareas

1. Abra el proyecto `EmployeePractice` del directorio `practices`.
2. Edite la clase `Employee` para sustituir el método `toString()` de la clase `Object`. El método `toString` de `Object` devuelve un valor `String`.
 - a. Agregue una sentencia `return` que devuelva una cadena que incluya el ID, el nombre, el número de la Seguridad Social y un salario del empleado como cadena con formato, donde cada línea aparecerá separada por un carácter de nueva línea (`"\n"`).
 - b. Para aplicar formato al valor `double salary`, utilice lo siguiente:
`NumberFormat.getCurrencyInstance().format(getSalary())`
 - c. Corrija cualquier sentencia de importación que falte.
 - d. Guarde la clase.

3. Sustituya el método `toString()` en la clase `Manager` para que incluya el valor del campo `deptName`. Separe esta cadena de la cadena `Employee` con un carácter de nueva línea.

Observe el icono de círculo de color verde con la "o" en el centro que aparece junto a la firma del método en la clase `Manager`. Esto indica que NetBeans sabe que este método sustituye al método de la clase principal, `Employee`. Mantenga el cursor sobre el icono para ver lo que este icono representa:



Haga clic en el icono y NetBeans abrirá la clase `Employee` y mostrará el método `toString()`.

4. (Opcional) Sustituya también el método `toString()` de la clase `Director` para que muestre todos los campos de un valor `Director` y el presupuesto disponible.

5. Cree una nueva clase denominada `EmployeeStockPlan` en el paquete `com.example.business`. Esta clase incluirá un solo método, `grantStock`, que toma un objeto `Employee` como parámetro y devuelve un número entero de opciones de compra de acciones según el tipo de empleado de que se trate:

Tipo de empleado	Número de opciones de compra de acciones
Director	1000
Gestor	100
Resto de empleados	10

- a. Agregue un método `grantStock` que tome una referencia de objeto `Employee` como parámetro y devuelva un entero.
 - b. En el cuerpo del método, determine el tipo de empleado que se transfiere usando la palabra clave `instanceof` y devuelva el número adecuado de opciones de compra de acciones según ese tipo.
 - c. Resuelva cualquier sentencia de importación que falte.
 - d. Guarde la clase `EmployeeStockPlan`.
6. Modifique la clase `EmployeeTest`. Sustituya las cuatro sentencias `print` del método `printEmployee` por una sola sentencia `print` que use el método `toString` que ha creado.
 7. Sobrecargue el método `printEmployee` para que tome un segundo parámetro, `EmployeeStockPlan`, e imprima el número de opciones de compra de acciones que recibirá este empleado.
 - a. En el ejemplo anterior, el método `printEmployee` llama al método `main`, crea una instancia de `EmployeeStockPlan` y transfiere esa instancia a cada uno de los métodos `printEmployee`.
 - b. El nuevo método `printEmployee` debe llamar al primer método `printEmployee` y el número de acciones otorgadas a este empleado:

```
printEmployee (emp);
System.out.println("Stock Options:    " + esp.grantStock(emp));
```

8. Guarde la clase `EmployeeTest` y ejecute la aplicación. Debe aparecer una salida para cada empleado que incluya el número de opciones de compra de acciones, como:

```
Employee id:      101
Employee name:    Jane Smith
Employee Soc Sec #: 012-34-5678
Employee salary:  $120,345.27
Stock Options:    10
```

9. Estaría bien saber a qué tipo pertenece cada uno de los empleados. Agregue lo siguiente al método `printEmployee` original encima de la sentencia `print` que imprime los campos de datos de empleados:

```
System.out.println("Employee type:      " +  
emp.getClass().getSimpleName());
```

De esta forma se imprimirá el nombre simple de la clase (`Manager`, `Engineer`, etc.). La salida del primer registro de empleado se debe parecer a la siguiente:

Employee type:	Engineer
Employee id:	101
Employee name:	Jane Smith
Employee Soc Sec #:	012-34-5678
Employee salary:	\$120,345.27
Stock Options:	10

Práctica 4-1: Nivel detallado: Sustitución de métodos y aplicación de polimorfismo

Visión general

En esta práctica, sustituirá el método `toString` de la clase `Object` en la clase `Employee` y en la clase `Manager`. Creará una clase `EmployeeStockPlan` con un método `grantStock` que use el operador `instanceof` para determinar la cantidad de acciones que otorgar en función del tipo de empleado de que se trate.

Supuestos

Tareas

1. Abra el proyecto `EmployeePractice` del directorio `practices`.
 - a. Seleccione `File > Open Project`.
 - b. Vaya a `D:\labs\04-Class_Design\practices`.
 - c. Seleccione `EmployeePractice` y haga clic en `Open Project`.
2. Edite la clase `Employee` para sustituir el método `toString()` de la clase `Object`. El método `toString` de `Object` devuelve un valor `String`.
 - a. Agregue el método `toString` a la clase `Employee` con la siguiente firma:

```
public String toString() {
```
 - b. Agregue una sentencia `return` que devuelva una cadena que incluya la información del empleado: ID, nombre, número de la Seguridad Social y un salario con formato como el siguiente:

```
return "Employee ID:      " + getEmpId() + "\n" +
      "Employee Name:     " + getName() + "\n" +
      "Employee SSN:      " + getSsn() + "\n" +
      "Employee Salary:   " +
      NumberFormat.getCurrencyInstance().format(getSalary());
```
 - c. Guarde la clase `Employee`.
3. Sustituya el método `toString` de la clase `Manager` para que incluya el valor del campo `deptName`.
 - a. Abra la clase `Manager`.
 - b. Agregue un método `toString` con la misma firma que el método `Employee toString`:

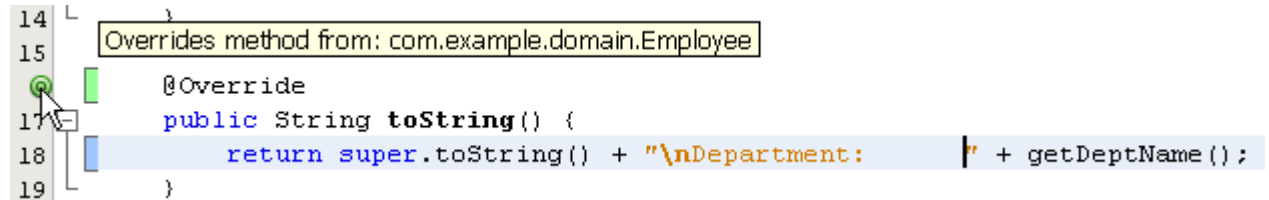
```
public String toString() {
```

El método `toString` de la clase `Manager` sustituye al método `toString` heredado de la clase `Employee`.

- c. Llame al método de la clase principal con la palabra clave `super` y agregue el nombre del departamento:

```
return super.toString() + "\nDepartment: " + getDeptName();
```

Observe el icono de círculo de color verde con la "o" en el centro que aparece junto a la firma del método en la clase `Manager`. Esto indica que NetBeans sabe que este método sustituye al método de la clase principal, `Employee`. Mantenga el cursor sobre el icono para ver lo que este icono representa:



Haga clic en el icono y NetBeans abrirá la clase `Employee` y mostrará el método `toString()`.

- d. Guarde la clase `Manager`.
- (Opcional) Sustituya también el método `toString()` de la clase `Director` para que muestre todos los campos de un valor `Director` y el presupuesto disponible.
 - Cree una nueva clase denominada `EmployeeStockPlan` en el paquete `com.example.business`. Esta clase incluirá un solo método, `grantStock`, que toma un objeto `Employee` como parámetro y devuelve un número entero de opciones de compra de acciones según el tipo de empleado de que se trate:

Tipo de empleado	Número de opciones de compra de acciones
Director	1000
Gestor	100
Resto de empleados	10

- Para crear el nuevo paquete y clase en un paso, haga clic con el botón derecho en `Source Package` y, a continuación, seleccione `New > Java Class`.
- Introduzca `EmployeeStockPlan` como valor para `Class Name` y `com.example.business` como valor para `Package` y haga clic en `Finish`.
- En la nueva clase, agregue campos a la clase para definir los niveles de acciones, como se muestra a continuación:

```
private final int employeeShares = 10;
private final int managerShares = 100;
private final int directorShares = 1000;
```

- Agregue un método `grantStock` que tome una referencia de objeto `Employee` como parámetro y devuelva un entero:

```
public int grantStock(Employee emp) {
```

- e. En el cuerpo del método, determine el tipo de empleado que se transfiere usando la palabra clave `instanceof` y devuelva el número adecuado de opciones de compra de acciones según ese tipo. El código se podría parecer al siguiente:

```
// Stock is granted based on the employee type
if (emp instanceof Director) {
    return directorShares;
} else {
    if (emp instanceof Manager) {
        return managerShares;
    } else {
        return employeeShares;
    }
}
```

- f. Resuelva cualquier sentencia de importación que falte.
- g. Guarde la clase `EmployeeStockPlan`.
6. Modifique la clase `EmployeeTest`. Sustituya las cuatro sentencias `print` del método `printEmployee` por una sola sentencia `print` que use el método `toString` que ha creado.

- a. Sustituya estas líneas:

```
System.out.println("Employee id: " + emp.getEmpId());
System.out.println("Employee name: " + emp.getName());
System.out.println("Employee Soc Sec #: " + emp.getSsn());
System.out.println("Employee salary: " +
    NumberFormat.getCurrencyInstance().format((double)
    emp.getSalary()));
```

- b. Con una línea que use el método `toString()`:
- ```
System.out.println(emp);
```
7. Sobrecargue el método `printEmployee` para que tome un segundo parámetro, `EmployeeStockPlan`, e imprima el número de opciones de compra de acciones que recibirá este empleado.

- a. Cree otro método `printEmployee` que tome una instancia de la clase `EmployeeStockPlan`:
- ```
public static void printEmployee(Employee emp, EmployeeStockPlan
    esp) {
```

- b. Este método llama en primer lugar al método `printEmployee` original:
- ```
printEmployee (emp);
```

- c. Agregue una instancia `print` para imprimir el número de opciones de compra de acciones a las que tenga derecho el empleado:

```
System.out.println("Stock Options: " +
 esp.grantStock(emp));
```

- d. En el ejemplo anterior, el método `printEmployee` llama al método `main`, crea una instancia de `EmployeeStockPlan` y transfiere esa instancia a cada uno de los métodos `printEmployee`:

```
EmployeeStockPlan esp = new EmployeeStockPlan();
printEmployee(eng, esp);
... modify the remaining printEmployee invocations
```

- e. Resuelva cualquier sentencia de importación que falte.
8. Guarde la clase `EmployeeTest` y ejecute la aplicación. Debe aparecer una salida para cada empleado que incluya el número de opciones de compra de acciones, como:

```
Employee id: 101
Employee name: Jane Smith
Employee Soc Sec #: 012-34-5678
Employee salary: $120,345.27
Stock Options: 10
```

9. Estaría bien saber a qué tipo pertenece cada uno de los empleados. Agregue lo siguiente al método `printEmployee` original encima de la sentencia `print` que imprime los campos de datos de empleados:

```
System.out.println("Employee type: " +
 emp.getClass().getSimpleName());
```

De esta forma se imprimirá el nombre simple de la clase (`Manager`, `Engineer`, etc.). La salida del primer registro de empleado se debe parecer a la siguiente:

```
Employee type: Engineer
Employee id: 101
Employee name: Jane Smith
Employee Soc Sec #: 012-34-5678
Employee salary: $120,345.27
Stock Options: 10
```

Carlos Jaramillo (cajaramillo@gmail.com) has a non-transferable  
license to use this Student Guide.

## **Prácticas de la lección 5: Diseño de clases avanzadas**

### **Capítulo 5**

## Prácticas de la lección 5: Visión general

---

### Visión general de las prácticas

En estas prácticas, utilizará las palabras clave Java abstract, final y static. También aprenderá a reconocer las clases anidadas.

## Práctica 5-1: Nivel de resumen: Aplicación de la palabra clave `abstract`

---

### Visión general

En esta práctica, utilizará una aplicación existente y refactorizará el código para utilizar la palabra clave `abstract`.

### Supuestos

Ha revisado la sección sobre la clase `abstract` de esta lección.

### Resumen

Se le ha asignado un proyecto que implanta la lógica para un banco. El software de banca solo soporta la creación de cuentas de ahorros. Estas cuentas permiten la retirada solo después de una fecha de vencimiento. Las cuentas de ahorros también se conocen como depósito a plazo, certificado de depósito (CD) o cuentas de depósito a plazo fijo. Mejorará el software para que soporte cuentas corrientes.

Las cuentas corrientes y las cuentas de ahorro tienen algunas similitudes y algunas diferencias. El diseño de clases que realice debería reflejar este hecho. En el futuro puede que se agreguen más tipos de cuentas.

### Tareas

1. Abra el proyecto `AbstractBanking` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\05-Advanced_Class_Design\practices`.
  - c. Seleccione `AbstractBanking` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas.
4. Revise la clase `TimeDepositAccount`.
  - a. Abra el archivo `TimeDepositAccount.java` (en el paquete `com.example`).
  - b. Identifique los campos y las implantaciones de método de `TimeDepositAccount` relacionadas con el tiempo o que, de alguna otra forma, sean específicos de `TimeDepositAccount`. Agregue un comentario de código si lo desea.
  - c. Identifique los campos y las implantaciones de métodos de `TimeDepositAccount` que podría usar cualquier tipo de cuenta. Agregue un comentario de código si lo desea.
5. Cree una nueva clase Java denominada `Account` en el paquete `com.example`.
6. Codifique la clase `Account`.
  - a. Esta clase se debe declarar como `abstract`.
  - b. Mueva cualquier campo e implantación de método de `TimeDepositAccount` que se podría usar con cualquier tipo de cuenta a la clase `Account`.

**Nota:** los campos y los métodos se deben eliminar de `TimeDepositAccount`.

- c. Agregue métodos `abstract` a la clase `Account` para cualquier método de `TimeDepositAccount` que esté relacionado con el tiempo, pero que tenga una firma de método que tendría sentido en cualquier tipo de cuenta.
- Indicación:** ¿Todas las cuentas tendrían una descripción?
- d. Agregue un constructor de clase `Account` que tenga un parámetro `double balance`.
- e. La clase `Account` debe tener un campo `balance` con nivel de acceso protegido que inicialice el constructor `Account`.
7. Modifique la clase `TimeDepositAccount`.
- a. `TimeDepositAccount` debe ser una subclase de `Account`.
- b. Modifique el constructor `TimeDepositAccount` para que llame al constructor de clase principal con el valor `balance`.
- c. Asegúrese de que sustituye los métodos de `abstract` `withdraw` y `getDescription` heredados de la clase `Account`.
- Nota:** conviene agregar `@Override` a cualquier método que debe sustituir un método de clase principal.
8. Modifique las clases `Customer` y `CustomerReport` para que utilicen las referencias `Account`.
- a. Abra el archivo `Customer.java` (en el paquete `com.example`).
- b. Cambie todas las referencias de `TimeDepositAccount` por referencias del tipo `Account`.
- c. Abra el archivo `CustomerReport.java` (en el paquete `com.example`).
- d. Cambie todas las referencias de `TimeDepositAccount` por referencias del tipo `Account`.
9. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas.
10. Cree una nueva clase Java denominada `CheckingAccount` en el paquete `com.example`.
- a. `CheckingAccount` debe ser una subclase de `Account`.
- b. Agregue un campo `overDraftLimit` a la clase `CheckingAccount`.

```
private final double overDraftLimit;
```

- c. Agregue un constructor `CheckingAccount` que tenga dos parámetros.
- `double balance`: transfiera este valor al constructor de clase principal.
  - `double overDraftLimit`: almacene este valor en el campo `overDraftLimit`.
- d. Agregue un constructor `CheckingAccount` que tenga un parámetro. Este constructor debe definir el campo `overDraftLimit` en cero.
- `double balance`: transfiera este valor al constructor de clase principal.
- e. Sustituya el método `getDescription` de `abstract` heredado de la clase `Account`.

```
@Override
public String getDescription() {
 return "Checking Account";
}
```

**Nota:** conviene agregar `@Override` a cualquier método que debe sustituir un método de clase principal.



- f. Sustituya el método `withdraw` de abstract heredado de la clase `Account`.
- El método `withdraw` debe permitir que el saldo de una cuenta se quede en negativo hasta la cantidad especificada en el campo `overDraftLimit`.
  - El método `withdraw` debe devolver `false` si no se puede realizar la retirada y `true` en caso de que sí se pueda realizar.

11. Modifique la clase `AbstractBankingMain` para crear cuentas corrientes para los clientes.

```
// Create several customers and their accounts
bank.addCustomer("Jane", "Simms");
customer = bank.getCustomer(0);
customer.addAccount(new TimeDepositAccount(500.00,
cal.getTime()));
customer.addAccount(new CheckingAccount(200.00, 400.00));

bank.addCustomer("Owen", "Bryant");
customer = bank.getCustomer(1);
customer.addAccount(new CheckingAccount(200.00));

bank.addCustomer("Tim", "Soley");
customer = bank.getCustomer(2);
customer.addAccount(new TimeDepositAccount(1500.00,
cal.getTime()));
customer.addAccount(new CheckingAccount(200.00));

bank.addCustomer("Maria", "Soley");
customer = bank.getCustomer(3);
// Maria and Tim have a shared checking account
customer.addAccount(bank.getCustomer(2).getAccount(1));
customer.addAccount(new TimeDepositAccount(150.00,
cal.getTime()));
```

**Nota:** tanto `Customer` como `CustomerReport` pueden utilizar instancias de `CheckingAccount`, ya que las ha modificado previamente para que utilicen referencias de tipo `Account`.

12. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas. Tenga en cuenta que la fecha que aparece debe ser dentro de 180 días.

```
CUSTOMERS REPORT
=====

Customer: Simms, Jane
 Time Deposit Account Sat Feb 04 11:14:54 CST 2012: current
 balance is 500.0
 Checking Account: current balance is 200.0

Customer: Bryant, Owen
 Checking Account: current balance is 200.0

Customer: Soley, Tim
 Time Deposit Account Sat Feb 04 11:14:54 CST 2012: current
 balance is 1500.0
 Checking Account: current balance is 200.0

Customer: Soley, Maria
 Checking Account: current balance is 200.0
 Time Deposit Account Sat Feb 04 11:14:54 CST 2012: current
 balance is 150.0
```

## Práctica 5-1: Nivel detallado: Aplicación de la palabra clave abstract

### Visión general

En esta práctica, utilizará una aplicación existente y refactorizará el código para utilizar la palabra clave `abstract`.

### Supuestos

Ha revisado la sección sobre la clase `abstract` de esta lección.

### Resumen

Se le ha asignado un proyecto que implanta la lógica para un banco. El software de banca solo soporta la creación de cuentas de ahorros. Estas cuentas permiten la retirada solo después de una fecha de vencimiento. Las cuentas de ahorros también se conocen como depósito a plazo, certificado de depósito (CD) o cuentas de depósito a plazo fijo. Mejorará el software para que soporte cuentas corrientes.

Las cuentas corrientes y las cuentas de ahorro tienen algunas similitudes y algunas diferencias. El diseño de clases que realice debería reflejar este hecho. En el futuro puede que se agreguen más tipos de cuentas.

### Tareas

1. Abra el proyecto `AbstractBanking` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\05-Advanced_Class_Design\practices`.
  - c. Seleccione `AbstractBanking` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas.

```
CUSTOMERS REPORT
=====

Customer: Simms, Jane
 Time Deposit Account Fri Mar 09 12:04:28 CST 2012: current
balance is 500,0

Customer: Bryant, Owen

Customer: Soley, Tim
 Time Deposit Account Fri Mar 09 12:04:28 CST 2012: current
balance is 1500,0

Customer: Soley, Maria
 Time Deposit Account Fri Mar 09 12:04:28 CST 2012: current
balance is 150.0
```

4. Revise la clase `TimeDepositAccount`.

- a. Abra el archivo `TimeDepositAccount.java` (en el paquete `com.example`).
- b. Identifique los campos y las implantaciones de método de `TimeDepositAccount` relacionadas con el tiempo o que, de alguna otra forma, sean específicos de `TimeDepositAccount`. Agregue un comentario al campo `maturityDate` y los métodos `withdraw` y `getDescription`. Por ejemplo:

```
// time deposit account specific code
private final Date maturityDate;
```

- c. Identifique los campos y las implantaciones de métodos de `TimeDepositAccount` que podría usar cualquier tipo de cuenta. Agregue un comentario de código al campo `balance` y los métodos `getBalance`, `deposit` y `toString`. Por ejemplo:

```
// generic account code
private double balance;
```

5. Cree una nueva clase denominada `Account` en el paquete `com.example`.

6. Codifique la clase `Account`.

- a. Esta clase se debe declarar como `abstract`.

```
public abstract class Account
```

- b. Mueva el campo `balance` y los métodos `getBalance`, `deposit` y `toString` de `TimeDepositAccount` a la clase `Account`.

**Nota:** los campos y los métodos se deben eliminar de `TimeDepositAccount`.

- c. Agregue un método `getDescription` de `abstract` a la clase `Account`.

```
public abstract String getDescription();
```

- d. Agregue un método `withdraw` de `abstract` a la clase `Account`.

```
public abstract boolean withdraw(double amount);
```

- e. La clase `Account` debe tener un campo `balance` de nivel de acceso protegido. Si ya ha movido este campo de `TimeDepositAccount`, solo tendrá que cambiar el nivel de acceso.

```
protected double balance;
```

- f. Agregue un constructor de clase `Account` que tenga un parámetro `double balance`.

```
public Account(double balance) {
 this.balance = balance;
}
```

7. Modifique la clase `TimeDepositAccount`.

- a. `TimeDepositAccount` debe ser una subclase de `Account`.

```
public class TimeDepositAccount extends Account
```

- b. Modifique el constructor `TimeDepositAccount` para que llame al constructor de clase principal con el valor `balance`.

```
super(balance);
```

- c. Asegúrese de que sustituye los métodos `withdraw` y `getDescription` de abstract heredados de la clase `Account`, mediante la anotación `@Override`.

```
@Override
public String getDescription() {
 return "Time Deposit Account " + maturityDate;
}
```

**Nota:** conviene agregar `@Override` a cualquier método que debe sustituir un método de clase principal.

8. Modifique las clases `Customer` y `CustomerReport` para que utilicen las referencias `Account`.
  - a. Abra el archivo `Customer.java` (en el paquete `com.example`).
  - b. Cambie todas las referencias de `TimeDepositAccount` por referencias del tipo `Account`.
  - c. Abra el archivo `CustomerReport.java` (en el paquete `com.example`).
  - d. Cambie todas las referencias de `TimeDepositAccount` por referencias del tipo `Account`.
9. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas.
10. Cree una nueva clase Java denominada `CheckingAccount` en el paquete `com.example`.

- a. `CheckingAccount` debe ser una subclase de `Account`.

```
public class CheckingAccount extends Account
```

- b. Agregue un campo `overDraftLimit` a la clase `CheckingAccount`.

```
private final double overDraftLimit;
```

- c. Agregue un constructor `CheckingAccount`.

```
public CheckingAccount(double balance, double overDraftLimit) {
 super(balance);
 this.overDraftLimit = overDraftLimit;
}
```

- d. Agregue un constructor `CheckingAccount` que tenga un parámetro.

```
public CheckingAccount(double balance) {
 this(balance, 0);
}
```

- e. Sustituya el método `getDescription` de abstract heredado de la clase `Account`.

```
@Override
public String getDescription() {
 return "Checking Account";
}
```

**Nota:** conviene agregar `@Override` a cualquier método que debe sustituir un método de clase principal.

- f. Sustituya el método `withdraw` de abstract heredado de la clase `Account`. El método `withdraw` debe permitir que el saldo de una cuenta se quede en negativo hasta la cantidad especificada en el campo `overDraftLimit`.

```
@Override
public boolean withdraw(double amount) {
 if(amount <= balance + overDraftLimit) {
 balance -= amount;
 return true;
 } else {
 return false;
 }
}
```

11. Modifique la clase `AbstractBankingMain` para crear cuentas corrientes para los clientes.

```
// Create several customers and their accounts
bank.addCustomer("Jane", "Simms");
customer = bank.getCustomer(0);
customer.addAccount(new TimeDepositAccount(500.00,
cal.getTime()));
customer.addAccount(new CheckingAccount(200.00, 400.00));

bank.addCustomer("Owen", "Bryant");
customer = bank.getCustomer(1);
customer.addAccount(new CheckingAccount(200.00));

bank.addCustomer("Tim", "Soley");
customer = bank.getCustomer(2);
customer.addAccount(new TimeDepositAccount(1500,00,
cal.getTime()));
customer.addAccount(new CheckingAccount(200.00));

bank.addCustomer("Maria", "Soley");
customer = bank.getCustomer(3);
// Maria and Tim have a shared checking account
customer.addAccount(bank.getCustomer(2).getAccount(1));
customer.addAccount(new TimeDepositAccount(150,00,
cal.getTime()));
```

**Nota:** Tanto `Customer` como `CustomerReport` pueden utilizar instancias de `CheckingAccount`, ya que las ha modificado previamente para que utilicen referencias de tipo `Account`.

12. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas. Tenga en cuenta que la fecha que aparece debe ser dentro de 180 días.

```
CUSTOMERS REPORT
=====

Customer: Simms, Jane
 Time Deposit Account Sat Feb 04 11:14:54 CST 2012: current
 balance is 500,0
 Checking Account: current balance is 200,0

Customer: Bryant, Owen
 Checking Account: current balance is 200,0

Customer: Soley, Tim
 Time Deposit Account Sat Feb 04 11:14:54 CST 2012: current
 balance is 1500,0
 Checking Account: current balance is 200,0

Customer: Soley, Maria
 Checking Account: current balance is 200,0
 Time Deposit Account Sat Feb 04 11:14:54 CST 2012: current
 balance is 150.0
```

## Práctica 5-2: Nivel de resumen: Aplicación del patrón de diseño Singleton

---

### Visión general

En esta práctica, utilizará una aplicación existente y refactorizará el código para implantar el patrón de diseño Singleton.

### Supuestos

Ha revisado las secciones sobre las palabras clave static y final de esta lección.

### Resumen

Se le ha asignado un proyecto que implanta la lógica para un banco. La aplicación permite actualmente crear un número ilimitado de instancias de `Bank`.

```
Bank bank = new Bank();
Bank bank2 = new Bank();
Bank bank3 = new Bank();
```

Con las palabras clave static y final, podrá limitar el número de instancias de `Bank` a una por Java Virtual Machine (JVM).

### Tareas

1. Abra el proyecto `SingletonBanking` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\05-Advanced_Class_Design\practices`.
  - c. Seleccione `SingletonBanking` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas.

```
CUSTOMERS REPORT
=====

Customer: Simms, Jane
 Time Deposit Account Fri Mar 09 12:04:28 CST 2012: current
balance is 500,0

Customer: Bryant, Owen

Customer: Soley, Tim
 Time Deposit Account Fri Mar 09 12:04:28 CST 2012: current
balance is 1500,0

Customer: Soley, Maria
 Time Deposit Account Fri Mar 09 12:04:28 CST 2012: current
balance is 150.0
```



4. Modifique la clase `Bank` para implantar el patrón de diseño Singleton.
  - a. Abra el archivo `Bank.java` (en el paquete `com.example`).
  - b. Cambie el nivel de acceso del constructor a `private`.
  - c. Agregue un nuevo campo denominado `instance`. El campo debe:
    - Ser `private`.
    - Estar marcado como `static`.
    - Estar marcado como `final`.
    - Ser del tipo `Bank`.
    - Haberse inicializado a una nueva instancia de `Bank`.
  - d. Cree un método estático denominado `getInstance` que devuelva el valor almacenado en el campo `instance`.
5. Modifique la clase `SingletonBankingMain` para que use el singleton `Bank`.
  - a. Abra el archivo `SingletonBankingMain.java` (en el paquete `com.example`).
  - b. Sustituya las llamadas al constructor `Bank` por llamadas al método `getInstance` creado anteriormente.
  - c. En el método `main`, cree una segunda referencia `Bank` local denominada `bank2` e inicialícela con el método `getInstance`.
  - d. Utilice la comprobación de igualdad de referencia para determinar si `bank` y `bank2` hacen referencia al mismo objeto.
 

```
if(bank == bank2) {
 System.out.println("bank and bank2 are the same object");
}
```
  - e. Intente inicializar solo el segundo elemento `Bank`, pero ejecutando el informe en el primer elemento `Bank`.
 

```
initializeCustomers(bank2);

// run the customer report
CustomerReport report = new CustomerReport();
report.setBank(bank);
report.generateReport();
```
6. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas.

## Práctica 5-2: Nivel detallado: Aplicación del patrón de diseño Singleton

---

### Visión general

En esta práctica, utilizará una aplicación existente y refactorizará el código para implantar el patrón de diseño Singleton.

### Supuestos

Ha revisado las secciones sobre las palabras clave `static` y final de esta lección.

### Resumen

Se le ha asignado un proyecto que implanta la lógica para un banco. La aplicación permite actualmente crear un número ilimitado de instancias de `Bank`.

```
Bank bank = new Bank();
Bank bank2 = new Bank();
Bank bank3 = new Bank();
```

Con las palabras clave `static` y `final`, podrá limitar el número de instancias de `Bank` a una por Java Virtual Machine (JVM).

### Tareas

1. Abra el proyecto `SingletonBanking` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\05-Advanced_Class_Design\practices`.
  - c. Seleccione `SingletonBanking` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas.
4. Modifique la clase `Bank` para implantar el patrón de diseño Singleton.
  - a. Abra el archivo `Bank.java` (en el paquete `com.example`).
  - b. Cambie el nivel de acceso del constructor a `private`.

```
private Bank() {
 customers = new Customer[10];
 numberOfCustomers = 0;
}
```

c. Agregue un nuevo campo denominado `instance`. El campo debe:

- Ser `private`.
- Estar marcado como `static`.
- Estar marcado como `final`.
- Ser del tipo `Bank`.
- Haberse inicializado a una nueva instancia de `Bank`.

```
private static final Bank instance = new Bank();
```

d. Cree un método estático denominado `getInstance` que devuelva el valor almacenado en el campo `instance`.

```
public static Bank getInstance() {
 return instance;
}
```

5. Modifique la clase `SingletonBankingMain` para que use el singleton `Bank`.

- Abra el archivo `SingletonBankingMain.java` (en el paquete `com.example`).
- Sustituya las llamadas al constructor `Bank` por llamadas al método `getInstance` creado anteriormente.

```
Bank bank = Bank.getInstance();
```

c. En el método `main`, cree una segunda referencia `Bank` local denominada `bank2` e inicialícela con el método `getInstance`.

```
Bank bank2 = Bank.getInstance();
```

d. Utilice la comprobación de igualdad de referencia para determinar si `bank` y `bank2` hacen referencia al mismo objeto.

```
if(bank == bank2) {
 System.out.println("bank and bank2 are the same object");
}
```

e. Inicialice solo el segundo elemento `Bank`, pero ejecute el informe en el primer elemento `Bank`.

```
initializeCustomers(bank2);

// run the customer report
CustomerReport report = new CustomerReport();
report.setBank(bank);
report.generateReport();
```

6. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas.

## (Opcional) Práctica 5-3: Uso de enumeraciones Java

### Visión general

En esta práctica, utilizará una aplicación existente y refactorizará el código para utilizar una enumeración.

### Supuestos

Ha revisado la sección sobre la enumeración de esta lección.

### Resumen

Se le ha asignado un proyecto que implanta la lógica para un banco. La aplicación permite actualmente crear instancias `TimeDepositAccount` con cualquier fecha de vencimiento.

```
//180 day term
Calendar cal = Calendar.getInstance();
cal.add(Calendar.DAY_OF_YEAR, 180);
new TimeDepositAccount(500.00, cal.getTime())
```

Al crear una nueva enumeración Java, modificará la aplicación para que solo permita crear instancias de `TimeDepositAccount` con una fecha de vencimiento de 90 o 180 días en el futuro.

### Tareas

1. Abra el proyecto `EnumBanking` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\05-Advanced_Class_Design\practices`.
  - c. Seleccione `EnumBanking` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas.
4. Cree una nueva enumeración Java denominada `DepositLength` en el paquete `com.example`.
5. Codifique la enumeración `DepositLength`.
  - a. Declare un campo `days` junto con un constructor correspondiente y un método `getter`.

```
private int days;

private DepositLength(int days) {
 this.days = days;
}

public int getDays() {
 return days;
}
```

- b. Cree dos instancias `DepositLength`, `THREE_MONTHS` y `SIX_MONTHS` que llamen al constructor `DepositLength` con valores de 90 y 180 respectivamente.
6. Modifique la clase `TimeDepositAccount` para que solo acepte instancias de `DepositLength` para el parámetro de fecha de vencimiento del constructor.
  - a. Abra el archivo `TimeDepositAccount.java` (en el paquete `com.example`).
  - b. Modifique el constructor existente para que reciba una enumeración como segundo parámetro.

```
public TimeDepositAccount(double balance, DepositLength
duration) {
 super(balance);
 Calendar cal = Calendar.getInstance();
 cal.add(Calendar.DAY_OF_YEAR, duration.getDays());
 this.maturityDate = cal.getTime();
}
```

7. Modifique la clase `EnumBankingMain` para que cree instancias de `TimeDepositAccount` con las dos instancias de `DepositLength` disponibles.
  - a. Abra el archivo `EnumBankingMain.java` (en el paquete `com.example`).
  - b. En el método `initializeCustomers`, elimine el código para crear calendarios.
  - c. En el método `initializeCustomers`, modifique la creación de todas las instancias de `TimeDepositAccount` para que usen la enumeración `DepositLength`.

```
customer.addAccount(new TimeDepositAccount(500.00,
DepositLength.SIX_MONTHS));
```

**Nota:** intente usar ambos valores, `SIX_MONTHS` y `THREE_MONTHS`. También puede usar una importación estática para reducir la longitud de la sentencia.

8. Ejecute el proyecto. Debería aparecer un informe con todos los clientes y sus cuentas. Ahora no se puede compilar una línea de código que crea un elemento `TimeDepositAccount` con una fecha de madurez no válida.

## (Opcional) Práctica 5-4: Reconocimiento de clases anidadas

---

### Visión general

En esta práctica, utilizará una aplicación existente e intentará reconocer la declaración y el uso de distintos tipos de clases anidadas.

### Supuestos

Ha revisado la sección sobre la clase anidada de esta lección.

### Resumen

Se le ha asignado una clase pequeña que solo contiene dos archivos `.java`. Si bien solo hay dos archivos `.java`, puede que se estén creando varias clases Java.

Intente determinar el número de clases que se están creando.

### Tareas

1. Abra el proyecto `NestedClasses` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\05-Advanced_Class_Design\practices`.
  - c. Seleccione `NestedClasses` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debe aparecer la salida en la ventana correspondiente.
4. Cuente el número de clases creadas en el archivo `OuterClass.java`.
  - a. Abra el archivo `OuterClass.java` (en el paquete `com.example`).
  - b. Determine el número total de clases creadas en este archivo.
  - c. Determine el número total de clases de nivel superior creadas en este archivo.
  - d. Determine el número total de clases anidadas creadas en este archivo.
  - e. Determine el número total de clases internas.
  - f. Determine el número total de clases de miembro.
  - g. Determine el número total de clases locales.
  - h. Determine el número total de clases anónimas.
  - i. Determine el número total de clases anidadas estáticas.

**Indicación:** en el separador `Files` de NetBeans, puede ver cuántos archivos `.class` se crean si observa la carpeta `build\classes` de un proyecto.

## (Opcional) Solución 5-4: Reconocimiento de clases anidadas

---

### Visión general

En esta solución, utilizará una aplicación existente y revisará el número y los tipos de clases anidadas creadas en un solo archivo `.java`.

### Supuestos

Ha revisado la sección sobre la clase anidada de esta lección.

### Resumen

Se le ha asignado una clase pequeña que solo contiene dos archivos `.java`. Si bien solo hay dos archivos `.java`, puede que se estén creando varias clases Java.

Revise el número de clases que se están creando.

### Tareas

1. Abra el proyecto `NestedClasses` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\05-Advanced_Class_Design\practices`.
  - c. Seleccione `NestedClasses` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debe aparecer la salida en la ventana correspondiente.
4. Abra el archivo `OuterClass.java` (en el paquete `com.example`).
  - El archivo `OuterClass.java` incluye:
    - 10 clases
      - 1 clase de nivel superior
      - 9 clases anidadas
        - 8 clases internas
          - 3 clases de miembro
          - 2 clases locales
          - 3 clases anónimas
        - 1 clase anidada estática

- Las clases se declaran en las siguientes líneas del archivo `OuterClasss.java`:
  - línea 3: clase de nivel superior
  - línea 10: clase interna local
  - línea 22: clase interna local anónima
  - línea 32: clase interna anónima
  - línea 40: clase interna anónima
  - línea 48: clase interna de miembro
  - línea 62: clase anidada estática
  - línea 72: clase interna de miembro
  - línea 74: clase interna de miembro
  - línea 77: clase interna local

**Indicación:** para que el número de línea aparezca en NetBeans, vaya al menú View y active Show Line Numbers.



## **Prácticas de la lección 6: Herencia con interfaces Java**

### **Capítulo 6**

## Prácticas de la lección 6: Visión general

---

### Visión general de las prácticas

En estas prácticas, utilizará interfaces Java y aplicará patrones de diseño.

## Práctica 6-1: Nivel de resumen: Implantación de una interfaz

### Visión general

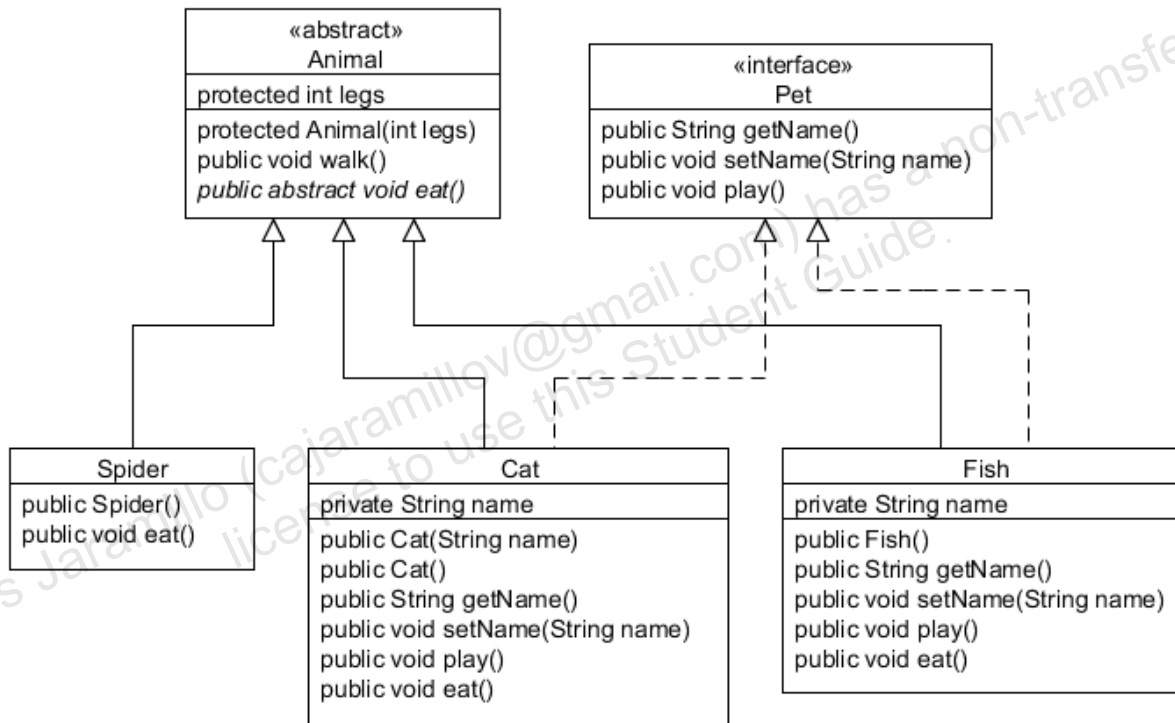
En esta práctica, creará una interfaz y la implantará.

### Supuestos

Ha revisado la sección sobre la interfaz de esta lección.

### Resumen

Se le ha asignado un proyecto que contiene una clase abstract denominada `Animal`. Creará una jerarquía de animales con raíz en la clase `Animal`. Varias de las clases de animales implantan una interfaz denominada `Pet`, que usted creará.



### Tareas

1. Abra el proyecto `Pet` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\06-Interfaces\practices`.
  - c. Seleccione `Pet` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debe aparecer texto en la ventana de salida.

4. Revise las clases `Animal` y `Spider`.
  - a. Abra el archivo `Animal.java` (en el paquete `com.example`).
  - b. Revise la clase `Animal` de abstract, que tendrá que ampliar.
  - c. Abra el archivo `Spider.java` (en el paquete `com.example`).
  - d. La clase `Spider` es un ejemplo de ampliación de la clase `Animal`.
5. Cree una nueva interfaz Java: `Pet` en el paquete `com.example`.
6. Codifique la interfaz de `Pet`. Esta interfaz debe incluir tres firmas de métodos:
  - `public String getName();`
  - `public void setName(String name);`
  - `public void play();`
7. Cree una nueva clase Java: `Fish` en el paquete `com.example`.
8. Codifique la clase `Fish`.
  - a. Esta clase debe:
    - Ampliar la clase `Animal`.
    - Implantar la interfaz de `Pet`.
  - b. Para terminar esta clase, cree:
    - Un campo `String` denominado `name`.
    - Métodos `getter` y `setter` para el campo `name`.
    - Un constructor sin argumentos que transfiera un valor 0 al constructor principal.
    - Un método `play()` que imprima "Just keep swimming."
    - Un método `eat()` que imprima "Fish eat pond scum."
    - Un método `walk()` que sustituya el método `walk` de la clase `Animal`. Primero debe llamar al método `walk` de la superclase para, a continuación, imprimir "Fish, of course, can't walk; they swim."
9. Cree una nueva clase Java: `Cat` en el paquete `com.example`.
10. Codifique la clase `Cat`.
  - a. Esta clase debe:
    - Ampliar la clase `Animal`.
    - Implantar la interfaz de `Pet`.
  - b. Para terminar esta clase, cree:
    - Un campo `String` denominado `name`.
    - Métodos `getter` y `setter` para el campo `name`.
    - Un constructor que reciba un nombre `String` y transfiera un valor 4 al constructor principal.
    - Un constructor sin argumentos que transfiera un valor "Fluffy" al otro constructor de esta clase.
    - Un método `play()` que imprima `name + " likes to play with string."`
    - Un método `eat()` que imprima "Cats like to eat spiders and fish."

11. Modifique la clase `PetMain`.

- a. Abra el archivo `PetMain.java` (en el paquete `com.example`).
- b. Revise el método `main`. Deben aparecer las siguientes líneas de código:

```
Animal a;
//test a spider with a spider reference
Spider s = new Spider();
s.eat();
s.walk();
//test a spider with an animal reference
a = new Spider();
a.eat();
a.walk();
```

- c. Agregue más líneas de código para probar las clases `Fish` y `Cat` creadas.
  - Intente usar todos los constructores.
  - Pruebe con todos los tipos de referencias posibles y determine los métodos que se pueden llamar con cada tipo de referencia. Utilice una referencia `Pet` mientras prueba las clases `Fish` y `Cat`.
- d. Implante y pruebe el método `playWithAnimal(Animal a)`.
  - Determine si el argumento implanta la interfaz de `Pet`. En caso afirmativo, convierta la referencia en un elemento `Pet` y llame al método `play`. En caso negativo, imprima un mensaje "Danger! Wild Animal".
  - Llame al método `playWithAnimal(Animal a)` desde `main`, transfiriendo cada uno de los tipos de animal.

12. Ejecute el proyecto. Debe aparecer texto en la ventana de salida.

## Práctica 6-1: Nivel detallado: Implantación de una interfaz

### Visión general

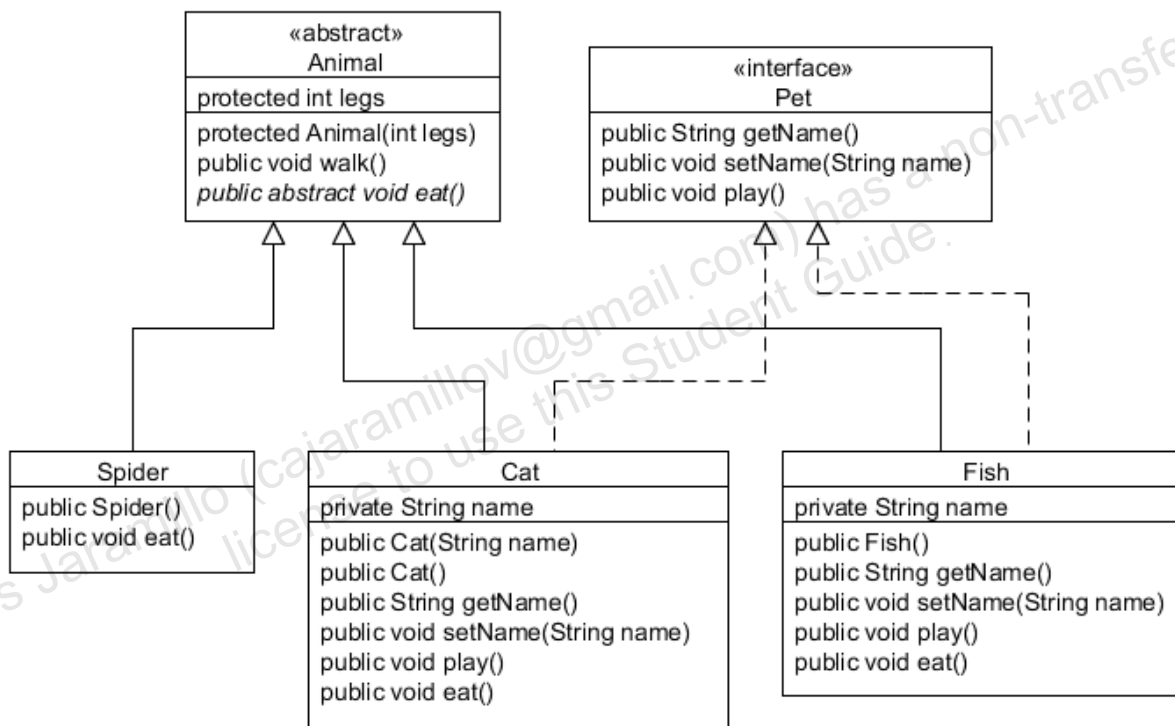
En esta práctica, creará una interfaz y la implantará.

### Supuestos

Ha revisado la sección sobre la interfaz de esta lección.

### Resumen

Se le ha asignado un proyecto que contiene una clase abstract denominada `Animal`. Creará una jerarquía de animales con raíz en la clase `Animal`. Varias de las clases de animales implantan una interfaz denominada `Pet`, que usted creará.



### Tareas

1. Abra el proyecto `Pet` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\06-Interfaces\practices`.
  - c. Seleccione `Pet` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debe aparecer texto en la ventana de salida.

4. Revise las clases `Animal` y `Spider`.
  - a. Abra el archivo `Animal.java` (en el paquete `com.example`).
  - b. Revise la clase `Animal` de abstract, que tendrá que ampliar.
  - c. Abra el archivo `Spider.java` (en el paquete `com.example`).
  - d. La clase `Spider` es un ejemplo de ampliación de la clase `Animal`.
5. Cree una nueva interfaz Java: `Pet` en el paquete `com.example`.
6. Codifique la interfaz de `Pet`. Esta interfaz debe incluir tres firmas de métodos:

```
public String getName();
public void setName(String name);
public void play();
```

7. Cree una nueva clase Java: `Fish` en el paquete `com.example`.
8. Codifique la clase `Fish`.
  - a. Esta clase debe ampliar la clase `Animal` e implantar la interfaz de `Pet`.

```
public class Fish extends Animal implements Pet
```

- b. Para terminar esta clase, cree:

- Un campo `String` denominado `name`.

```
private String name;
```

- Métodos `getter` y `setter` para el campo `name`.

```
@Override
public String getName() {
 return name;
}

@Override
public void setName(String name) {
 this.name = name;
}
```

- Un constructor sin argumentos que transfiera un valor 0 al constructor principal.

```
public Fish() {
 super(0);
}
```

- Un método `play()` que imprima "Just keep swimming."

```
@Override
public void play() {
 System.out.println("Just keep swimming.");
}
```

- Un método `eat()` que imprima "Fish eat pond scum."

```
@Override
public void eat() {
 System.out.println("Fish eat pond scum.");
}
```

- Un método `walk()` que sustituya el método `walk` de la clase `Animal`. Primero debe llamar al método `walk` de la superclase para, a continuación, imprimir "Fish, of course, can't walk; they swim."

```
@Override
public void walk() {
 super.walk();
 System.out.println("Fish, of course, can't walk; they swim.");
}
```

9. Cree una nueva clase Java: `Cat` en el paquete `com.example`.

10. Codifique la clase `Cat`.

- a. Esta clase debe ampliar la clase `Animal` e implantar la interfaz de `Pet`.

```
public class Cat extends Animal implements Pet
```

- b. Para terminar esta clase, cree:

- Un campo `String` denominado `name`.
- Métodos `getter` y `setter` para el campo `name`.
- Un constructor que reciba un nombre `String` y transfiera un valor 4 al constructor principal.

```
public Cat(String name) {
 super(4);
 this.name = name;
}
```

- Un constructor sin argumentos que transfiera un valor "Fluffy" al otro constructor de esta clase.

```
public Cat() {
 this("Fluffy");
}
```

- Un método `play()` que imprima `name + " likes to play with string."`

```
@Override
public void play() {
 System.out.println(name + " likes to play with string.");
}
```

- Un método `eat()` que imprima "Cats like to eat spiders and fish."



11. Modifique la clase `PetMain`.

- a. Abra el archivo `PetMain.java` (en el paquete `com.example`).
- b. Revise el método `main`. Deben aparecer las siguientes líneas de código:

```
Animal a;
//test a spider with a spider reference
Spider s = new Spider();
s.eat();
s.walk();
//test a spider with an animal reference
a = new Spider();
a.eat();
a.walk();
```

- c. Agregue más líneas de código para probar las clases `Fish` y `Cat` creadas.
  - Intente usar todos los constructores.
  - Pruebe con todos los tipos de referencias posibles y determine los métodos que se pueden llamar con cada tipo de referencia. Utilice una referencia `Pet` mientras prueba las clases `Fish` y `Cat`.

```
Pet p;

Cat c = new Cat("Tom");
c.eat();
c.walk();
c.play();
a = new Cat();
a.eat();
a.walk();
p = new Cat();
p.setName("Mr. Whiskers");
p.play();

Fish f = new Fish();
f.setName("Guppy");
f.eat();
f.walk();
f.play();
a = new Fish();
a.eat();
a.walk();
```

d. Implante y pruebe el método `playWithAnimal(Animal a)`.

- Determine si el argumento implanta la interfaz de `Pet`. En caso afirmativo, convierta la referencia en un elemento `Pet` y llame al método `play`. En caso negativo, imprima un mensaje "Danger! Wild Animal".

```
public static void playWithAnimal(Animal a) {
 if(a instanceof Pet) {
 Pet p = (Pet)a;
 p.play();
 } else {
 System.out.println("Danger! Wild Animal");
 }
}
```

- Llame al método `playWithAnimal(Animal a)` al final del método `main`, transfiriendo cada uno de los tipos de animal.

```
playWithAnimal(s);
playWithAnimal(c);
playWithAnimal(f);
```

12. Ejecute el proyecto. Debe aparecer texto en la ventana de salida.

## Práctica 6-2: Nivel de resumen: Aplicación del patrón DAO

### Visión general

En esta práctica, utilizará una aplicación existente y refactorizará el código para implantar el patrón de diseño del objeto de acceso a datos (DAO).

### Supuestos

Ha revisado las secciones sobre DAO de esta lección.

### Resumen

Se le ha asignado un proyecto que implanta la lógica de una aplicación de recursos humanos. La aplicación permite crear, recuperar, actualizar, suprimir y mostrar objetos de tipo `Employee`. Los objetos `Employee` se almacenan actualmente en memoria con una matriz. Debe mover cualquier código relacionado con la persistencia de objetos `Employee` de la clase `Employee`. En posteriores prácticas, proporcionará otras implantaciones de persistencia. Posteriormente, no será necesario modificar esta aplicación al sustituir la implantación de persistencia.

### Tareas

1. Abra el proyecto `EmployeeMemoryDAO` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\06-Interfaces\practices`.
  - c. Seleccione `EmployeeMemoryDAO` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.

```
[C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit:
```

**Nota:** las fechas que introduzca deben tener el formato: 26 de noviembre de 1976  
Los ID de empleado deben oscilar entre 0 y 9.

4. Revise la clase `Employee`.
  - a. Abra el archivo `Employee.java` (en el paquete `com.example.model`).
  - b. Busque la matriz que se usa para almacenar empleados.

```
private static Employee[] employeeArray = new Employee[10];
```

**Nota:** el ID del empleado se usa como índice de matriz.
  - c. Busque cualquier método que utilice el campo `employeeArray`. Estos métodos se usan para mantener los objetos de empleados.
5. Cree un nuevo paquete `com.example.dao`.
6. Cree una interfaz de `EmployeeDAO` en el paquete `com.example.dao`.

7. Termine la interfaz `EmployeeDAO` con las siguientes firmas de métodos.

```
public void add(Employee emp);
public void update(Employee emp);
public void delete(int id);
public Employee findById(int id);
public Employee[] getAllEmployees();
```

8. Cree una clase `EmployeeDAOMemoryImpl` en el paquete `com.example.dao`.
9. Termine la clase `EmployeeDAOMemoryImpl`.
- Mueva el elemento `employeeArray` y los métodos relacionados de la clase `Employee` a la clase `EmployeeDAOMemoryImpl`.
  - Implante la interfaz de `EmployeeDAO`. Modifique los métodos que ha movido en el paso anterior para que pasen a ser los métodos que necesita la interfaz de `EmployeeDAO`.
- Indicación:** en este DAO, los métodos `add` y `update` funcionarán de la misma forma.

10. Actualice la clase `EmployeeTestInteractive`.
- La clase `EmployeeTestInteractive` ya no se compila; revise los errores.
  - Cree una instancia de `EmployeeDAO` en el método `main`. Utilice la interfaz de `EmployeeDAO` como tipo de referencia.
  - Modifique las líneas que contienen errores para usar la instancia de `EmployeeDAO`.
11. Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.

**Nota:** si bien es funcional, la clase `EmployeeTestInteractive` sigue estando ligada a un tipo específico de DAO, pues hace referencia al elemento `EmployeeDAO` al implantar la clase por nombre.

```
EmployeeDAO dao = new EmployeeDAOMemoryImpl();
```

En los siguientes pasos, eliminará este fuerte acoplamiento de la clase `EmployeeTestInteractive` creando una fábrica de DAO.

12. Modifique la interfaz de `EmployeeDAOMemoryImpl`.
- Agregue un constructor sin argumentos protegido.
13. Cree una clase `EmployeeDAOFactory` en el paquete `com.example.dao`.
14. Termine la clase `EmployeeDAOFactory`.
- Agregue un método que devuelva una instancia `EmployeeDAO`.

```
public EmployeeDAO createEmployeeDAO() {
 return new EmployeeDAOMemoryImpl();
}
```

15. Actualice la clase `EmployeeTestInteractive` para que use la clase `EmployeeDAOFactory`.
- Obtenga una instancia de `EmployeeDAOFactory` en el método `main`.
  - Obtenga una instancia de `EmployeeDAO` mediante la fábrica creada en el paso anterior.
16. Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.
- En el futuro, podrá cambiar el mecanismo de persistencia para usar una base de datos sin cambiar los tipos de referencia o las llamadas a métodos en la clase `EmployeeTestInteractive`.

## Práctica 6-2: Nivel detallado: Aplicación del patrón DAO

### Visión general

En esta práctica, utilizará una aplicación existente y refactorizará el código para implantar el patrón de diseño del objeto de acceso a datos (DAO).

### Supuestos

Ha revisado las secciones sobre DAO de esta lección.

### Resumen

Se le ha asignado un proyecto que implanta la lógica de una aplicación de recursos humanos. La aplicación permite crear, recuperar, actualizar, suprimir y mostrar objetos de tipo `Employee`. Los objetos `Employee` se almacenan actualmente en memoria con una matriz. Debe mover cualquier código relacionado con la persistencia de objetos `Employee` de la clase `Employee`. En posteriores prácticas, proporcionará otras implantaciones de persistencia. Posteriormente, no será necesario modificar esta aplicación al sustituir la implantación de persistencia.

### Tareas

1. Abra el proyecto `EmployeeMemoryDAO` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\06-Interfaces\practices`.
  - c. Seleccione `EmployeeMemoryDAO` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.

|                                                            |
|------------------------------------------------------------|
| [C]reate   [R]ead   [U]pdate   [D]elete   [L]ist   [Q]uit: |
|------------------------------------------------------------|

**Nota:** las fechas que introduzca deben tener el formato: 26 de noviembre de 1976  
Los ID de empleado deben oscilar entre 0 y 9.

4. Revise la clase `Employee`.
  - a. Abra el archivo `Employee.java` (en el paquete `com.example.model`).
  - b. Busque la matriz que se usa para almacenar empleados. Reubicará este campo en un paso posterior.

|                                                                        |
|------------------------------------------------------------------------|
| <pre>private static Employee[] employeeArray = new Employee[10];</pre> |
|------------------------------------------------------------------------|

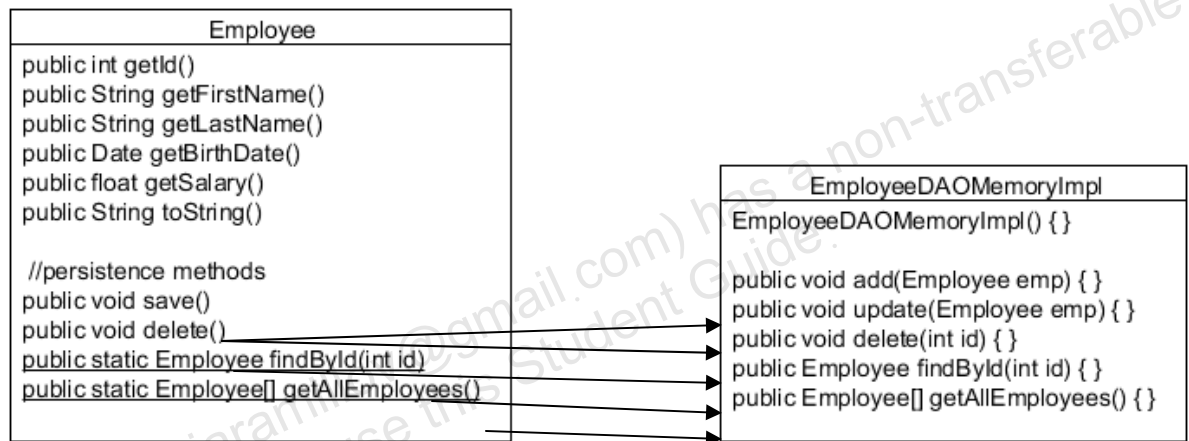
**Nota:** el ID del empleado se usa como índice de matriz.

- c. Busque los métodos `save`, `delete`, `findById` y `getAllEmployees` que utilicen el campo `employeeArray`. Estos métodos se usan para mantener los objetos de empleados. Reubicará estos métodos en un paso posterior.
5. Cree un nuevo paquete `com.example.dao`.
6. Cree una interfaz de `EmployeeDAO` en el paquete `com.example.dao`.

7. Termine la interfaz `EmployeeDAO` con las siguientes firmas de métodos. Agregue las sentencias de importación necesarias.

```
public void add(Employee emp);
public void update(Employee emp);
public void delete(int id);
public Employee findById(int id);
public Employee[] getAllEmployees();
```

8. Cree una clase `EmployeeDAOMemoryImpl` en el paquete `com.example.dao`.
9. Termine la clase `EmployeeDAOMemoryImpl`.
- a. Mueva el elemento `employeeArray` y los métodos relacionados de la clase `Employee` a la clase `EmployeeDAOMemoryImpl`.



- b. Implante la interfaz de `EmployeeDAO`. Modifique los métodos que ha movido en el paso anterior para que pasen a ser los métodos que necesita la interfaz de `EmployeeDAO`.
- El método `save` se convierte en el método `add` y se modifica para que tenga un parámetro `Employee`.
  - El método `save` se duplica para convertirse en el método `update` y se modifica para que tenga un parámetro `Employee`.
  - El método `delete` se modifica para incluir un parámetro `id`.
  - El método `findById` ya no es de tipo `static`.
  - El método `getAllEmployees` ya no es de tipo `static`.

```
public class EmployeeDAOMemoryImpl implements EmployeeDAO {
 private static Employee[] employeeArray = new Employee[10];

 public void add(Employee emp) {
 employeeArray[emp.getId()] = emp;
 }

 public void update(Employee emp) {
 employeeArray[emp.getId()] = emp;
 }

 public void delete(int id) {
 employeeArray[id] = null;
 }

 public Employee findById(int id) {
 return employeeArray[id];
 }

 public Employee[] getAllEmployees() {
 List<Employee> emps = new ArrayList<>();
 for (Employee e : employeeArray) {
 if (e != null) {
 emps.add(e);
 }
 }
 return emps.toArray(new Employee[0]);
 }
}
```

10. Actualice la clase `EmployeeTestInteractive`.

- La clase `EmployeeTestInteractive` ya no se compila; revise los errores.
- Cree una instancia de `EmployeeDAO` en el método `main`. Utilice la interfaz de `EmployeeDAO` como tipo de referencia. Sustituya la línea:

```
//TODO create dao
```

Por:

```
EmployeeDAO dao = new EmployeeDAOMemoryImpl();
```



- c. Modifique las líneas que contienen errores para usar la instancia de `EmployeeDAO`. Por ejemplo:

```
case 'C':
 emp = inputEmployee(in);
 dao.add(emp);
 System.out.println("Successfully added Employee Record: " +
emp.getId());
 System.out.println("\n\nCreated " + emp);
 break;
```

**Nota:** también puede eliminar la ahora innecesaria búsqueda del objeto employee que está presente al suprimir un empleado.

```
// Find this Employee record
emp = null;
emp = Employee.findById(id);
if (emp == null) {
 System.out.println("\n\nEmployee " + id + " not found");
 break;
}
```

11. Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.

**Nota:** si bien es funcional, la clase `EmployeeTestInteractive` sigue estando ligada a un tipo específico de DAO, pues hace referencia al elemento `EmployeeDAO` al implantar la clase por nombre.

```
EmployeeDAO dao = new EmployeeDAOMemoryImpl();
```

En los siguientes pasos, eliminará este fuerte acoplamiento de la clase `EmployeeTestInteractive` creando una fábrica de DAO.

12. Modifique la interfaz de `EmployeeDAOMemoryImpl`.

- a. Agregue un constructor sin argumentos protegido.

```
EmployeeDAOMemoryImpl() {
}
```

13. Cree una clase `EmployeeDAOFactory` en el paquete `com.example.dao`.

14. Termine la clase `EmployeeDAOFactory`.

- Agregue un método que devuelva una instancia `EmployeeDAO`.

```
public class EmployeeDAOFactory {

 public EmployeeDAO createEmployeeDAO() {
 return new EmployeeDAOMemoryImpl();
 }

}
```

15. Actualice la clase `EmployeeTestInteractive` para que use la clase `EmployeeDAOFactory`.

a. Obtenga una instancia de `EmployeeDAOFactory` en el método `main`. Sustituya la línea:

```
//TODO create factory
```

Por:

```
EmployeeDAOFactory factory = new EmployeeDAOFactory();
```

b. Obtenga una instancia de `EmployeeDAO` mediante la fábrica creada en el paso anterior. Sustituya la línea:

```
EmployeeDAO dao = new EmployeeDAOMemoryImpl();
```

Por:

```
EmployeeDAO dao = factory.createEmployeeDAO();
```

c. Corrija cualquier importación.

16. Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.

En el futuro, podrá cambiar el mecanismo de persistencia para usar una base de datos sin cambiar los tipos de referencia o las llamadas a métodos en la clase `EmployeeTestInteractive`. Observe que ninguna de las clases `*MemoryImpl` las usa el nombre desde dentro de la clase `EmployeeTestInteractive`.

## (Opcional) Práctica 6-3: Implantación de la composición

---

### Visión general

En esta práctica, utilizará una aplicación existente y la refactorizará para utilizar una composición.

### Supuestos

Ha revisado las secciones sobre la interfaz y la composición de esta lección.

### Resumen

Se le ha asignado un proyecto pequeño que representa una jerarquía de animales con raíz en la clase `Animal`. Varias de las clases animal implantan una interfaz denominada `Pet`. Este proyecto es una implantación finalizada de la práctica “Implantación de una interfaz”.

Existen algunos posibles problemas con el diseño del proyecto existente. Si deseara restringir el nombre de una mascota a menos de 20 caracteres, ¿cuántas clases tendría que modificar? ¿Empeoraría este problema si se agregaran nuevos animales?

Si algunos tipos de animales, como el pez, no pueden andar, ¿deberían tener un método `walk`?

### Tareas

1. Abra el proyecto `PetComposition` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\06-Interfaces\practices`.
  - c. Seleccione `PetComposition` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.

2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debe aparecer la salida en la ventana correspondiente.
4. Centralice toda la funcionalidad de “name”.

A todas las mascotas se les pueden asignar nombres, pero tal vez desee asignar nombres a objetos que no puedan jugar. Por ejemplo, podría asignarle a un balón de volley el nombre “Wilson”. El diseño debería reflejar este hecho.

- a. Cree una interfaz de `Nameable` (en el paquete `com.example`).
- b. Termine la interfaz de `Nameable` con las firmas de métodos `setName` y `getName`.

```
public interface Nameable {

 public void setName(String name);

 public String getName();

}
```

- c. Cree una clase `NameableImpl` (en el paquete `com.example`).

- d. Termine la clase `NameableImpl`. Debe:
  - Implantar la interfaz de `Nameable`.
  - Contener un campo `String` privado denominado `name`.
  - Solo aceptar nombres con menos de 20 caracteres de longitud.
  - Imprimir "Name too long" si alguno de los nombres es muy largo.
- e. Modifique la interfaz de `Pet`.
  - Amplíe la interfaz de `Nameable`.
  - Elimine las firmas de método `getName` y `setName` (ahora se han heredado).
- f. Modifique las clases `Fish` y `Cat` para usar la composición.
  - Suprima el campo `name`.
  - Suprima los métodos `getName` y `setName` existentes.
  - Agregue un campo `Nameable`.

```
private Nameable nameable = new NameableImpl();
```

- Agregue los métodos `getName` y `setName`, que deleguen en el campo `Nameable`.  
**Indicación:** coloque el cursor dentro de los corchetes angulares de la clase. Abra el menú Source, seleccione Insert Code, Delegate Method, active la casilla de control `Nameable` y haga clic en el botón Generate.
- Sustituya cualquier uso del campo `name` antiguo por llamadas a los métodos `getName` y `setName`.

5. Centralice toda la funcionalidad de `walk`.

Solo algunos animales pueden caminar. Elimine el método `walk` de la clase `Animal` y utilice interfaces y la composición para facilitar el uso de `walk`.

- a. Cree una interfaz de `Ambulatory` (en el paquete `com.example`).
- b. Termine la interfaz de `Ambulatory` con la firma de método `walk`.

```
public interface Ambulatory {

 public void walk();

}
```

- c. Cree una clase `AmbulatoryImpl` (en el paquete `com.example`).
- d. Termine la clase `AmbulatoryImpl`. Debe:
  - Implantar la interfaz de `Ambulatory`.
  - Contener un campo `int` privado denominado `legs`.
  - Contener un constructor de un solo argumento que reciba un valor `int` que se almacene en el campo `legs`.
  - Contener un método `walk`.

```
public void walk() {
 System.out.println("This animal walks on " + legs + "
legs.");
}
```

- e. Suprima el método `walk` de la clase `Fish`.

f. Modifique las clases `Spider` y `Cat` para usar la composición.

- Agregue un campo `Ambulatory`.

```
private Ambulatory ambulatory;
```

- Agregue un método `walk` que delegue en el campo `Ambulatory`.

**Indicación:** coloque el cursor dentro de los corchetes angulares de la clase. Abra el menú `Source`, seleccione `Insert Code`, `Delegate Method`, active la casilla de control `Ambulatory` y haga clic en el botón `Generate`.

g. Inicialice el campo `ambulatory` de los constructores `Spider` y `Cat`. Por ejemplo:

```
public Spider() {
 ambulatory = new AmbulatoryImpl(8);
}
```

6. Modifique la clase `PetMain` para probar el método `walk`. Solo se puede llamar al método `walk` en las referencias `Spider`, `Cat` o `Ambulatory`.

Carlos Jaramillo (cajaramillo@gmail.com) has a non-transferable  
license to use this Student Guide.

## **Prácticas de la lección 7: Elementos genéricos y recopilaciones**

### **Capítulo 7**

## Prácticas de la lección 7: Visión general

---

### Visión general de las prácticas

En estas prácticas, utilizará elementos genéricos y recopilaciones para practicar con los conceptos que se tratan en la clase teórica. Para cada práctica, se le proporcionará un proyecto de NetBeans. Realice el proyecto como se indica en las instrucciones.



## Práctica 7-1: Nivel de resumen: Recuento de números de artículo mediante elementos HashMap

### Visión general

En esta práctica, utilizará la recopilación HashMap para contar una lista de números de artículo.

### Supuestos

Ha revisado la sección sobre recopilaciones de esta lección.

### Resumen

Se le ha pedido que cree un programa simple para contar una lista de números de artículo de una longitud arbitraria. Con la siguiente asignación de números de artículo a descripciones, contará el número de cada artículo. Producirá un informe que muestre el recuento de cada artículo ordenado por la descripción de producto del artículo. La asignación de número de artículo a descripción es la siguiente:

| Número de artículo | Descripción      |
|--------------------|------------------|
| 1S01               | Blue Polo Shirt  |
| 1S02               | Black Polo Shirt |
| 1H01               | Red Ball Cap     |
| 1M02               | Duke Mug         |

Una vez terminado, el informe debe tener el siguiente aspecto:

```
=== Product Report ===
Name: Black Polo Shirt Count: 6
Name: Blue Polo Shirt Count: 7
Name: Duke Mug Count: 3
Name: Red Ball Cap Count: 5
```

### Tareas

Abra el proyecto Generics-Practice01 y realice los siguientes cambios.

1. Para la clase `ProductCounter`, agregue dos campos de asignación privados. La primera asignación cuenta los números de artículo. El orden de las claves no importa. La segunda asignación almacena la asignación de la descripción del producto al número de artículo. Las claves se deben ordenar alfabéticamente mediante la descripción de la segunda asignación.
2. Cree un constructor de un argumento que acepte un elemento `Map` como parámetro. La asignación que almacena la asignación de descripción a número de artículo se debe transferir aquí.

3. Cree un método `processList()` para procesar una lista de números de artículo `String`. Utilice un elemento `HashMap` para almacenar el recuento actual según el número de artículo.

```
public void processList(String[] list){ }
```

4. Cree un método `printReport()` para imprimir los resultados.

```
public void printReport(){ }
```

5. Agregue código al método `main` para imprimir los resultados.
6. Ejecute la clase `ProductCounter.java` para asegurarse de que el programa produce la salida que desea.

## Práctica 7-1: Nivel detallado: Recuento de números de artículo mediante elementos HashMap

### Visión general

En esta práctica, utilizará la recopilación HashMap para contar una lista de números de artículo.

### Supuestos

Ha revisado la sección sobre recopilaciones de esta lección.

### Resumen

Se le ha pedido que cree un programa simple para contar una lista de números de artículo de una longitud arbitraria. Con la siguiente asignación de números de artículo a descripciones, contará el número de cada artículo. Producirá un informe que muestre el recuento de cada artículo ordenado por la descripción de producto del artículo. La asignación de número de artículo a descripción es la siguiente:

| Número de artículo | Descripción      |
|--------------------|------------------|
| 1S01               | Blue Polo Shirt  |
| 1S02               | Black Polo Shirt |
| 1H01               | Red Ball Cap     |
| 1M02               | Duke Mug         |

Una vez terminado, el informe debe tener el siguiente aspecto:

```
=== Product Report ===
Name: Black Polo Shirt Count: 6
Name: Blue Polo Shirt Count: 7
Name: Duke Mug Count: 3
Name: Red Ball Cap Count: 5
```

### Tareas

Abra el proyecto Generics-Practice01 y realice los siguientes cambios.

1. Para la clase `ProductCounter`, agregue dos campos de asignación privados. La primera asignación cuenta los números de artículo. El orden de las claves no importa. La segunda asignación almacena la asignación de la descripción del producto al número de artículo. Las claves se deben ordenar alfabéticamente mediante la descripción de la segunda asignación.

```
private Map<String, Long> productCountMap = new HashMap<>();
private Map<String, String> productNames = new TreeMap<>();
```

2. Cree un constructor de un argumento que acepte un elemento `Map` como parámetro.

```
public ProductCounter(Map productNames){
 this.productNames = productNames;
}
```

3. Cree un método `processList()` para procesar una lista de números de artículo `String`. Utilice un elemento `HashMap` para almacenar el recuento actual según el número de artículo.

```
public void processList(String[] list){
 long curVal = 0;
 for(String itemNumber:list){
 if (productCountMap.containsKey(itemNumber)){
 curVal = productCountMap.get(itemNumber);
 curVal++;
 productCountMap.put(itemNumber, new
Long(curVal));
 } else {
 productCountMap.put(itemNumber, new Long(1));
 }
 }
}
```

4. Cree un método `printReport()` para imprimir los resultados.

```
public void printReport(){
 System.out.println("=== Product Report ===");
 for (String key:productNames.keySet()){
 System.out.print("Name: " + key);
 System.out.println("\t\tCount: " +
productCountMap.get(productNames.get(key)));
 }
}
```

5. Agregue el siguiente código al método `main` para imprimir los resultados.

```
ProductCounter pc1 = new ProductCounter (productNames);
pc1.processList(parts);
pc1.printReport();
```

6. Ejecute la clase `ProductCounter.java` para asegurarse de que el programa produce la salida que desea.

## Práctica 7-2: Nivel de resumen: Coincidencia de paréntesis mediante Deque

---

### Visión general

En esta práctica, utilizará el objeto `Deque` para hacer coincidir los paréntesis en una sentencia de programación.

### Supuestos

Ha revisado la sección sobre recopilaciones de esta lección.

### Resumen

Utilizará la estructura de datos `Deque` como pila para hacer coincidir los paréntesis en una sentencia de programación. Se le proporcionarán varias líneas de ejemplo con sentencias lógicas. Pruebe las líneas para asegurarse de que los paréntesis coinciden; devolverán `true` si coinciden y `false` si no lo hacen.

Por ejemplo, la salida del programa podría ser parecida a esta:

```
Line 0 is valid
Line 1 is invalid
Line 2 is invalid
Line 3 is valid
```

### Tareas

Abra el proyecto `Generics-Practice02` y realice los siguientes cambios.

1. Modifique el método `processLine()` de `ParenMatcher.java` para que lea una línea y convierta la cadena en una matriz de carácter.
2. Realice bucle de la matriz. Inserte "(" en la pila. Cuando se detecte un ")", realice una operación `pop` en un "(" de la pila. Dos condiciones deben devolver `false`.
  - a. Si necesita llamar a una operación `pop` y la pila está vacía, el número de paréntesis no coincide; se devuelve `false`.
  - b. Si después de terminar el bucle, queda un "(" en la pila, devuelva `false`. El número de paréntesis no coincide.
3. Ejecute la clase `ParanMatcher.java` para asegurarse de que el programa produce la salida que desea.

## Práctica 7-2: Nivel detallado: Coincidencia de paréntesis mediante Deque

---

### Visión general

En esta práctica, utilizará el objeto `Deque` para hacer coincidir los paréntesis en una sentencia de programación.

### Supuestos

Ha revisado la sección sobre recopilaciones de esta lección.

### Resumen

Utilizará la estructura de datos `Deque` como pila para hacer coincidir los paréntesis en una sentencia de programación. Se le proporcionarán varias líneas de ejemplo con sentencias lógicas. Pruebe las líneas para asegurarse de que los paréntesis coinciden; devolverán `true` si coinciden y `false` si no lo hacen.

Por ejemplo, la salida del programa podría ser parecida a esta:

```
Line 0 is valid
Line 1 is invalid
Line 2 is invalid
Line 3 is valid
```

### Tareas

Abra el proyecto `Generics-Practice02` y realice los siguientes cambios.

1. Modifique el método `processLine()` de `ParenMatcher.java` para que lea una línea y convierta la cadena en una matriz de caracteres. Borre la pila y convierta la línea en una matriz de carácter.

```
stack.clear();
curLine = line.toCharArray();
```

2. En el mismo método, realice bucle de la matriz. Inserte "(" en la pila. Cuando se detecte un ")", realice la operación pop en un "(" de la pila. Si queda un "(" en la pila, o intenta realizar una operación pop en una pila vacía, los paréntesis no coincidirán; se devuelve `false`. En caso contrario, se devuelve `true`. Para ello, agregue el siguiente código al método `processLine` (sustituya la sentencia `return true;`).

```

 for (char c:curLine){
 switch (c) {
 case '(':stack.push(c);break;
 case ')':{
 if (stack.size() > 0){
 stack.pop();
 } else {
 return false;
 }
 break;
 }
 }
 }
 if (stack.size() > 0){
 return false; // Missing match invalid expression
 } else {
 return true; //
 }
 }

```

3. Ejecute la clase `ParanMatcher.java` para asegurarse de que el programa produce la salida que desea.

## Práctica 7-3: Nivel de resumen: Recuento de inventario y ordenación con elementos Comparator

---

### Visión general

En esta práctica, procesará transacciones relacionadas con camisas para una tienda de Duke's Choice. Calculará el nivel de inventario para una serie de camisas. A continuación, imprimirá los datos ordenados por descripción y por recuento de inventario.

### Supuestos

Ha revisado todo el contenido de esta lección.

### Resumen

Todas las tiendas Duke's Choice venden una serie de productos, entre ellos, camisas. En esta práctica, procesará las transacciones relacionadas con camisas y calculará los niveles de inventario. Una vez que se hayan calculado los niveles, imprimirá un informe ordenado por descripción y un informe ordenado por recuento de inventario. Creará dos clases que implanten la interfaz de Comparator para permitir la ordenación de las camisas por recuento y por descripción.

Por ejemplo, la salida del programa podría ser parecida a esta:

```
=== Inventory Report - Description ===
```

```
Shirt ID: P002
```

```
Description: Black Polo Shirt
```

```
Color: Black
```

```
Size: M
```

```
Inventory: 15
```

```
Shirt ID: P001
```

```
Description: Blue Polo Shirt
```

```
Color: Blue
```

```
Size: L
```

```
Inventory: 24
```

```
Shirt ID: P003
```

```
Description: Maroon Polo Shirt
```

```
Color: Maroon
```

```
Size: XL
```

```
Inventory: 20
```



Shirt ID: P004  
Description: Tan Polo Shirt  
Color: Tan  
Size: S  
Inventory: 19

=== Inventory Report - Count ===  
Shirt ID: P002  
Description: Black Polo Shirt  
Color: Black  
Size: M  
Inventory: 15

Shirt ID: P004  
Description: Tan Polo Shirt  
Color: Tan  
Size: S  
Inventory: 19

Shirt ID: P003  
Description: Maroon Polo Shirt  
Color: Maroon  
Size: XL  
Inventory: 20

Shirt ID: P001  
Description: Blue Polo Shirt  
Color: Blue  
Size: L  
Inventory: 24

## Tareas

Abra el proyecto Generics-Practice03 y realice los siguientes cambios.

1. Revise la clase `Shirt` y la interfaz de `InventoryCount` para ver cómo ha cambiado la clase `Shirt` para soportar funciones de inventario.
2. Revise la clase `DukeTransaction` para ver cómo se definen las transacciones para este programa.
3. Actualice la clase `SortShirtByCount Comparator` para ordenar las camisas por recuento.
4. Actualice la clase `SortShirtByDesc Comparator` para ordenar las camisas por descripción.

5. Actualice la clase `TestItemCounter` para procesar las camisas y las transacciones para producir el informe que desee.
  - Realice bucle de las transacciones y actualice el objeto `shirt` adecuado incluido en `polos map`.
  - Cada clase `Shirt` implanta la interfaz de `InventoryCount`. Utilice estos métodos y el campo `count` para aumentar y reducir los niveles de inventario.
  - Imprima la lista de camisas por descripción.
  - Imprima la lista de camisas por recuento.
6. Ejecute la clase `TestItemCounter.java` para asegurarse de que el programa produce la salida que desea.

## Práctica 7-3: Nivel detallado: Recuento de inventario y ordenación con elementos Comparator

---

### Visión general

En esta práctica, procesará transacciones relacionadas con camisas para una tienda de Duke's Choice. Calculará el nivel de inventario para una serie de camisas. A continuación, imprimirá los datos ordenados por descripción y por recuento de inventario.

### Supuestos

Ha revisado todo el contenido de esta lección.

### Resumen

Todas las tiendas Duke's Choice venden una serie de productos, entre ellos, camisas. En esta práctica, procesará las transacciones relacionadas con camisas y calculará los niveles de inventario. Una vez que se hayan calculado los niveles, imprimirá un informe ordenado por descripción y un informe ordenado por recuento de inventario. Creará dos clases que implanten la interfaz de Comparator para permitir la ordenación de las camisas por recuento y por descripción.

Por ejemplo, la salida del programa podría ser parecida a esta:

```
=== Inventory Report - Description ===
```

```
Shirt ID: P002
```

```
Description: Black Polo Shirt
```

```
Color: Black
```

```
Size: M
```

```
Inventory: 15
```

```
Shirt ID: P001
```

```
Description: Blue Polo Shirt
```

```
Color: Blue
```

```
Size: L
```

```
Inventory: 24
```

```
Shirt ID: P003
```

```
Description: Maroon Polo Shirt
```

```
Color: Maroon
```

```
Size: XL
```

```
Inventory: 20
```

Shirt ID: P004  
Description: Tan Polo Shirt  
Color: Tan  
Size: S  
Inventory: 19

=== Inventory Report - Count ===

Shirt ID: P002  
Description: Black Polo Shirt  
Color: Black  
Size: M  
Inventory: 15

Shirt ID: P004  
Description: Tan Polo Shirt  
Color: Tan  
Size: S  
Inventory: 19

Shirt ID: P003  
Description: Maroon Polo Shirt  
Color: Maroon  
Size: XL  
Inventory: 20

Shirt ID: P001  
Description: Blue Polo Shirt  
Color: Blue  
Size: L  
Inventory: 24

## Tareas

Abra el proyecto Generics-Practice03 y realice los siguientes cambios.

1. Revise la clase `Shirt` y la interfaz de `InventoryCount` para ver cómo ha cambiado la clase `Shirt` para soportar funciones de inventario.
2. Revise la clase `DukeTransaction` para ver cómo se definen las transacciones para este programa.

3. Actualice la clase `SortShirtByCount Comparator` para ordenar las camisas por recuento.

```
public class SortShirtByCount implements Comparator<Shirt>{
 public int compare(Shirt s1, Shirt s2){
 Long c1 = new Long(s1.getCount());
 Long c2 = new Long(s2.getCount());

 return c1.compareTo(c2);
 }
}
```

4. Actualice la clase `SortShirtByDesc Comparator` para ordenar las camisas por descripción.

```
public class SortShirtByDesc implements Comparator<Shirt>{
 public int compare(Shirt s1, Shirt s2){
 return
s1.getDescription().compareTo(s2.getDescription());
 }
}
```

5. Actualice la clase `TestItemCounter` para procesar las camisas y las transacciones para producir el informe que desee.

- Realice bucle de las transacciones y actualice el objeto shirt adecuado incluido en polos Map. Esto producirá un recuento de inventario para cada producto.

```
// Count the shirts
for (DukeTransaction transaction:transactions){
 if (polos.containsKey(transaction.getProductID())){
 currentShirt = polos.get(transaction.getProductID());
 } else {
 System.out.println("Error: Invalid part number");
 }

 switch (transaction.getTransactionType()) {
 case "Purchase":currentShirt.
addItem(transaction.getCount()); break;

 case "Sale":currentShirt.
removeItems(transaction.getCount()); break;

 default: System.out.println("Error: Invalid Transaction
Type"); continue;
 }
}
```

- Imprima la lista de camisas por descripción.

```
// Convert to List
List<Shirt> poloList = new ArrayList<>(polos.values());

// Init Comparators
Comparator sortDescription = new SortShirtByDesc();
Comparator sortCount = new SortShirtByCount();

// Print Results - Sort by Description
Collections.sort(poloList, sortDescription);
System.out.println("=== Inventory Report - Description ===");

for(Shirt shirt:poloList){
 System.out.println(shirt.toString());
}
```

- Imprima la lista de camisas por recuento.

```
// Print Results - Sort by Count
Collections.sort(poloList, sortCount);
System.out.println("=== Inventory Report - Count ===");

for(Shirt shirt:poloList){
 System.out.println(shirt.toString());
}
```

6. Ejecute la clase `TestItemCounter.java` para asegurarse de que el programa produce la salida que desea.

## **Prácticas de la lección 8: Procesamiento de cadenas**

### **Capítulo 8**

## Prácticas de la lección 8: Visión general

---

### Visión general de las prácticas

En estas prácticas, utilizará expresiones regulares y `String.split()` para manipular cadenas en Java. Para cada práctica, se le proporcionará un proyecto de NetBeans. Realice el proyecto como se indica en las instrucciones.



## Práctica 8-1: Nivel de resumen: Análisis de texto con `split()`

---

### Visión general

En esta práctica, analizará texto delimitado por comas y convertirá los datos en objetos Shirt.

### Supuestos

Ha participado en la clase teórica de esta lección.

### Resumen

Se le han proporcionado algunos datos de shirt delimitados por comas. Analizará los datos, los almacenará en objetos shirt e imprimirá los resultados. La salida del programa debería tener el siguiente aspecto.

```
=== Shirt List ===
Shirt ID: S001
Description: Black Polo Shirt
Color: Black
Size: XL

Shirt ID: S002
Description: Black Polo Shirt
Color: Black
Size: L

Shirt ID: S003
Description: Blue Polo Shirt
Color: Blue
Size: XL

Shirt ID: S004
Description: Blue Polo Shirt
Color: Blue
Size: M

Shirt ID: S005
Description: Tan Polo Shirt
Color: Tan
Size: XL

Shirt ID: S006
Description: Black T-Shirt
Color: Black
Size: XL
```

Shirt ID: S007  
Description: White T-Shirt  
Color: White  
Size: XL

Shirt ID: S008  
Description: White T-Shirt  
Color: White  
Size: L

Shirt ID: S009  
Description: Green T-Shirt  
Color: Green  
Size: S

Shirt ID: S010  
Description: Orange T-Shirt  
Color: Orange  
Size: S

Shirt ID: S011  
Description: Maroon Polo Shirt  
Color: Maroon  
Size: S

## Tareas

Abra el proyecto `StringsPractice01` y realice los siguientes cambios.

1. Edite el método `main` del archivo `StringSplitTest.java`.
2. Analice cada una de las líneas de la matriz `shirts`.
3. Convierta los datos de `shirt` en un elemento `List` de objetos `Shirt`.
4. Imprima la lista de objetos `shirt`.
5. Ejecute el archivo `StringSplitTest.java` y verifique que la salida es similar a la mostrada en la sección “Resumen” de esta práctica.

## Práctica 8-1: Nivel detallado: Análisis de texto con `split()`

---

### Visión general

En esta práctica, analizará texto delimitado por comas y convertirá los datos en objetos Shirt.

### Supuestos

Ha participado en la clase teórica de esta lección.

### Resumen

Se le han proporcionado algunos datos de shirt delimitados por comas. Analizará los datos, los almacenará en objetos shirt e imprimirá los resultados. La salida del programa debería tener el siguiente aspecto.

```
=== Shirt List ===
Shirt ID: S001
Description: Black Polo Shirt
Color: Black
Size: XL

Shirt ID: S002
Description: Black Polo Shirt
Color: Black
Size: L

Shirt ID: S003
Description: Blue Polo Shirt
Color: Blue
Size: XL

Shirt ID: S004
Description: Blue Polo Shirt
Color: Blue
Size: M

Shirt ID: S005
Description: Tan Polo Shirt
Color: Tan
Size: XL

Shirt ID: S006
Description: Black T-Shirt
Color: Black
Size: XL
```

Shirt ID: S007  
Description: White T-Shirt  
Color: White  
Size: XL

Shirt ID: S008  
Description: White T-Shirt  
Color: White  
Size: L

Shirt ID: S009  
Description: Green T-Shirt  
Color: Green  
Size: S

Shirt ID: S010  
Description: Orange T-Shirt  
Color: Orange  
Size: S

Shirt ID: S011  
Description: Maroon Polo Shirt  
Color: Maroon  
Size: S

## Tareas

Abra el proyecto StringsPractice01 y realice los siguientes cambios.

1. Edite el método `main` del archivo `StringSplitTest.java`.
2. Analice cada una de las líneas de la matriz `shirts`. Cree un objeto `Shirt` para cada línea y agregue el elemento `Shirt` a un elemento `List`. Un bucle `for` para realizar estos pasos podría ser el siguiente:

```
for(String curLine:shirts){
 String[] e = curLine.split(",");
 shirtList.add(new Shirt(e[0], e[1], e[2], e[3]));
}
```

3. Imprima la lista de objetos shirt. Un bucle para realizar esta operación podría tener este aspecto:

```
System.out.println("=== Shirt List ===");
for (Shirt shirt:shirtList){
 System.out.println(shirt.toString());
}
```

4. Ejecute el archivo `StringSplitTest.java` y verifique que la salida es similar a la mostrada en la sección “Resumen” de esta práctica.

## Práctica 8-2: Nivel de resumen: Creación de un programa de búsqueda de expresiones regulares

---

### Visión general

En esta práctica, creará un programa que buscará un archivo de texto mediante expresiones regulares.

### Supuestos

Ha participado en la clase teórica de esta lección.

### Resumen

Crearé una aplicación simple que realizará un bucle en un archivo de texto (`gettys.html`) y que buscará texto mediante expresiones regulares. Si se encuentra el texto que se desea en una línea, imprimirá el número de línea y el texto de línea. Por ejemplo, si ha realizado una búsqueda de “<h4>”, la salida sería:

```
9 <h4>Abraham Lincoln</h4>
10 <h4>Thursday, November 19, 1863</h4>
```

### Tareas

Abra el proyecto `StringsPractice02` y realice los siguientes cambios. Tenga en cuenta que se le ha proporcionado el código para leer un archivo.

**Nota:** el archivo `gettys.html` está en la raíz de la carpeta de proyectos. Para examinar el archivo, con el proyecto abierto, haga clic en el separador Files. Haga clic dos veces en el archivo para abrirlo y examinar su contenido.

1. Edite el archivo `FindText.java`.
2. Cree un elemento `Pattern` y un campo `Matcher`.
3. Genere un elemento `Matcher` según el objeto `Pattern` proporcionado.
4. Busque cada una de las líneas del patrón proporcionado.
5. Imprima el número de línea y la línea que tenga texto coincidente.

6. Ejecute el archivo `FindText.java` y busque estos patrones.
  - Todas las líneas que contengan: `<h4>`.
  - Todas las líneas que contengan la palabra “to” (por ejemplo, la línea 17 no se debe seleccionar).
  - Todas las líneas que contengan `'class="line"'`.
  - Todas las líneas que contengan un par de corchetes `{ }`.
  - Las líneas que empiecen por `<p` o `<d`.
  - Las líneas que solo contengan etiquetas de cierre HTML (por ejemplo, `</div>`).
7. (Opcional) Modifique el programa para que acepte el nombre de archivo y el patrón de expresión regular en la línea de comandos.

## Práctica 8-2: Nivel detallado: Creación de un programa de búsqueda de expresiones regulares

---

### Visión general

En esta práctica, creará un programa que buscará un archivo de texto mediante expresiones regulares.

### Supuestos

Ha participado en la clase teórica de esta lección.

### Resumen

Crearé una aplicación simple que realizará un bucle en un archivo de texto y que buscará texto mediante expresiones regulares. Si se encuentra el texto que se desea en una línea, imprimirá el número de línea y el texto de línea. Por ejemplo, si ha realizado una búsqueda de "<h4>", la salida sería:

```
9 <h4>Abraham Lincoln</h4>
10 <h4>Thursday, November 19, 1863</h4>
```

### Tareas

Abra el proyecto `StringsPractice02` y realice los siguientes cambios. Tenga en cuenta que se le ha proporcionado el código para leer un archivo.

**Nota:** el archivo `gettys.html` está en la raíz de la carpeta de proyectos. Para examinar el archivo, con el proyecto abierto, haga clic en el separador Files. Haga clic dos veces en el archivo para abrirlo y examinar su contenido.

1. Edite el archivo `FindText.java`.
2. Cree campos para un elemento `Pattern` y un objeto `Matcher`.

```
private Pattern pattern;
private Matcher m;
```

3. Fuera del bucle de búsqueda, cree e inicialice el objeto de patrón.

```
pattern = Pattern.compile("<h4>");
```

4. Dentro del bucle de búsqueda, genere un elemento `Matcher` según el objeto `Pattern` proporcionado.

```
m = pattern.matcher(line);
```



5. Dentro del bucle de búsqueda, busque cada una de las líneas del patrón proporcionado. Imprima el número de línea y la línea que tenga texto coincidente.

```
if (m.find()) {
 System.out.println(" " + c + " " + line);
}
```

6. Ejecute el archivo `FindText.java` y busque estos patrones.

- Todas las líneas que contengan: `<h4>`.

```
pattern = Pattern.compile("<h4>");
```

- Todas las líneas que contengan la palabra "to" (por ejemplo, la línea 17 no se debe seleccionar).

```
pattern = Pattern.compile("\\bto\\b");
```

- Todas las líneas que empiecen por 4 espacios.

```
pattern = Pattern.compile("^\\s{4}");
```

- Las líneas que empiecen por "`<p`" o "`<d`".

```
pattern = Pattern.compile("^<[p|d]");
```

- Las líneas que solo contengan etiquetas de cierre HTML (por ejemplo, "`</div>`").

```
pattern = Pattern.compile("^</.*?>$");
```

7. (Opcional) Modifique el programa para que acepte el nombre de archivo y el patrón de expresión regular en la línea de comandos.

## Práctica 8-3: Nivel de resumen: Transformación de HTML mediante expresiones regulares

### Visión general

En esta práctica, utilizará expresiones regulares para transformar etiquetas `<p>` en etiquetas `<span>`.

### Supuestos

Ha participado en la clase teórica de esta lección.

### Resumen

Ha decidido que desea cambiar el formato del archivo `gettys.html`. En lugar de usar etiquetas `<p>`, debe usar etiquetas `<span>`. Además, considera que el valor de la clase debería ser "sentence" en lugar de "line". Utilizará expresiones regulares para buscar las líneas que desea cambiar. A continuación, utilizará expresiones regulares para transformar las etiquetas y los atributos de la forma descrita. Las líneas transformadas deben aparecer en la consola. La salida se debe parecer a lo siguiente:

```
13 Four score and seven years ago our
 fathers brought forth on this continent a new nation, conceived
 in liberty, and dedicated to the proposition that all men are
 created equal.
14 Now we are engaged in a great civil
 war, testing whether that nation, or any nation, so conceived
 and so dedicated, can long endure.
15 We are met on a great battle-field of
 that war.
16 We have come to dedicate a portion of
 that field, as a final resting place for those who here gave
 their lives that that nation might live.
17 It is altogether fitting and proper
 that we should do this.
21 But, in a larger sense, we can not
 dedicate, we can not consecrate, we can not hallow this
 ground.
...
```

Un posible acercamiento al problema podría ser dividir el algoritmo en tres pasos.

1. Divida la línea en tres partes: la etiqueta de inicio, el contenido y la etiqueta de finalización.
2. Sustituya las etiquetas actuales por una nueva etiqueta.
3. Sustituya el valor de atributo por un nuevo valor de atributo.

A continuación, devuelva la línea con el nuevo formato.

Las firmas de método para sustituir la etiqueta y los atributos podrían tener este aspecto:

```
public String replaceTag(String tag, String targetTag,
String replaceTag){ }
public String replaceAttribute(String tag, String attribute,
String value){ }
```

## Tareas

Abra el proyecto `StringsPractice03` y realice los siguientes cambios. Tenga en cuenta que se le ha proporcionado el código para leer un archivo.

1. Edite el archivo `SearchReplace.java`.
2. Cree un objeto `Pattern` para que coincida con toda la línea.
3. Conforme realiza bucle del archivo, haga lo siguiente:
  - Cree un objeto `Matcher` para que coincida con la línea actual.
  - Ejecute el método `find()` para buscar una coincidencia.
  - Si hay una coincidencia, sustituya las etiquetas de inicio y de finalización.
  - Sustituya el atributo.
4. Cree un método que sustituirá el contenido de cualquiera de las etiquetas.
5. Cree un método que sustituirá el atributo de una etiqueta.
6. Ejecute el archivo `SearchReplace.java` y produzca la salida que aparece en la sección "Resumen" de esta práctica.

## Práctica 8-3: Nivel detallado: Transformación de HTML mediante expresiones regulares

---

### Visión general

En esta práctica, utilizará expresiones regulares para transformar etiquetas `<p>` en etiquetas `<span>`.

### Supuestos

Ha participado en la clase teórica de esta lección.

### Resumen

Ha decidido que desea cambiar el formato del archivo `gettys.html`. En lugar de usar etiquetas `<p>`, debe usar etiquetas `<span>`. Además, considera que el valor de la clase debería ser "sentence" en lugar de "line". Utilizará expresiones regulares para buscar las líneas que desea cambiar. A continuación, utilizará expresiones regulares para transformar las etiquetas y los atributos de la forma descrita. Las líneas transformadas deben aparecer en la consola. La salida se debe parecer a lo siguiente:

```
13 Four score and seven years ago our
 fathers brought forth on this continent a new nation, conceived
 in liberty, and dedicated to the proposition that all men are
 created equal.
14 Now we are engaged in a great civil
 war, testing whether that nation, or any nation, so conceived
 and so dedicated, can long endure.
15 We are met on a great battle-field of
 that war.
16 We have come to dedicate a portion of
 that field, as a final resting place for those who here gave
 their lives that that nation might live.
17 It is altogether fitting and proper
 that we should do this.
21 But, in a larger sense, we can not
 dedicate, we can not consecrate, we can not hallow this
 ground.
...
```

Un posible acercamiento al problema podría ser dividir el algoritmo en tres pasos.

1. Divida la línea en tres partes: la etiqueta de inicio, el contenido y la etiqueta de finalización.
2. Sustituya las etiquetas actuales por una nueva etiqueta.
3. Sustituya el valor de atributo por un nuevo valor de atributo.

A continuación, devuelva la línea con el nuevo formato.

Las firmas de método para sustituir la etiqueta y los atributos podrían tener este aspecto:

```
public String replaceTag(String tag, String targetTag,
String replaceTag){ }
public String replaceAttribute(String tag, String attribute,
String value){ }
```

## Tareas

Abra el proyecto `StringsPractice03` y realice los siguientes cambios. Tenga en cuenta que se le ha proporcionado el código para leer un archivo.

1. Edite el archivo `SearchReplace.java`.
2. Cree un objeto `Pattern` para que coincida con toda la línea.

```
Pattern pattern1 = Pattern.compile("<" + targetTag +
".*?>)(.*?)(</" + targetTag + ".*?>");
```

3. Conforme realiza bucle del archivo, haga lo siguiente:

- Cree un objeto `Matcher` para que coincida con la línea actual.

```
Matcher m = pattern1.matcher(line);
```

- Ejecute el método `find()` para buscar una coincidencia. Si hay una coincidencia, sustituya las etiquetas de inicio y de finalización. Sustituya el atributo.

```
if (m.find()) {
 String newStart = replaceTag(m.group(1), targetTag,
replaceTag);
 newStart = replaceAttribute(newStart, attribute, value);
 String newEnd = replaceTag(m.group(3), targetTag, replaceTag);

 String newLine = newStart + m.group(2) + newEnd;
 System.out.printf("%3d %s\n", c, newLine);
}
```

4. Cree un método que sustituirá el contenido de cualquiera de las etiquetas.

```
public String replaceTag(String tag, String targetTag, String
replaceTag){
 Pattern p = Pattern.compile(targetTag); // targetTag is
regex
 Matcher m = p.matcher(tag); // tag is text to replace
 if (m.find()) {
```

```
 return m.replaceFirst(replaceTag); // swap target with
replace
 }
 return targetTag;
}
```

5. Cree un método que sustituirá el atributo de una etiqueta.

```
public String replaceAttribute(String tag, String attribute,
String value){
 Pattern p = Pattern.compile(attribute + "=" + "\\.*?\\");
 Matcher m = p.matcher(tag); // tag is text to replace
 if (m.find()) {
 return m.replaceFirst(attribute + "=" + "\"" + value +
"\");
 }
 return tag;
}
```

6. Ejecute el archivo SearchReplace.java y produzca la salida que aparece en la sección "Resumen" de esta práctica.

## **Prácticas de la lección 9: Excepciones y afirmaciones**

### **Capítulo 9**

## Prácticas de la lección 9: Visión general

---

### Visión general de las prácticas

En estas prácticas, utilizará sentencias `try-catch`, ampliará la clase `Exception` y utilizará las palabras clave `throw` y `throws`.



## Práctica 9-1: Nivel de resumen: Captura de excepciones

### Visión general

En esta práctica, creará un nuevo proyecto y capturará excepciones comprobadas y no comprobadas.

### Supuestos

Ha revisado la sección sobre manejo de excepciones de esta lección.

### Resumen

Crearé un proyecto que lee datos de un archivo. Se le proporcionará el código de lectura de archivos. Su tarea es agregar el código de manejo de excepciones adecuado.

### Tareas

1. Cree un nuevo proyecto `ExceptionPractice` como proyecto principal.
  - a. Seleccione `File > New Project`.
  - b. Seleccione `Java` en `Categories` y `Java Application` en `Projects`. Haga clic en el botón `Next`.
  - c. Introduzca la siguiente información en el cuadro de diálogo "Name and Location":
    - Project Name: `ExceptionPractice`
    - Project Location: `D:\labs\09-Exceptions\practices`
    - (activada) Create Main Class: `com.example.ExceptionMain`
    - (activada) Set as Main Project
  - d. Haga clic en el botón *Finish*.
2. Agregue la siguiente línea al método `main`.

```
System.out.println("Reading from file:" + args[0]);
```

**Nota:** se utilizará un argumento de línea de comandos para especificar el archivo que se leerá. Actualmente no se proporcionan argumentos; no corrija aún este descuido.

3. Ejecute el proyecto. Debe aparecer un mensaje de error similar a:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
 at com.example.ExceptionMain.main(ExceptionMain.java:7)
Java Result: 1
```

4. Rodee la línea `println` del código agregado con una sentencia `try-catch`.
  - La cláusula `catch` debe:
    - Aceptar un parámetro del tipo `ArrayIndexOutOfBoundsException`.
    - Imprimir el mensaje: `"No file specified, quitting!"`.
    - Salir de la aplicación con un estado de salida 1 usando el método `static` adecuado en la clase `System`.

**Nota:** como el compilador no le ha forzado a manejar o a declarar la excepción `ArrayIndexOutOfBoundsException`, se trata de una excepción no comprobada. Normalmente, no es necesario que use un bloque `try-catch` para tratar una excepción no comprobada. Comprobar la longitud de la matriz `args` es otro método con el que se puede garantizar que se ha proporcionado un argumento de línea de comandos.

5. Ejecute el proyecto. Debe aparecer un mensaje de error similar a:

```
No file specified, quitting!
Java Result: 1
```

6. Agregue un argumento de línea de comandos al proyecto.
  - a. Haga clic con el botón derecho en el proyecto `ExceptionPractice` y seleccione `Properties`.
  - b. En el cuadro de diálogo `Project Properties`, seleccione la categoría `Run`.
  - c. En el campo `Arguments`, introduzca un valor:  
`D:\labs\resources\DeclarationOfIndependence.txt`
  - d. Haga clic en el botón `OK`.

7. Ejecute el proyecto. Debe ver un mensaje similar a:

```
Reading from
file:D:\labs\resources\DeclarationOfIndependence.txt
```

**Advertencia:** ejecutar el proyecto no es lo mismo que ejecutar el archivo. El argumento de la línea de comandos solo se transferirá al método `main` si ejecuta el proyecto.

8. Agregue las siguientes líneas de código al método `main` bajo las líneas que ha agregado previamente:

```
BufferedReader b =
 new BufferedReader(new FileReader(args[0]));
String s = null;
while((s = b.readLine()) != null) {
 System.out.println(s);
}
```

9. Para ejecutar el asistente de `Fix Imports`, haga clic con el botón derecho en la ventana de código fuente.
10. Ahora deben aparecer errores del compilador en algunas de las líneas que acaba de agregar. Estas líneas podrían generar excepciones comprobadas. Al crear manualmente el proyecto o mantener el cursor sobre la línea con errores, debe aparecer un mensaje similar a este:

```
unreported exception FileNotFoundException; must be caught or
declared to be thrown
```

11. Modifique las propiedades del proyecto para permitir la sentencia `try-with-resources`.
  - a. Haga clic con el botón derecho en el proyecto `ExceptionPractice` y seleccione `Properties`.
  - b. En el cuadro de diálogo `Project Properties`, seleccione la categoría `Sources`.
  - c. En la lista desplegable `Source/Binary Format`, seleccione `JDK 7`.
  - d. Haga clic en el botón `OK`.
12. Rodee el código de E/S del archivo que se proporciona en el paso 8 con una sentencia `try-with-resources`.
  - La línea con la que se crea e inicializa `BufferedReader` debe ser un recurso que se cierre automáticamente.
  - Agregue una cláusula `catch` para una excepción `FileNotFoundException`. En la cláusula `catch`:
    - Imprima `"File not found:" + args[0]`
    - Salga de la aplicación.
  - Agregue una cláusula `catch` para una excepción `IOException`. En la cláusula `catch`:
    - Imprima `"Error reading file:"` junto con el mensaje disponible en el objeto `IOException`.
    - Salga de la aplicación.
13. Ejecute el proyecto. Debe aparecer el contenido del archivo `D:\labs\resources\DeclarationOfIndependence.txt` en la ventana de salida.

## Práctica 9-1: Nivel detallado: Captura de excepciones

---

### Visión general

En esta práctica, creará un nuevo proyecto y capturará excepciones comprobadas y no comprobadas.

### Supuestos

Ha revisado la sección sobre manejo de excepciones de esta lección.

### Resumen

Crearé un proyecto que lee datos de un archivo. Se le proporcionará el código de lectura de archivos. Su tarea es agregar el código de manejo de excepciones adecuado.

### Tareas

1. Cree un nuevo proyecto `ExceptionPractice` como proyecto principal.
  - a. Seleccione `File > New Project`.
  - b. Seleccione `Java` en `Categories` y `Java Application` en `Projects`. Haga clic en el botón `Next`.
  - c. Introduzca la siguiente información en el cuadro de diálogo "Name and Location":
    - Project Name: `ExceptionPractice`
    - Project Location: `D:\labs\09-Exceptions\practices`.
    - (activada) Create Main Class: `com.example.ExceptionMain`
    - (activada) Set as Main Project
  - d. Haga clic en el botón `Finish`.

2. Agregue la siguiente línea al método `main`.

```
System.out.println("Reading from file:" + args[0]);
```

**Nota:** se utilizará un argumento de línea de comandos para especificar el archivo que se leerá. Actualmente no se proporcionan argumentos; no corrija aún este descuido.

3. Ejecute el proyecto. Debe aparecer un mensaje de error similar a:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
 at com.example.ExceptionMain.main(ExceptionMain.java:7)
Java Result: 1
```

4. Rodee la línea `println` del código agregado con una sentencia `try-catch`.

- La cláusula `catch` debe:
  - Aceptar un parámetro del tipo `ArrayIndexOutOfBoundsException`.
  - Imprimir el mensaje: "No file specified, quitting!".
  - Salir de la aplicación con un estado de salida 1 mediante el método `System.exit(1)`.

```
try {
 System.out.println("Reading from file:" + args[0]);
} catch (ArrayIndexOutOfBoundsException e) {
 System.out.println("No file specified, quitting!");
 System.exit(1);
}
```

**Nota:** como el compilador no le ha forzado a manejar o declarar la excepción `ArrayIndexOutOfBoundsException`, se trata de una excepción no comprobada. Normalmente, no es necesario que use un bloque `try-catch` para tratar una excepción no comprobada. Comprobar la longitud de la matriz `args` es otro método con el que se puede garantizar que se ha proporcionado un argumento de línea de comandos.

5. Ejecute el proyecto. Debe aparecer un mensaje de error similar a:

```
No file specified, quitting!
Java Result: 1
```

6. Agregue un argumento de línea de comandos al proyecto.

- Haga clic con el botón derecho en el proyecto `ExceptionPractice` y haga clic en `Properties`.
- En el cuadro de diálogo `Project Properties`, seleccione la categoría `Run`.
- En el campo `Arguments`, introduzca un valor:  
`D:\labs\resources\DeclarationOfIndependence.txt`
- Haga clic en el botón `OK`.

7. Ejecute el proyecto. Debe ver un mensaje similar a:

```
Reading from
file:D:\labs\resources\DeclarationOfIndependence.txt
```

**Advertencia:** ejecutar el proyecto no es lo mismo que ejecutar el archivo. El argumento de la línea de comandos solo se transferirá al método `main` si ejecuta el proyecto.

8. Agregue las siguientes líneas de código al método `main` bajo las líneas que ha agregado previamente:

```
BufferedReader b =
 new BufferedReader(new FileReader(args[0]));
String s = null;
while((s = b.readLine()) != null) {
 System.out.println(s);
}
```

9. Para ejecutar el asistente de Fix Imports, haga clic con el botón derecho en la ventana de código fuente.
10. Ahora deben aparecer errores del compilador en algunas de las líneas que acaba de agregar. Estas líneas podrían generar excepciones comprobadas. Al crear manualmente el proyecto o mantener el cursor sobre la línea con errores, debe aparecer un mensaje similar a este:

```
unreported exception FileNotFoundException; must be caught or
declared to be thrown
```

11. Modifique las propiedades del proyecto para permitir la sentencia `try-with-resources`.
- Haga clic con el botón derecho en el proyecto `ExceptionPractice` y seleccione `Properties`.
  - En el cuadro de diálogo `Project Properties`, seleccione la categoría `Sources`.
  - En la lista desplegable `Source/Binary Format`, seleccione `JDK 7`.
  - Haga clic en el botón `OK`.
12. Rodee el código de E/S del archivo que se proporciona en el paso 8 con una sentencia `try-with-resources`.
- La línea con la que se crea e inicializa `BufferedReader` debe ser un recurso que se cierre automáticamente.
  - Agregue una cláusula `catch` para una excepción `FileNotFoundException`. En la cláusula `catch`:
    - Imprima `"File not found:" + args[0]`
    - Salga de la aplicación.
  - Agregue una cláusula `catch` para una excepción `IOException`. En la cláusula `catch`:
    - Imprima `"Error reading file:"` junto con el mensaje disponible en el objeto `IOException`.
    - Salga de la aplicación.

```
try (BufferedReader b =
 new BufferedReader(new FileReader(args[0]));) {
 String s = null;
 while((s = b.readLine()) != null) {
 System.out.println(s);
 }
} catch (FileNotFoundException e) {
 System.out.println("File not found:" + args[0]);
}
```

```
 System.exit(1);
 } catch(IOException e) {
 System.out.println("Error reading file:" + e.getMessage());
 System.exit(1);
 }
}
```

13. Ejecute el proyecto. Debe aparecer el contenido del archivo

D:\labs\resources\DeclarationOfIndependence.txt en la ventana de salida.

## Práctica 9-2: Nivel de resumen: Ampliación de `Exception`

### Visión general

En esta práctica, utilizará una aplicación existente y refactorizará el código para utilizar una clase de excepción personalizada y un recurso que permite cierre automático personalizado.

### Supuestos

Ha revisado la sección sobre manejo de excepciones de esta lección.

### Resumen

Se le ha asignado un proyecto que implanta la lógica de una aplicación de recursos humanos. La aplicación permite crear, recuperar, actualizar, suprimir y mostrar objetos de tipo `Employee`. Es el mismo proyecto que ha realizado en la práctica “Aplicación del patrón DAO”.

Actualmente, las únicas excepciones que genera la implantación de DAO (`EmployeeDAOMemoryImpl`) son excepciones no comprobadas, como `ArrayIndexOutOfBoundsException`.

En las futuras implantaciones de DAO no es necesario volver a escribir la lógica de aplicación (`EmployeeTestInteractive`). Sin embargo, algunas implantaciones de DAO generarán excepciones comprobadas que se deben tratar. Al crear una clase de excepción comprobada personalizada que se usará para encapsular cualquier excepción generada por DAO, puede que parezca que todas las implantaciones de DAO generan el mismo tipo de excepción. De esta forma, ya no será necesario cambiar ninguna lógica de aplicación al crear implantaciones de DAO de bases de datos en posteriores prácticas.

### Tareas

1. Abra el proyecto `DAOException` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\09-Exceptions\practices`.
  - c. Seleccione `DAOException` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.

|          |        |          |          |        |         |
|----------|--------|----------|----------|--------|---------|
| [C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit: |
|----------|--------|----------|----------|--------|---------|

4. Cree una clase `DAOException` en el paquete `com.example.dao`.
5. Termine la clase `DAOException`. La clase `DAOException` debe:
  - Ampliar la clase `Exception`.
  - Contener cuatro constructores con parámetros que coincidan con esos cuatro constructores presentes en la clase `Exception`. Para cada constructor, utilice `super()` para llamar al constructor de la clase principal con parámetros coincidentes.
6. Modifique la interfaz de `EmployeeDAO`.
  - Todos los métodos deben declarar que se puede detectar una excepción `DAOException` durante la ejecución.



- Amplíe la interfaz de `AutoCloseable`.

7. Modifique el método `add` en la clase `EmployeeDAOMemoryImpl` para:

- Declarar que se puede producir una excepción `DAOException` durante la ejecución de este método.
- Utilizar una sentencia `if` para confirmar que con la adición no se sobrescribirá un empleado existente. Si este es el caso, genere una excepción `DAOException` y entréguela al emisor de la llamada al método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.
- Utilice un bloque `try-catch` para capturar la excepción no comprobada `ArrayIndexOutOfBoundsException` que se podría generar.
- En el bloque `catch` que acaba de crear, genere una excepción `DAOException` y envíela al emisor de la llamada del método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.

**Nota:** la comprobación de la longitud de `employeeArray` se podría utilizar para determinar si se debe detectar la excepción `DAOException`. Sin embargo, el uso de un bloque `try-catch` será típico de la estructura usada al crear un DAO de base de datos.

8. Modifique el método `update` en la clase `EmployeeDAOMemoryImpl` para:

- Declarar que se puede producir una excepción `DAOException` durante la ejecución de este método.
- Utilice una sentencia `if` para validar que se está actualizando un empleado existente. Si no se está haciendo, genere una excepción `DAOException` y envíela al emisor de la llamada al método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.
- Utilice un bloque `try-catch` para capturar la excepción no comprobada `ArrayIndexOutOfBoundsException` que se podría generar.
- En el bloque `catch` que acaba de crear, genere una excepción `DAOException` y envíela al emisor de la llamada del método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.

9. Modifique el método `delete` en la clase `EmployeeDAOMemoryImpl` para:

- Declarar que se puede producir una excepción `DAOException` durante la ejecución de este método.
- Utilice una sentencia `if` para validar que se está suprimiendo un empleado existente. Si no se está haciendo, genere una excepción `DAOException` y envíela al emisor de la llamada al método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.
- Utilice un bloque `try-catch` para capturar la excepción no comprobada `ArrayIndexOutOfBoundsException` que se podría generar.
- En el bloque `catch` que acaba de crear, genere una excepción `DAOException` y envíela al emisor de la llamada del método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.

10. Modifique el método `findByld` en la clase `EmployeeDAOMemoryImpl` para:

- Declarar que se puede producir una excepción `DAOException` durante la ejecución de este método.
- Utilice un bloque `try-catch` para capturar la excepción no comprobada `ArrayIndexOutOfBoundsException` que se podría generar.
- En el bloque `catch` que acaba de crear, genere una excepción `DAOException` y envíela al emisor de la llamada del método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.

11. Agregue un método `close` en la clase `EmployeeDAOMemoryImpl` para implantar la interfaz de `AutoCloseable`.

```
@Override
public void close() {
 System.out.println("No database connection to close just yet");
}
```

**Nota:** la clase `EmployeeDAOMemoryImpl` implanta `EmployeeDAO`, que amplía `AutoCloseable` y, por tanto, la clase `EmployeeDAOMemoryImpl` debe proporcionar un método `close`.

12. Modifique la clase `EmployeeTestInteractive` para manejar los objetos `DAOException` que devuelve `EmployeeDAO`.

- Importe la clase `com.example.dao.DAOException`.
- Modifique el método `executeMenu` para declarar que devuelve una excepción adicional del tipo `DAOException`.
- Elimine la sentencia `throws` del método `main`.

```
public static void main(String[] args) throws Exception
```

d. Modifique el método `main` para utilizar una sentencia `try-with-resources`.

- Rodee el bucle `do-while` con un bloque `try`.
- Convierta las referencias `EmployeeDAO` y `BufferedReader` en recursos de cierre automático.
- Agregue una cláusula `catch` para una excepción `IOException` al final del bloque `try` para manejar ambos errores de E/S que devuelve el método `executeMenu` y cuando se cierre automáticamente `BufferedReader`.

```
catch (IOException e) {
 System.out.println("Error " + e.getClass().getName() +
 " , quitting.");
 System.out.println("Message: " + e.getMessage());
}
```

- Agregue una segunda cláusula catch para un elemento `Exception` al final del bloque try para manejar los errores cuando cierre automáticamente `EmployeeDAO`.

```
catch(Exception e) {
 System.out.println("Error closing resource " +
e.getClass().getName());
 System.out.println("Message: " + e.getMessage());
}
```

**Nota:** en este momento, la aplicación se compilará y ejecutará, pero las instancias `DAOException` generadas harán que la aplicación finalice. Por ejemplo, si crea un empleado con un ID 100, la aplicación saltará fuera del bucle `do-while` y se transferirá a esta cláusula catch.

- Agregue un bloque try-catch anidado en el método `main` que maneje excepciones del tipo `DAOException` que pueda devolver el método `executeMenu`.

```
try {
 timeToQuit = executeMenu(in, dao);
} catch (DAOException e) {
 System.out.println("Error " + e.getClass().getName());
 System.out.println("Message: " + e.getMessage());
}
```

- Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.

```
[C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit:
```

Intente suprimir un empleado que no exista. Debe ver un mensaje similar a:

```
Error com.example.dao.DAOException
Message: Error deleting employee in DAO, no such employee 1
```

## Práctica 9-2: Nivel detallado: Ampliación de `Exception`

### Visión general

En esta práctica, utilizará una aplicación existente y refactorizará el código para utilizar una clase de excepción personalizada y un recurso que permite cierre automático personalizado.

### Supuestos

Ha revisado la sección sobre manejo de excepciones de esta lección.

### Resumen

Se le ha asignado un proyecto que implanta la lógica de una aplicación de recursos humanos. La aplicación permite crear, recuperar, actualizar, suprimir y mostrar objetos de tipo `Employee`. Es el mismo proyecto que ha realizado en la práctica “Aplicación del patrón DAO”.

Actualmente, las únicas excepciones que genera la implantación de DAO (`EmployeeDAOMemoryImpl`) son excepciones no comprobadas, como `ArrayIndexOutOfBoundsException`.

En las futuras implantaciones de DAO no es necesario volver a escribir la lógica de aplicación (`EmployeeTestInteractive`). Sin embargo, algunas implantaciones de DAO generarán excepciones comprobadas que se deben tratar. Al crear una clase de excepción comprobada personalizada que se usará para encapsular cualquier excepción generada por DAO, puede que parezca que todas las implantaciones de DAO generan el mismo tipo de excepción. De esta forma, ya no será necesario cambiar ninguna lógica de aplicación al crear implantaciones de DAO de bases de datos en posteriores prácticas.

### Tareas

1. Abra el proyecto `DAOException` como proyecto principal.
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\09-Exceptions\practices`.
  - c. Seleccione `DAOException` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.

`[C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit:`

4. Cree una clase `DAOException` en el paquete `com.example.dao`.
5. Termine la clase `DAOException`. La clase `DAOException` debe:
  - Ampliar la clase `Exception`.
  - Contener cuatro constructores con parámetros que coincidan con esos cuatro constructores presentes en la clase `Exception`. Para cada constructor, utilice `super()` para llamar al constructor de la clase principal con parámetros coincidentes.

```
public class DAOException extends Exception {

 public DAOException() {
 super();
 }
}
```

```
}

public DAOException(String message) {
 super(message);
}

public DAOException(Throwable cause) {
 super(cause);
}

public DAOException(String message, Throwable cause) {
 super(message, cause);
}
}
```

6. Modifique todos los métodos en la interfaz de `EmployeeDAO`.

- Todos los métodos deben declarar que se puede detectar una excepción `DAOException` durante la ejecución.
- Amplíe la interfaz de `AutoCloseable`.

```
public interface EmployeeDAO extends AutoCloseable {

 public void add(Employee emp) throws DAOException;

 public void update(Employee emp) throws DAOException;

 public void delete(int id) throws DAOException;

 public Employee findById(int id) throws DAOException;

 public Employee[] getAllEmployees() throws DAOException;

}
```

7. Modifique el método `add` en la clase `EmployeeDAOMemoryImpl` para:

- Declarar que se puede producir una excepción `DAOException` durante la ejecución de este método.
- Utilizar una sentencia `if` para confirmar que con la adición no se sobrescribirá un empleado existente. Si este es el caso, genere una excepción `DAOException` y entréguela al emisor de la llamada al método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.
- Utilice un bloque `try-catch` para capturar la excepción no comprobada `ArrayIndexOutOfBoundsException` que se podría generar.
- En el bloque `catch` que acaba de crear, genere una excepción `DAOException` y envíela al emisor de la llamada del método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.

```
public void add(Employee emp) throws DAOException {
 if(employeeArray[emp.getId()] != null) {
 throw new DAOException("Error adding employee in DAO,
employee id already exists " + emp.getId());
 }
 try {
 employeeArray[emp.getId()] = emp;
 } catch (ArrayIndexOutOfBoundsException e) {
 throw new DAOException("Error adding employee in DAO, id
must be less than " + employeeArray.length);
 }
}
```

**Nota:** comprobar la longitud de `employeeArray` se podría usar para determinar si se debe devolver `DAOException`. Sin embargo, el uso de un bloque `try-catch` será típico de la estructura usada al crear un DAO de base de datos.

8. Modifique el método `update` en la clase `EmployeeDAOMemoryImpl` para:

- Declarar que se puede producir una excepción `DAOException` durante la ejecución de este método.
- Utilizar una sentencia `if` para validar que se está actualizando un empleado existente. Si no se está haciendo, genere una excepción `DAOException` y envíela al emisor de la llamada al método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.
- Utilizar un bloque `try-catch` para capturar la excepción no comprobada `ArrayIndexOutOfBoundsException` que se podría generar.

- En el bloque catch que acaba de crear, genere una excepción `DAOException` y envíela al emisor de la llamada del método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.

```
public void update(Employee emp) throws DAOException {
 if(employeeArray[emp.getId()] == null) {
 throw new DAOException("Error updating employee in DAO,
no such employee " + emp.getId());
 }
 try {
 employeeArray[emp.getId()] = emp;
 } catch (ArrayIndexOutOfBoundsException e) {
 throw new DAOException("Error updating employee in DAO,
id must be less than " + employeeArray.length);
 }
}
```

9. Modifique el método `delete` en la clase `EmployeeDAOMemoryImpl` para:

- Declarar que se puede producir una excepción `DAOException` durante la ejecución de este método.
- Utilizar una sentencia `if` para validar que se está suprimiendo un empleado existente. Si no se está haciendo, genere una excepción `DAOException` y envíela al emisor de la llamada al método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.
- Utilizar un bloque `try-catch` para capturar la excepción no comprobada `ArrayIndexOutOfBoundsException` que se podría generar.
- En el bloque `catch` que acaba de crear, genere una excepción `DAOException` y envíela al emisor de la llamada del método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.

```
public void delete(int id) throws DAOException {
 if(employeeArray[id] == null) {
 throw new DAOException("Error deleting employee in DAO,
no such employee " + id);
 }
 try {
 employeeArray[id] = null;
 } catch (ArrayIndexOutOfBoundsException e) {
 throw new DAOException("Error deleting employee in DAO,
id must be less than " + employeeArray.length);
 }
}
```

10. Modifique el método `findById` en la clase `EmployeeDAOMemoryImpl` para:

- Declarar que se puede producir una excepción `DAOException` durante la ejecución de este método.
- Utilizar un bloque `try-catch` para capturar la excepción no comprobada `ArrayIndexOutOfBoundsException` que se podría generar.
- En el bloque `catch` que acaba de crear, genere una excepción `DAOException` y envíela al emisor de la llamada del método. La excepción `DAOException` debe contener un argumento `String` de mensaje que indique lo que no se ha realizado correctamente y el motivo.

```
public Employee findById(int id) throws DAOException {
 try {
 return employeeArray[id];
 } catch (ArrayIndexOutOfBoundsException e) {
 throw new DAOException("Error finding employee in DAO",
e);
 }
}
```

11. Agregue un método `close` en la clase `EmployeeDAOMemoryImpl` para implantar la interfaz de `AutoCloseable`.

```
@Override
public void close() {
 System.out.println("No database connection to close just
yet");
}
```

**Nota:** la clase `EmployeeDAOMemoryImpl` implanta `EmployeeDAO`, que amplía `AutoCloseable` y, por tanto, la clase `EmployeeDAOMemoryImpl` debe proporcionar un método `close`.

12. Modifique la clase `EmployeeTestInteractive` para manejar los objetos `DAOException` que devuelve `EmployeeDAO`.

a. Importe la clase `com.example.dao.DAOException`.

```
import com.example.dao.DAOException;
```

b. Modifique el método `executeMenu` para declarar que devuelve una excepción adicional del tipo `DAOException`.

```
public static boolean executeMenu(BufferedReader in, EmployeeDAO
dao) throws IOException, DAOException {
```

c. Elimine la sentencia `throws` del método `main`.

```
public static void main(String[] args) throws Exception
```

d. Modifique el método `main` para utilizar una sentencia `try-with-resources`.

- Rodee el bucle `do-while` con un bloque `try`.
- Convierta las referencias `EmployeeDAO` y `BufferedReader` en recursos de cierre automático.



- Agregue una cláusula `catch` para una excepción `IOException` al final del bloque `try` para manejar ambos errores de E/S que devuelve el método `executeMenu` y cuando se cierre automáticamente `BufferedReader`.
- Agregue una segunda cláusula `catch` para un elemento `Exception` al final del bloque `try` para manejar los errores cuando cierre automáticamente `EmployeeDAO`.

```
try (EmployeeDAO dao = factory.createEmployeeDAO();
 BufferedReader in =
new BufferedReader(new InputStreamReader(System.in))) {
 do {
 timeToQuit = executeMenu(in, dao);
 } while (!timeToQuit);
} catch (IOException e) {
 System.out.println("Error " + e.getClass().getName() +
 " , quitting.");
 System.out.println("Message: " + e.getMessage());
} catch (Exception e) {
 System.out.println("Error closing resource " +
 e.getClass().getName());
 System.out.println("Message: " + e.getMessage());
}
```

**Nota:** en este momento, la aplicación se compilará y ejecutará, pero las instancias `DAOException` generadas harán que la aplicación finalice. Por ejemplo, si crea un empleado con un ID 100, la aplicación saltará fuera del bucle `do-while` y se transferirá a esta cláusula `catch`.

- e. Agregue un bloque `try-catch` anidado en el método `main` que maneje excepciones del tipo `DAOException` que pueda devolver el método `executeMenu`.

```
try {
 timeToQuit = executeMenu(in, dao);
} catch (DAOException e) {
 System.out.println("Error " + e.getClass().getName());
 System.out.println("Message: " + e.getMessage());
}
```

13. Ejecute el proyecto. Debe aparecer un menú. Pruebe todas las opciones de menú.

```
[C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit:
```

Intente suprimir un empleado que no exista. Debe ver un mensaje similar a:

```
Error com.example.dao.DAOException
Message: Error deleting employee in DAO, no such employee 1
```

Carlos Jaramillo (cajaramillo@gmail.com) has a non-transferable  
license to use this Student Guide.

## **Prácticas de la lección 10: Conceptos fundamentales de E/S en Java**

### **Capítulo 10**

## Prácticas de la lección 10: Visión general

---

### Visión general de las prácticas

En estas prácticas, utilizará algunas de las clases `java.io` para leer de la consola, abrirá y leerá archivos, además de serializar y anular la serialización de objetos en el sistema de archivos y desde este.

## Práctica 10-1: Nivel de resumen: Escritura de una aplicación simple de E/S de la consola

---

### Visión general

En esta práctica, escribirá una aplicación simple basada en consola que lea del sistema de la consola y escriba en él. En NetBeans, la consola se abre como una ventana en el IDE.

### Tareas

1. Abra el proyecto `FileScanner` en el siguiente directorio:  
`D:\labs\10-IO_Fundamentals\practices\`
2. Abra el archivo `FileScanInteractive`.  
Tenga en cuenta que la clase tiene un método denominado `countTokens` que ya está escrito. Este método toma un elemento `file` de `String` y un elemento `search` de `String` como parámetros. Con este método se abrirá el nombre de archivo transferido y se utilizará una instancia de un elemento `Scanner` para buscar el token de búsqueda. Para cada token encontrado, el método aumenta el campo de entero `instanceCount`. Cuando el archivo finaliza, devuelve el valor de `instanceCount`. Tenga en cuenta que la clase vuelve a emitir cualquier excepción `IOException` que detecte, por lo que tendrá que usar este método dentro de un bloque `try-catch`.
3. Codifique el método `main` para comprobar el número de argumentos transferidos. La aplicación espera al menos un argumento (una cadena que represente el archivo que se va a abrir). Si el número de argumentos es inferior a uno, salga de la aplicación con un código de error (-1).
  - a. Al método `main` se le transfiere una matriz de elementos `String`. Utilice el atributo `length` para determinar si la matriz contiene menos de un argumento.
  - b. Imprima un mensaje si hay menos de un argumento y utilice `System.exit` para devolver un código de error (-1 se suele usar para indicar un error).
4. Guarde el primer argumento transferido a la aplicación como `String`.
5. Cree una instancia de la clase `FileScanInteractive`. La necesitará para llamar al método `countTokens`.
6. Abra la consola del sistema para introducir datos con un lector en buffer.
  - a. Utilice `try-with-resources` para abrir un elemento `BufferedReader` encadenado a la entrada de la consola del sistema (recuerde que `System.in` es un flujo de entrada conectado a la consola del sistema).
  - b. No olvide agregar una sentencia `catch` al bloque `try`. Cualquier excepción que se devuelva será de tipo `IOException`.
  - c. En un bucle `while`, lea de la consola del sistema en una cadena hasta que se introduzca la cadena "q" en la consola por sí misma.  
**Nota:** puede utilizar `equalsIgnoreCase` para permitir a los usuarios introducir una "Q" en mayúsculas o en minúsculas. Asimismo, el método `trim()` es una buena idea para eliminar los caracteres de espacios en blanco que haya en la entrada.
  - d. Si la cadena que se ha leído de la consola no es el carácter de terminación, llame al método `countTokens`, transfiriendo el nombre de archivo y la cadena de búsqueda.

- e. Imprima una cadena que indique cuántas veces ha aparecido el token de búsqueda en el archivo.
  - f. Agregue cualquier sentencia de importación que falte.
7. Guarde la clase `FileScanInteractive`.
8. Si no tiene errores de compilación, puede probar la aplicación con un archivo del directorio de recursos.
- a. Haga clic con el botón derecho en el proyecto y seleccione Properties.
  - b. Haga clic en Run.
  - c. Introduzca el nombre de un archivo para abrirlo en el cuadro de texto Arguments (por ejemplo, `D:\labs\resources\DeclarationOfIndependence.txt`).
  - d. Haga clic en OK.
  - e. Ejecute la aplicación e intente buscar algunas palabras como `when`, `rights` y `free`. La salida debe ser parecida a la siguiente:

```
Searching through the file:
D:\labs\resources\DeclarationOfIndependence.txt
Enter the search string or q to exit: when
The word "when" appears 3 times in the file.
Enter the search string or q to exit: rights
The word "rights" appears 3 times in the file.
Enter the search string or q to exit: free
The word "free" appears 4 times in the file.
Enter the search string or q to exit: q
BUILD SUCCESSFUL (total time: 16 seconds)
```

## Práctica 10-1: Nivel detallado: Escritura de una aplicación simple de E/S de la consola

---

### Visión general

En esta práctica, escribirá una aplicación simple basada en consola que lea del sistema de la consola y escriba en él. En NetBeans, la consola se abre como una ventana en el IDE.

### Tareas

1. Abra el proyecto `FileScanner` en el siguiente directorio:  
`D:\labs\10-IO_Fundamentals\practices\`
  - a. Seleccione `File > Open Project`.
  - b. Vaya a `D:\labs\10-IO_Fundamentals\practices`.
  - c. Seleccione `FileScanner` y active la casilla de control "Open as Main Project".
  - d. Haga clic en el botón `Open Project`.
2. Abra el archivo `FileScanInteractive`.

Tenga en cuenta que la clase tiene un método denominado `countTokens` que ya está escrito. Este método toma un elemento `file` de `String` y un elemento `search` de `String` como parámetros. Con este método se abrirá el nombre de archivo transferido y se utilizará una instancia de un elemento `Scanner` para buscar el token de búsqueda. Para cada token encontrado, el método aumenta el campo de entero `instanceCount`. Cuando el archivo finaliza, devuelve el valor de `instanceCount`. Tenga en cuenta que la clase vuelve a emitir cualquier excepción `IOException` que detecte, por lo que tendrá que usar este método dentro de un bloque `try-catch`.
3. Codifique el método `main` para comprobar el número de argumentos transferidos. La aplicación espera al menos un argumento (una cadena que represente el archivo que se va a abrir). Si el número de argumentos es inferior a uno, salga de la aplicación con un código de error (-1).
  - a. Al método `main` se le transfiere una matriz de elementos `String`. Utilice el atributo `length` para determinar si la matriz contiene menos de un argumento.
  - b. Imprima un mensaje si hay menos de un argumento y utilice `System.exit` para devolver un código de error (-1 se suele usar para indicar un error). Por ejemplo:

```
if (args.length < 1) {
 System.out.println("Usage: java FileScanInteractive <file to
search>");
 System.exit(-1);
}
```

4. Guarde el primer argumento transferido a la aplicación como `String`.  
`String file = args[0];`
5. Cree una instancia de la clase `FileScanInteractive`. La necesitará para llamar al método `countTokens`.

```
FileScanInteractive scan = new FileScanInteractive ();
```

6. Abra la consola del sistema para introducir datos con un lector en buffer.
  - a. Utilice `try-with-resources` para abrir un elemento `BufferedReader` encadenado a la entrada de la consola del sistema (recuerde que `System.in` es un flujo de entrada conectado a la consola del sistema).

- b. No olvide agregar una sentencia `catch` al bloque `try`. Cualquier excepción que se devuelva será de tipo `IOException`. Por ejemplo:

```
try (BufferedReader in =
 new BufferedReader(new InputStreamReader(System.in))) {

} catch (IOException e) { // Catch any IO exceptions.
 System.out.println("Exception: " + e);
 System.exit(-1);
}
```

- c. En el bloque `try` que ha creado, agregue un bucle `while`. Este bucle se debe ejecutar hasta que se detecte una sentencia `break`. Dentro de un bucle `while`, lea datos de la consola del sistema en una cadena hasta que se introduzca la cadena "q" en la consola por sí misma.  
**Nota:** puede usar `equalsIgnoreCase` para permitir que los usuarios introduzcan una "Q" mayúscula o minúscula. Asimismo, el método `trim()` es una buena opción para eliminar cualquier carácter de espacio en blanco de la entrada.
  - d. Si la cadena que se ha leído de la consola no es el carácter de terminación, llame al método `countTokens`, transfiriendo el nombre de archivo y la cadena de búsqueda.
  - e. Imprima una cadena que indique cuántas veces ha aparecido el token de búsqueda en el archivo.
  - f. El aspecto del código dentro del bloque `try` debe ser parecido a este:

```
String search = "";
System.out.println ("Searching through the file: " + file);
while (true) {
 System.out.print("Enter the search string or q to exit: ");
 search = in.readLine().trim();
 if (search.equalsIgnoreCase("q")) break;
 int count = scan.countTokens(file, search);
 System.out.println("The word \"" + search + "\" appears "
 + count + " times in the file.");
}
```

- g. Agregue cualquier sentencia de importación que falte.
7. Guarde la clase `FileScanInteractive`.



8. Si no tiene errores de compilación, puede probar la aplicación con un archivo del directorio de recursos.
  - a. Haga clic con el botón derecho en el proyecto y seleccione Properties.
  - b. Haga clic en Run.
  - c. Introduzca el nombre de un archivo para abrirlo en el cuadro de texto Arguments (por ejemplo, D:\labs\resources\DeclarationOfIndependence.txt).
  - d. Haga clic en OK.
  - e. Ejecute la aplicación e intente buscar algunas palabras como `when`, `rights` y `free`. La salida debe ser parecida a la siguiente:

```
Searching through the file:
D:\labs\resources\DeclarationOfIndependence.txt
Enter the search string or q to exit: when
The word "when" appears 3 times in the file.
Enter the search string or q to exit: rights
The word "rights" appears 3 times in the file.
Enter the search string or q to exit: free
The word "free" appears 4 times in the file.
Enter the search string or q to exit: q
BUILD SUCCESSFUL (total time: 16 seconds)
```

## Práctica 10-2: Nivel de resumen: Serialización y anulación de la serialización de un objeto ShoppingCart

---

### Visión general

En esta práctica, utilizará la clase `java.io.ObjectOutputStream` para escribir un objeto Java en el sistema de archivos (serializar) y, a continuación, utilizará el mismo flujo para volver a leer el archivo en una referencia de objeto. También personalizará la serialización y anulación de serialización del objeto `ShoppingCart`.

### Tareas

1. Abra el proyecto `SerializeShoppingCart` del directorio `D:\labs\10-IO_Fundamentals\practices`.
2. Amplíe el paquete `com.example.test`. Observe que hay dos clases principales Java en este paquete, `SerializeTest` y `DeserializeTest`. Escribirá el código en estas clases principales para serializar y anular la serialización de objetos `ShoppingCart`.
3. Abra la clase `SerializeTest`. Escribirá los métodos en esta clase para escribir varios objetos `ShoppingCart` en el sistema de archivos.
  - a. Lea el código. Observará que la clase solicita el ID de carro y creará una instancia de `ShoppingCart` con el ID de carro en el constructor.
  - b. A continuación, el código agrega tres objetos `Item` a `ShoppingCart`.
  - c. Tras esto, el código imprime el número de artículos del carro y el coste total de los elementos del mismo. Revise las clases `ShoppingCart` y `Item` del paquete `com.example.domain` para obtener información sobre su funcionamiento.
  - d. Escribirá el código para abrir un elemento `ObjectOutputStream` y `ShoppingCart` como objeto serializado en el sistema de archivos.
4. Cree el bloque `try` para abrir un elemento `FileOutputStream` encadenado a un elemento `ObjectOutputStream`. El nombre de archivo ya se ha creado.
  - a. Su código se insertará en la ubicación de la línea de comentario al final del archivo.
  - b. Abra un elemento `FileOutputStream` con la cadena `cartFile` en un bloque `try-with-resources`.
  - c. Transfiera la instancia del flujo de salida del archivo a una clase `ObjectOutputStream` para escribir la instancia de objeto serializado en el archivo.
  - d. Escriba el objeto `cart` en la instancia de flujo de salida del objeto mediante el método `writeObject`.
  - e. Asegúrese de capturar cualquier `IOException` y de salir con un error si es necesario.
  - f. Agregue un mensaje que indique que se ha realizado correctamente antes de que finalice el método:

```
System.out.println ("Successfully serialized shopping cart with ID: " + cart.getCartId());
```
  - g. Guarde el archivo.
5. Abra la clase `DeserializeTest`. El método `main` de esta clase se lee de la consola para el ID del carro de compra del cliente cuya serialización se va a anular.

6. Su código se insertará en la ubicación de la línea de comentario al final del archivo.
  - a. Abra un elemento `FileOutputStream` con la cadena `cartFile` en un bloque `try-with-resources`.
  - b. Transfiera la instancia del flujo de salida del archivo a una clase `ObjectOutputStream` para escribir la instancia de objeto serializado en el archivo.
  - c. Lea el objeto `cart` del flujo de salida del objeto mediante el método `readObject`. Asegúrese de convertir el resultado en el tipo de objeto adecuado.
  - d. Tendrá que capturar tanto `ClassNotFoundException` como `IOException`, por lo que debe usar una expresión multi-catch.
  - e. Por último, imprima los resultados de `cart` (todo su contenido) y el coste total de `cart` mediante el siguiente código:

```
System.out.println ("Shopping Cart contains: ");
List<Item> cartContents = cart.getItems();
for (Item item : cartContents) {
 System.out.println (item);
}
System.out.println ("Shopping cart total: " +
 NumberFormat.getCurrencyInstance().format(cart.getCartTotal()));
```

- f. Guarde el archivo.
7. Abra la clase `ShoppingCart`. Personalizará la serialización y anulación de serialización de esta clase agregando los dos métodos llamados durante la serialización/anulación de serialización.
  - a. Agregue un método `writeObject` llamado durante la serialización. Este método debe serializar los campos del objeto actual para, a continuación, agregar un registro de hora (instancia de objeto `Date`) al final del flujo de objetos.
8. Agregue un método a la clase `ShoppingCart` que se llama durante la anulación de la serialización.
  - a. Agregue un método `readObject` con la firma adecuada. Con este método se volverá a calcular el coste total del carro de compra, además de imprimirse el registro de hora que se agregó al flujo.
  - b. Guarde el archivo.
9. Pruebe la aplicación. Esta aplicación tiene dos métodos `main`, por lo que tendrá que ejecutar uno y luego el otro.
  - a. Para ejecutar la clase `SerializeTest`, haga clic con el botón derecho en el nombre de la clase y seleccione `Run File`.
  - b. La salida se debe parecer a la siguiente:

```
Enter the ID of the cart file to create and serialize or q exit.
101
Shopping cart 101 contains 3 items
Shopping cart total: $58.39
Successfully serialized shopping cart with ID: 101
```

- c. Para ejecutar `DeserializeTest`, haga clic con el botón derecho en el nombre de la clase y seleccione `Run File`.
- d. Introduzca el ID 101 para que la salida tenga un aspecto similar a este:

```
Enter the ID of the cart file to deserialize or q exit.
```

```
101
```

```
Restored Shopping Cart from: Oct 26, 2011
```

```
Successfully deserialized shopping cart with ID: 101
```

```
Shopping cart contains:
```

```
Item ID: 101 Description: Duke Plastic Circular Flying Disc
Cost: 10.95
```

```
Item ID: 123 Description: Duke Soccer Pro Soccer ball Cost:
29.95
```

```
Item ID: 45 Description: Duke "The Edge" Tennis Balls - 12-Ball
Bag Cost: 17.49
```

```
Shopping cart total: $58.39
```

## Práctica 10-2: Nivel detallado: Serialización y anulación de la serialización de un objeto `ShoppingCart`

---

### Visión general

En esta práctica, utilizará la clase `java.io.ObjectOutputStream` para escribir un objeto Java en el sistema de archivos (serializar) y, a continuación, utilizará el mismo flujo para volver a leer el archivo en una referencia de objeto. También personalizará la serialización y anulación de serialización del objeto `ShoppingCart`.

### Tareas

1. Abra el proyecto `SerializeShoppingCart` del directorio `D:\labs\10-IO_Fundamentals\practices`.
  - a. Seleccione `File > Open Project`.
  - b. Vaya al directorio `D:\labs\10-IO_Fundamentals\practices`.
  - c. Seleccione el proyecto `SerializeShoppingCart`.
  - d. Haga clic en el botón `Open Project`.
2. Amplíe el paquete `com.example.test`. Observe que hay dos clases principales Java en este paquete, `SerializeTest` y `DeserializeTest`. Escribirá el código en estas clases principales para serializar y anular la serialización de objetos `ShoppingCart`.
3. Abra la clase `SerializeTest`. Escribirá los métodos en esta clase para escribir varios objetos `ShoppingCart` en el sistema de archivos.
  - a. Lea el código. Observará que la clase solicita el ID de carro y creará una instancia de `ShoppingCart` con el ID de carro en el constructor.
  - b. A continuación, el código agrega tres objetos `Item` a `ShoppingCart`.
  - c. Tras esto, el código imprime el número de artículos del carro y el coste total de los elementos del mismo. Revise las clases `ShoppingCart` y `Item` del paquete `com.example.domain` para obtener información sobre su funcionamiento.
  - d. Escribirá el código para abrir un elemento `ObjectOutputStream` y `ShoppingCart` como objeto serializado en el sistema de archivos.
4. Cree el bloque `try` para abrir un elemento `FileOutputStream` encadenado a un elemento `ObjectOutputStream`. El nombre de archivo ya se ha creado.
  - a. Su código se insertará en la ubicación de la línea de comentario al final del archivo.
  - b. Abra un elemento `FileOutputStream` con la cadena `cartFile` en un bloque `try-with-resources`.
  - c. Transfiera la instancia del flujo de salida del archivo a una clase `ObjectOutputStream` para escribir la instancia de objeto serializado en el archivo.
  - d. Escriba el objeto `cart` en la instancia de flujo de salida del objeto mediante el método `writeObject`.
  - e. Asegúrese de capturar cualquier `IOException` y de salir con un error si es necesario.

- f. El código se podría parecer al siguiente:

```
try (FileOutputStream fos = new FileOutputStream (cartFile);
 ObjectOutputStream o = new ObjectOutputStream (fos)) {
 o.writeObject(cart);
} catch (IOException e) {
 System.out.println ("Exception serializing " + cartFile + ":
" + e);
 System.exit(-1);
}
```

- g. Agregue un mensaje que indique que se ha realizado correctamente antes de que finalice el método:
- ```
System.out.println ("Successfully serialized shopping cart with
ID: " + cart.getCartId());
```
- h. Agregue cualquier sentencia de importación que falte.
- i. Guarde el archivo.
5. Abra la clase `DeserializeTest`. El método `main` de esta clase se lee de la consola para el ID del carro de compra del cliente cuya serialización se va a anular.
6. Su código se insertará en la ubicación de la línea de comentario al final del archivo.
- a. Abra un elemento `FileInputStream` con la cadena `cartFile` en un bloque `try-with-resources`.
- b. Transfiera la instancia del flujo de salida del archivo a una clase `ObjectInputStream` para escribir la instancia de objeto serializado en el archivo.
- c. Lea el objeto `cart` del flujo de salida del objeto mediante el método `readObject`. Asegúrese de convertir el resultado en el tipo de objeto adecuado.
- d. Tendrá que capturar tanto `ClassNotFoundException` como `IOException`, por lo que debe usar una expresión multi-catch.
- e. El código se debe parecer al siguiente:

```
try (FileInputStream fis = new FileInputStream (cartFile);
    ObjectInputStream in = new ObjectInputStream (fis)) {
    cart = (ShoppingCart)in.readObject();
} catch (final ClassNotFoundException | IOException e) {
    System.out.println ("Exception deserializing " + cartFile +
": " + e);
    System.exit(-1);
}
System.out.println ("Successfully deserialized shopping cart
with ID: " + cart.getCartId());
```

- f. Por último, imprima los resultados de `cart` (todo su contenido) y el coste total de `cart` mediante el siguiente código:

```
System.out.println ("Shopping cart contains: ");
List<Item> cartContents = cart.getItems();
for (Item item : cartContents) {
    System.out.println (item);
}
System.out.println ("Shopping cart total: " +
    NumberFormat.getCurrencyInstance().format(cart.getCartTotal()));
```

- g. Guarde el archivo.

7. Abra la clase `ShoppingCart`. Personalizará la serialización y anulación de serialización de esta clase agregando los dos métodos llamados durante la serialización/anulación de serialización.

- a. Agregue un método llamado durante la serialización que agregará un registro de hora (instancia de objeto `Date`) al final del flujo de objetos.
- b. Agregue un método con la firma:

```
private void writeObject(ObjectOutputStream oos) throws
IOException {
```

- c. Asegúrese de que el método serializa primero los campos del objeto actual y, a continuación, escriba la instancia del objeto `Date`:

```
    oos.defaultWriteObject();
    oos.writeObject(new Date());
}
```

8. Agregue un método a la clase `ShoppingCart` que se llama durante la anulación de la serialización. Con este método se volverá a calcular el coste total del carro de compra, además de imprimirse el registro de hora que se agregó al flujo.

- a. Agregue un método con la firma:

```
private void readObject(ObjectInputStream ois) throws
IOException, ClassNotFoundException {
```

- b. Este método anulará la serialización de los campos del flujo de objeto y volverá a calcular el valor de dólares total del contenido del carro actual:

```
    ois.defaultReadObject();
    if (cartTotal == 0 && (items.size() > 0)) {
        for (Item item : items)
            cartTotal += item.getCost();
    }
```

- c. Obtenga el objeto `Date` del flujo serializado e imprima el registro de hora en la consola.

```
    Date date = (Date)ois.readObject();
    System.out.println ("Restored Shopping Cart from: " +
        DateFormat.getDateInstance().format(date));
}
```

- d. Guarde el objeto `ShoppingCart`.

9. Pruebe la aplicación. Esta aplicación tiene dos métodos main, por lo que tendrá que ejecutar uno y luego el otro.

- a. Para ejecutar la clase `SerializeTest`, haga clic con el botón derecho en el nombre de la clase y seleccione Run File. Introduzca un ID de carro, como 101.
- b. La salida se debe parecer a la siguiente:

```
Enter the ID of the cart file to create and serialize or q exit.
101
Shopping cart 101 contains 3 items
Shopping cart total: $58.39
Successfully serialized shopping cart with ID: 101
```

- c. Para ejecutar `DeserializeTest`, haga clic con el botón derecho en el nombre de la clase y seleccione Run File.
- d. Introduzca el ID 101 para que la salida tenga un aspecto como este:

```
Enter the ID of the cart file to deserialize or q exit.
101
Restored Shopping Cart from: Oct 26, 2011
Successfully deserialized shopping cart with ID: 101
Shopping cart contains:
Item ID: 101 Description: Duke Plastic Circular Flying Disc
Cost: 10.95
Item ID: 123 Description: Duke Soccer Pro Soccer ball Cost:
29.95
Item ID: 45 Description: Duke "The Edge" Tennis Balls - 12-Ball
Bag Cost: 17.49
Shopping cart total: $58.39
```


Prácticas de la lección 11: E/S de archivos Java (NIO.2)

Capítulo 11

Prácticas de la lección 11: Visión general

Visión general de las prácticas

En la primera práctica, utilizará la API de JDK 7 NIO.2 para escribir una aplicación con el fin de crear cartas personalizadas, mediante la fusión de una plantilla de una carta con una lista de nombres y mediante el uso de los métodos `Files` y `Path`. En la segunda práctica, utilizará el método `walkFileTree` para copiar todos los archivos y directorios de una carpeta en otra del disco. En la práctica opcional final, utilizará el mismo método para escribir una aplicación que busque y suprima de forma recursiva todos los archivos que coincidan con un patrón proporcionado.

Práctica 11-1: Nivel de resumen: Escritura de una aplicación de fusión de archivos

Visión general

En esta práctica, utilizará el método `Files.readAllLines` para leer todo el contenido de dos archivos: una plantilla de carta y una lista de nombres a los que enviar las cartas. Después de crear una carta con un nombre de la lista de nombres, utilizará el método `Files.write` para crear la carta personalizada. También utilizará las clases `Pattern` y `Matcher` que aparecieron en la lección “Procesamiento de cadenas”.

Supuestos

Ha participado en la clase teórica de esta lección. Tenga en cuenta que en el directorio de ejemplos se incluyen proyectos de NetBeans para ayudarle a comprender cómo usar los métodos `readAllLines` y `write` de la clase `Files`.

Tareas

1. Abra el archivo `FormTemplate` en el directorio `resources`.
Tenga en cuenta que se trata de una carta con una cadena `<NAME>` que se sustituirá por un nombre del archivo de la lista de nombres.
2. Abra el archivo `NamesList.txt` del directorio `resources`.
 - a. Este archivo contiene los nombres a los que se van a enviar cartas.
 - b. Agregue el nombre al final de la lista.
 - c. Guarde el archivo.
3. Abra el proyecto `FormLetterWriter` del directorio `practices`.
4. Amplíe la clase `FormLetterWriter`. Observe que esta clase contiene el método `main` y que en la aplicación se necesitan dos parámetros: uno es la ruta de acceso a la plantilla de cartas y el otro es la ruta de acceso al archivo que contiene la lista de nombres que se van a sustituir en la carta.
 - a. Después de comprobar el número válido de argumentos, el método `main` comprueba si el objeto `Path` apunta a archivos válidos.
 - b. El método `main` crea una instancia de la clase `FileMerge` con el objeto `Path` de la carta y la lista de nombres del objeto `Path`.
 - c. En un bloque `try`, el método `main` llama al método `writeMergedForm` de la clase `FileMerge`. Se trata del método que escribirá en esta práctica.
5. Amplíe la clase `FileMerge`.
 - a. Observe que el método `writeMergedForms` está vacío. Se trata del método que escribirá en esta práctica.
 - b. El método `mergeName` utiliza el objeto `Pattern` definido en las declaraciones de campo para sustituir a la cadena de la plantilla de carta (primer argumento) por un nombre de la lista de nombres (segundo argumento). Devuelve `String`. Por ejemplo, sustituye "Dear `<NAME>`," por "Dear Wilson Ball,".
 - c. El método `hasToken` devuelve un valor `boolean` para indicar si la cadena transferida contiene el token. Esto resulta útil para identificar qué cadena tiene el token que se va a sustituir por el nombre de la lista de nombres.

6. Codifique el método `writeMergedForms`. El plan general para este método consiste en leer toda la carta, línea a línea, para, a continuación, leer la lista entera de nombres y fusionarlos con la carta, sustituyendo el marcador de posición de la plantilla por un nombre de la lista y, posteriormente, escribiendo ese contenido como archivo. El resultado neto debería ser que, si la lista incluyera diez nombres, debería tener al final diez archivos de cartas personalizadas destinadas a los nombres de la lista correspondientes. Estos diez archivos se escribirán en el directorio de recursos.
 - a. Lea todas las líneas de la carta en el campo `formLetter` y todas las líneas de la lista de nombres en el campo `nameList`.
Nota: debido a que `writeMergedForms` devuelve `IOException`, no es necesario que inserte estas sentencias en un bloque `try`. El emisor de la llamada de este método es el encargado de manejar las excepciones que se puedan devolver.
 - b. Cree un bucle `for` para iterar con la lista de cadenas de nombres (`nameList`).
 - c. Dentro de este, para el bucle, cree un nuevo objeto `List` para que incluya las cadenas de la carta. Necesita este nuevo objeto `List` para que incluya las cadenas de plantilla de carta modificada que se van a escribir.
 - d. Aún dentro del bucle `for`, tendrá que crear un nombre para la carta personalizada. Una forma sencilla de hacerlo es usar el nombre de la lista de nombres. Debe sustituir los espacios del nombre por caracteres de subrayado para facilitar la lectura del nombre de archivo. Cree un nuevo objeto `Path` relativo a la ruta de acceso de plantilla de carta.
 - e. Cree otro bucle `for`, anidado en el primer bucle, para que se itere con las líneas de la plantilla de carta y busque la cadena de token ("`<NAME>`") para sustituirla por el nombre `String` de `nameList`. Utilice el método `hasToken` para buscar el objeto `String` que contiene la cadena de token y sustituya esa cadena por la que contiene el nombre de `nameList`. Utilice el método `mergeName` para crear el nuevo objeto `String`. Agregue el objeto `String` modificado y el resto de cadenas de `formLetter` al nuevo `customLetter List`.
 - f. Aún dentro del primer bucle `for`, escriba el elemento `List` modificado del objeto `Strings` que representa la carta personalizada en el sistema de archivos mediante el método `Files.write`. Imprima un mensaje que indique que la escritura del archivo se ha realizado correctamente y cierre el bucle `for` exterior.
 - g. Guarde la clase `FileMerge`.
7. Modifique el proyecto `FormLetterWriter` para transferir el archivo de carta y el archivo de lista de nombres al método `main`.
 - a. Haga clic con el botón derecho en el proyecto y seleccione `Properties`.
 - b. Seleccione `Run`.
 - c. En el campo de texto `Arguments`, introduzca:
`D:\labs\resources\FormTemplate.txt D:\labs\resources\NamesList.txt`
 - d. Haga clic en `OK`.

8. Ejecute el proyecto. Deben aparecer los nuevos archivos que se han creado con los nombres de la lista de nombres. Cada archivo se debe personalizar con el nombre de la lista de nombres. Por ejemplo, el archivo `Tom_McGinn.txt` debe contener:

Dear Tom McGinn,

It has come to our attention that you would like to prove your Java Skills. May we recommend that you consider certification from Oracle? Oracle has globally recognized Certification exams that will test your Java knowledge and skills.

Start with the Oracle Certified Java Associate exam, and then continue to the Oracle Certified Java Programmer Professional for a complete certification profile.

Good Luck!

Oracle University

Práctica 11-1: Nivel detallado: Escritura de una aplicación de fusión de archivos

Visión general

En esta práctica, utilizará el método `Files.readAllLines` para leer todo el contenido de dos archivos: una plantilla de carta y una lista de nombres a los que enviar las cartas. Después de crear una carta con un nombre de la lista de nombres, utilizará el método `Files.write` para crear la carta personalizada. También utilizará las clases `Pattern` y `Matcher` que aparecieron en la lección “Procesamiento de cadenas”.

Supuestos

Ha participado en la clase teórica de esta lección. Tenga en cuenta que en el directorio de ejemplos se incluyen proyectos de NetBeans para ayudarle a comprender cómo usar los métodos `readAllLines` y `write` de la clase `Files`.

Tareas

1. Abra el archivo `FormTemplate` en el directorio `resources`.
 - a. Seleccione `File > Open File`.
 - b. Navegue al directorio `resources` de `D:\labs`
 - c. Seleccione el archivo `FormTemplate.txt` y haga clic en el botón `Open`.
Tenga en cuenta que se trata de una carta con un token de marcador de posición de cadena `<NAME>` que se sustituirá por un nombre del archivo de la lista de nombres.
2. Abra el archivo `NamesList.txt` del directorio `resources`.
 - a. Este archivo contiene los nombres a los que se van a enviar cartas.
 - b. Agregue el nombre al final de la lista.
 - c. Guarde el archivo.
3. Abra el proyecto `FormLetterWriter` del directorio `practices`.
 - a. Seleccione `File > Open Project`.
 - b. Vaya a `D:\labs\11-NIO.2\practices`.
 - c. Seleccione `FormLetterWriter`.
 - d. Active la casilla de control “`Open as Main Project`”.
 - e. Haga clic en el botón `Open Project`.
4. Amplíe la clase `FormLetterWriter`. Observe que esta clase contiene el método `main` y que en la aplicación se necesitan dos parámetros: uno es la ruta de acceso a la plantilla de cartas y el otro es la ruta de acceso al archivo que contiene la lista de nombres que se van a sustituir en la carta.
 - a. Después de comprobar el número válido de argumentos, el método `main` comprueba si el objeto `Path` apunta a archivos válidos.
 - b. El método `main` crea una instancia de la clase `FileMerge` con el objeto `Path` de la carta y la lista de nombres del objeto `Path`.
 - c. En un bloque `try`, el método `main` llama al método `writeMergedForm` de la clase `FileMerge`. Se trata del método que escribirá en esta práctica.

5. Amplíe la clase `FileMerge`.
 - a. Observe que el método `writeMergedForms` está vacío. Se trata del método que escribirá en esta práctica.
 - b. El método `mergeName` utiliza el objeto `Pattern` definido en las declaraciones de campo para sustituir a la cadena de la plantilla de carta (primer argumento) por un nombre de la lista de nombres (segundo argumento). Devuelve `String`. Por ejemplo, sustituye "Dear <NAME>," por "Dear Wilson Ball,".
 - c. El método `hasToken` devuelve un valor `boolean` para indicar si la cadena transferida contiene el token. Esto resulta útil para identificar qué cadena tiene el token que se va a sustituir por el nombre de la lista de nombres.
6. Codifique el método `writeMergedForms`. El plan general para este método consiste en leer toda la carta, línea a línea, para, a continuación, leer la lista entera de nombres y fusionarlos con la carta, sustituyendo el marcador de posición de la plantilla por un nombre de la lista y, posteriormente, escribiendo ese contenido como archivo. El resultado neto debería ser que, si la lista incluyera diez nombres, debería tener al final diez archivos de cartas personalizadas destinadas a los nombres de la lista correspondientes. Estos diez archivos se escribirán en el directorio de recursos.
 - a. Cree una instancia del elemento `FileScanInteractive` por defecto. Este argumento se necesita para el método `Files.readAllLines`.
`Charset cs = Charset.defaultCharset();`
 - b. Lea todas las líneas de la carta en el campo `formLetter` y todas las líneas de la lista de nombres en el campo `nameList`.
Nota: debido a que `writeMergedForms` devuelve `IOException`, no es necesario que inserte estas sentencias en un bloque `try`. El emisor de la llamada de este método es el encargado de manejar las excepciones que se puedan devolver.

```
formLetter = Files.readAllLines(form, cs);
nameList = Files.readAllLines(list, cs);
```

- c. Cree un bucle `for` para iterar con la lista de cadenas de nombres (`nameList`).
- d. Dentro de este, para el bucle, cree un nuevo objeto `List` para que incluya las cadenas de la carta. Necesitará este nuevo objeto `List` para que incluya las cadenas de plantilla de carta modificada que se van a escribir.

```
for (int j = 0; j < nameList.size(); j++) {
    customLetter = new ArrayList<>();
```

- e. Aún dentro del bucle `for`, tiene que crear un nombre para la carta personalizada. Una forma sencilla de hacerlo es usar el nombre de la lista de nombres. Debe sustituir los espacios del nombre por caracteres de subrayado para facilitar la lectura del nombre de archivo. Cree un nuevo objeto `Path` relativo a la ruta de acceso de plantilla de carta.

```
String formName = nameList.get(j).replace(' ',
    '_').concat(".txt");
Path formOut = form.getParent().resolve(formName);
```

- f. Cree otro bucle `for`, anidado en el primer bucle, para que se itere con las líneas de la plantilla de carta y busque la cadena de marcador de posición de token ("`<NAME>`") para sustituirla por el nombre `String` de `nameList`. Utilice el método `hasToken` para buscar el objeto `String` que contiene la cadena de token y sustituya esa cadena por la que contiene el nombre de `nameList`. Utilice el método `mergeName` para crear el nuevo objeto `String`. Agregue el objeto `String` modificado y el resto de cadenas de `formLetter` al nuevo `customLetter List`.

```
for (int k = 0; k < formLetter.size(); k++) {
    if (hasToken(formLetter.get(k))) {
        customLetter.add(mergeName(formLetter.get(k),
nameList.get(j)));
    } else {
        customLetter.add(formLetter.get(k));
    }
}
```

- g. Por último, aún dentro del primer bucle `for`, escriba el elemento `List` modificado del objeto `Strings` que representa la carta personalizada en el sistema de archivos mediante el método `Files.write`. Imprima un mensaje que indique que la escritura del archivo se ha realizado correctamente y cierre el bucle `for` exterior.

```
Files.write(formOut, customLetter, cs);
System.out.println ("Wrote form letter to: " +
nameList.get(j));
} // closing brace for the outer for loop
```

- h. Vuelva a formatear el código para asegurarse de que todo está en el lugar correcto. Pulse la combinación de teclas `Ctrl-Alt-F` o haga clic con el botón derecho en el panel del editor y seleccione `Format`.
- i. Guarde la clase `FileMerge`.
7. Modifique el proyecto `FormLetterWriter` para transferir el archivo de carta y el archivo de lista de nombres al método `main`.
- Haga clic con el botón derecho en el proyecto y seleccione `Properties`.
 - Seleccione `Run`.
 - En el campo de texto `Arguments`, introduzca:
`D:\labs\resources\FormTemplate.txt D:\labs\resources\NamesList.txt`
 - Haga clic en `OK`.

8. Ejecute el proyecto. Deben aparecer los nuevos archivos que se han creado con los nombres de la lista de nombres. Cada archivo se debe personalizar con el nombre de la lista de nombres. Por ejemplo, el archivo `Tom_McGinn.txt` debe contener:

Dear Tom McGinn,

It has come to our attention that you would like to prove your Java Skills. May we recommend that you consider certification from Oracle? Oracle has globally recognized Certification exams that will test your Java knowledge and skills.

Start with the Oracle Certified Java Associate exam, and then continue to the Oracle Certified Java Programmer Professional for a complete certification profile.

Good Luck!

Oracle University

Práctica 11-2: Nivel de resumen: Copia recursiva

Visión general

En esta práctica, escribirá clases Java que usen la clase `FileVisitor` para copiar de forma recursiva un directorio en otro.

Supuestos

Ha participado en la clase teórica de esta lección.

Tareas

1. Abra el proyecto `RecursiveCopyExercise` del directorio `D:\labs\11-NIO.2\practices`.
2. Amplíe la carpeta `Source Packages` y sus subcarpetas y busque la clase `Copy.java`.
 - a. Observe que la clase `Copy.java` contiene el método `main`.
 - b. La aplicación toma dos argumentos, una ruta de acceso `source` y otra `target`.
 - c. Si existen el archivo o el directorio de destino, al usuario se le pregunta si desea sobrescribirlos.
 - d. Si la respuesta es afirmativa (o la letra correspondiente), el método continúa.
 - e. Se crea una instancia de la clase `CopyFileTree` con `source` y `target`.
 - f. A continuación, esta instancia se transfiere al método `walkFileTree` (con el objeto de `Path source`).

Tendrá que proporcionar cuerpos de método para los métodos de la clase `CopyFileTree.java`.

3. Abra la clase `CopyFileTree.java`.
 - a. Esta clase implanta la interfaz de `FileVisitor`. Observe que `FileVisitor` es una interfaz genérica y que esta interfaz se incluye con la clase `Path`. Esto permite a la interfaz definir el tipo de argumentos transferidos a sus métodos.
 - b. `CopyFileTree` implanta todos los métodos definidos por `FileVisitor`.
 - c. Su tarea es escribir cuerpos de métodos para los métodos `preVisitDirectory` y `visitFile`. No necesitará el método `postVisitDirectory` y se le habrá proporcionado un cuerpo de método para el método `visitFileFailed`.
4. Escriba el cuerpo del método para `preVisitDirectory`. A este método se le llama para el nodo de inicio del árbol y para cada uno de los subdirectorios. Por tanto, debe copiar el directorio del origen en el destino. Si el archivo ya existe, puede ignorar esa excepción (porque está realizando la copia debido a que el usuario ha decidido sobrescribirlo).
 - a. Para empezar, cree un nuevo directorio relativo al destino transferido, pero que sea el nombre de nodo del origen. La llamada de método para realizar esta acción es:

```
Path newdir = target.resolve(source.relativeTo(target).getFileName());
```
 - b. En un bloque `try`, copie el directorio transferido al método `preVisitDirectory` en el directorio `newdir` que ha creado.
 - c. Puede ignorar cualquier excepción `FileAlreadyExistsException` que se detecte, ya que está sobrescribiendo las carpetas y los archivos existentes de esta copia.

- d. Capture cualquier otra excepción `IOExceptions` y utilice `SKIP_SUBTREE` para evitar que se produzcan errores repetidos.
5. Escriba el cuerpo del método para `visitFile`. La llamada a este método se produce cuando el nodo al que se llega es un archivo. El archivo se transfiere como argumento al método.
 - a. Como ocurre con `preVisitDirectory`, debe racionalizar el archivo al que se ha llegado (ruta de acceso de origen) con la ruta de acceso que desee para el destino. Utilice la misma llamada de método que anteriormente (pero esta vez use `file` en lugar de `dir`):


```
Path newdir = target.resolve(source.relativeize(file));
```
 - b. Como en el método `preVisitDirectory`, utilice el método `Files.copy` en un bloque `try`. Asegúrese de transferir `REPLACE_EXISTING` como opción para sobrescribir cualquier archivo que haya en el directorio.
 - c. Capture cualquier excepción `IOException` detectada y muestre un error.
 - d. Corrija cualquier importación que falte.
 - e. Guarde la clase.
6. Para probar la aplicación, copie un directorio (preferiblemente con subdirectorios) en otra ubicación del disco. Por ejemplo, copie el directorio `D:\labs\11-NIO.2` en `D:\Temp`.
 - a. Haga clic con el botón derecho en el proyecto y seleccione `Properties`.
 - b. Haga clic en `Run`.
 - c. Introduzca lo siguiente como argumentos:


```
D:\labs\11-NIO.2 D:\Temp
```
 - d. Haga clic en `OK`.
7. Ejecute el proyecto. Debe aparecer el siguiente mensaje:


```
Successfully copied D:\labs\11-NIO.2 to D:\Temp
```

 - a. Vuelva a ejecutar el proyecto y aparecerá esta pregunta:


```
Target directory exists. Overwrite existing files? (yes/no):
```

Práctica 11-2: Nivel detallado: Copia recursiva

Visión general

En esta práctica, escribirá clases Java que usen la clase `FileVisitor` para copiar de forma recursiva un directorio en otro.

Supuestos

Ha participado en la clase teórica de esta lección.

Tareas

1. Abra el proyecto `RecursiveCopyExercise` del directorio `D:\labs\11-NIO.2\practices`.
 - a. Seleccione `File > Open Project`.
 - b. Vaya a `D:\labs\11-NIO.2\practices`.
 - c. Seleccione `RecursiveCopyExercise`.
 - d. Active la casilla de control "Open as Main Project".
 - e. Haga clic en el botón `Open Project`.
2. Amplíe la carpeta `Source Packages` y sus subcarpetas y busque la clase `Copy.java`.
 - a. Observe que la clase `Copy.java` contiene el método `main`.
 - b. La aplicación toma dos argumentos, una ruta de acceso `source` y otra `target`.
 - c. Si existen el archivo o el directorio de destino, al usuario se le pregunta si desea sobrescribirlos.
 - d. Si la respuesta es afirmativa (o la letra correspondiente), el método continúa.
 - e. Se crea una instancia de la clase `CopyFileTree` con `source` y `target`.
 - f. A continuación, esta instancia se transfiere al método `walkFileTree` (con el objeto de `Path source`).

Tendrá que proporcionar cuerpos de método para los métodos de la clase `CopyFileTree.java`.
3. Abra la clase `CopyFileTree.java`.
 - a. Esta clase implanta la interfaz de `FileVisitor`. Observe que `FileVisitor` es una interfaz genérica y que esta interfaz se incluye con la clase `Path`. Esto permite a la interfaz definir el tipo de argumentos transferidos a sus métodos.
 - b. `CopyFileTree` implanta todos los métodos definidos por `FileVisitor`.
 - c. Su tarea es escribir cuerpos de métodos para los métodos `preVisitDirectory` y `visitFile`. No necesitará el método `postVisitDirectory` y se le habrá proporcionado un cuerpo de método para el método `visitFileFailed`.

4. Escriba el cuerpo del método para `preVisitDirectory`. A este método se le llama para el nodo de inicio del árbol y para cada uno de los subdirectorios. Por tanto, debe copiar el directorio del origen en el destino. Si el archivo ya existe, puede ignorar esa excepción (porque está realizando la copia debido a que el usuario ha decidido sobrescribirlo).

- a. Para empezar, cree un nuevo directorio relativo al destino transferido y que sea el nombre de nodo del origen. La llamada de método para realizar esta acción es:
`Path newdir = target.resolve(source.relativize(dir));`
- b. En un bloque `try`, copie el directorio transferido al método `preVisitDirectory` en el directorio `newdir` que ha creado.

```
try {
    Files.copy(dir, newdir);
```

- c. Puede ignorar cualquier excepción `FileAlreadyExistsException` que se detecte, ya que está sobrescribiendo las carpetas y los archivos existentes de esta copia.

```
} catch (FileAlreadyExistsException x) {
    // ignore
```

- d. Capture cualquier otra excepción `IOExceptions` y utilice la devolución `SKIP_SUBTREE` para evitar que se produzcan errores repetidos.

```
} catch (IOException x) {
    System.err.format("Unable to create: %s %s%n",
                      newdir, x);
    return SKIP_SUBTREE;
}
```

5. Escriba el cuerpo del método para `visitFile`. La llamada a este método se produce cuando el nodo al que se llega es un archivo. El archivo se transfiere como argumento al método.

- a. Como ocurre con `preVisitDirectory`, debe racionalizar el archivo al que se ha llegado (ruta de acceso de origen) con la ruta de acceso que desee para el destino. Utilice la misma llamada de método que anteriormente (pero esta vez use `file` en lugar de `dir`):

```
Path newdir = target.resolve(source.relativize(file));
```

- b. Como en el método `preVisitDirectory`, utilice el método `Files.copy` en un bloque `try`. Asegúrese de transferir `REPLACE_EXISTING` como opción para sobrescribir cualquier archivo que haya en el directorio.

```
try {
    Files.copy(file, newdir, REPLACE_EXISTING);
```

Nota: para usar el tipo de enumeración `REPLACE_EXISTING`, debe importar la clase de enumeración `java.nio.file.StandardCopyOption` con una importación estática, como esta:

```
import static java.nio.file.StandardCopyOption.*;
```

- c. Capture cualquier excepción `IOException` detectada y muestre un error.

```
} catch (IOException x) {  
    System.err.format("Unable to copy: %s %s%n", source, x);  
}
```

- d. Corrija cualquier importación que falte.
- e. Guarde la clase.
6. Para probar la aplicación, copie un directorio (preferiblemente con subdirectorios) en otra ubicación del disco. Por ejemplo, copie el directorio `D:\labs\11-NIO.2` en `D:\Temp`.
- a. Haga clic con el botón derecho en el proyecto y seleccione `Properties`.
- b. Haga clic en `Run`.
- c. Introduzca lo siguiente como argumentos:
`D:\labs\11-NIO.2 D:\Temp`
- d. Haga clic en `OK`.
7. Ejecute el proyecto. Debe aparecer el siguiente mensaje:
`Successfully copied D:\labs\11-NIO.2 to D:\Temp`
- a. Vuelva a ejecutar el proyecto y aparecerá esta pregunta:
`Target directory exists. Overwrite existing files? (yes/no):`

(Opcional) Práctica 11-3: Nivel de resumen: Uso de `PathMatcher` para realizar una supresión recursiva

Visión general

En esta práctica, escribirá un método `main` de Java que cree una clase `PathMatcher` y utilice `FileVisitor` para suprimir de forma recursiva un patrón de archivos o directorios.

Supuestos

Ha terminado la práctica anterior.

Tareas

1. Abra el proyecto `RecursiveDeleteExercise` del directorio de prácticas.
2. Amplíe las carpetas `Source Packages`.
3. Abra el archivo de clase `Delete.java`. Se trata de la clase que contiene el método `main`. La clase `main` acepta dos argumentos: el primero es la ruta de acceso de inicio y el otro es el patrón que se va a suprimir.
4. Debe codificar el resto de la clase. Busque en los comentarios las indicaciones sobre las acciones que debe realizar.
 - a. Para empezar, cree un objeto `PathMatcher` a partir de la cadena de búsqueda transferida como segundo argumento. Para obtener una instancia de `PathMatcher`, tendrá que usar la clase `FileSystems` para obtener una instancia de `path matcher` del sistema de archivos por defecto.
 - b. Cree un objeto `Path` a partir del primer argumento.
 - c. Si la ruta de acceso de inicio es un archivo, compárela con el patrón mediante la instancia de `PathMatcher` que ha creado. Si existe una coincidencia, suprima el archivo y, a continuación, finalice la aplicación.
 - d. Si la ruta de acceso de inicio es un directorio, cree una instancia de `DeleteFileTree` con el directorio de inicio y el objeto `PathMatcher` como argumentos iniciales en el constructor. Transfiera el directorio de inicio y el árbol de archivos a un método `Files.walkFileTree` para buscar de forma recursiva el patrón que se va a suprimir.
 - e. Corrija cualquier importación que falte.
 - f. Guarde la clase `Delete`.
5. Abra el archivo de clase `DeleteFileTree`. Esta clase implanta `FileVisitor`. Esta clase busca de forma recursiva instancias de archivos o directorios que coincidan con el objeto `PathMatcher` transferido en el constructor. Esta clase está terminada, con la excepción del método `delete`.
 - a. Al método `delete` lo llaman los métodos `preVisitDirectory` y `visitFile`. Debe comprobar si el archivo o el directorio a los que han llegado estos métodos coinciden con el patrón.
 - b. Solo es necesario que coincida el nombre de ruta de acceso en el nodo, por lo que use el método `Path.getFileName` para obtener el nombre de archivo al final de la ruta de acceso completa.
 - c. Si el nombre coincide, utilice el método `Files.delete` para intentar suprimir el patrón de archivo e imprimir una sentencia de resultado o imprimir un error si se detecta una excepción `IOException`.

- d. Guarde la clase `DeleteFileTree`.
- 6. Ejecute la aplicación `Delete` con un directorio temporal.
 - a. Por ejemplo, si ha terminado la primera práctica, podrá suprimir todos los archivos de clase Java del directorio `D:\Temp`.
 - b. Haga clic con el botón derecho en el proyecto y seleccione `Properties`.
 - c. Haga clic en `Run` e introduzca lo siguiente en el campo de texto `Arguments`:
`D:\Temp\examples *.class`
 - d. Ejecute el proyecto.

(Opcional) Práctica 11-3: Nivel detallado: Uso de `PathMatcher` para realizar una supresión recursiva

Visión general

En esta práctica, escribirá un método `main` de Java que cree una clase `PathMatcher` y utilice `FileVisitor` para suprimir de forma recursiva un patrón de archivos o directorios.

Supuestos

Ha terminado la práctica anterior.

Tareas

1. Abra el proyecto `RecursiveDeleteExercise` del directorio de prácticas.
 - a. Seleccione `File > Open Project`.
 - b. Vaya a `D:\labs\11-NIO.2\practices`.
 - c. Seleccione `RecursiveDeleteExercise`.
 - d. Haga clic en el botón `Open Project`.
2. Amplíe las carpetas `Source Packages`.
3. Abra el archivo de clase `Delete.java`. Se trata de la clase que contiene el método `main`. La clase `main` acepta dos argumentos: el primero es la ruta de acceso de inicio y el otro es el patrón que se va a suprimir.
4. Debe codificar el resto de la clase. Busque en los comentarios las indicaciones sobre las acciones que debe realizar.
 - a. Para empezar, cree un objeto `PathMatcher` a partir de la cadena de búsqueda transferida como segundo argumento. Para obtener una instancia de `PathMatcher`, tendrá que usar la clase `FileSystems` para obtener una instancia de `path matcher` del sistema de archivos por defecto.

```
PathMatcher matcher =
FileSystems.getDefault().getPathMatcher("glob:" + args[1]);
```
 - b. Cree un objeto `Path` a partir del primer argumento.

```
Path root = Paths.get(args[0]);
```

- c. Si la ruta de acceso de inicio es un archivo, compárela con el patrón mediante la instancia de `PathMatcher` que ha creado. Si existe una coincidencia, suprima el archivo y, a continuación, finalice la aplicación.

```
if (!Files.isDirectory(root)) {
    Path name = root.getFileName();
    if (name != null && matcher.matches(name)) {
        try {
            Files.delete(root);
            System.out.println("Deleted : " + root);
            System.exit(0);
        } catch (IOException e) {
            System.err.println("Exception deleting file: " +
                               root);
            System.err.println("Exception: " + e);
            System.exit(-1);
        }
    }
}
```

- d. Si la ruta de acceso de inicio es un directorio, cree una instancia de `DeleteFileTree` con el directorio de inicio y el objeto `PathMatcher` como argumentos iniciales en el constructor. Transfiera el directorio de inicio y el árbol de archivos a un método `Files.walkFileTree` para buscar de forma recursiva el patrón que se va a suprimir.

```
DeleteFileTree deleter = new DeleteFileTree(root, matcher);
try {
    Files.walkFileTree(root, deleter);
} catch (IOException e) {
    System.out.println("Exception: " + e);
}
```

- e. Corrija cualquier importación que falte.
- f. Guarde la clase `Delete`.
5. Abra el archivo de clase `DeleteFileTree`. Esta clase implanta `FileVisitor`. Esta clase busca de forma recursiva instancias de archivos o directorios que coincidan con el objeto `PathMatcher` transferido en el constructor. Esta clase está terminada, con la excepción del método `delete`.
- a. Al método `delete` lo llaman los métodos `preVisitDirectory` y `visitFile`. Debe comprobar si el archivo o el directorio a los que han llegado estos métodos coinciden con el patrón.
- b. Solo es necesario que coincida el nombre de ruta de acceso en el nodo, por lo que use el método `Path.getFileName` para obtener el nombre de archivo al final de la ruta de acceso completa.

```
Path name = file.getFileName();
```

- c. Si el nombre coincide, utilice el método `Files.delete` para intentar suprimir el patrón de archivo e imprimir una sentencia de resultado o imprimir un error si se detecta una excepción `IOException`.

```
if (matcher.matches(name)) {  
    //if (name != null && matcher.matches(name)) {  
    try {  
        Files.delete(file);  
        System.out.println("Deleted: " + file);  
    } catch (IOException e) {  
        System.err.println("Unable to delete: " + name);  
        System.err.println("Exception: " + e);  
    }  
}
```

- d. Guarde la clase `DeleteFileTree`.
6. Ejecute la aplicación `Delete` con un directorio temporal.
- Por ejemplo, si ha terminado la primera práctica, podrá suprimir todos los archivos de clase Java del directorio `D:\Temp`.
 - Haga clic con el botón derecho en el proyecto y seleccione `Properties`.
 - Haga clic en `Run` e introduzca lo siguiente en el campo de texto `Arguments`:
`D:\Temp\examples *.class`
 - Ejecute el proyecto.

Carlos Jaramillo (cajaramillo@gmail.com) has a non-transferable
license to use this Student Guide.

Prácticas de la lección 12: Thread

Capítulo 12

Prácticas de la lección 12: Visión general

Visión general de las prácticas

En estas prácticas, utilizará las funciones multithread del lenguaje de programación Java.

Práctica 12-1: Nivel de resumen: Sincronización de acceso a datos compartidos

Visión general

En esta práctica, agregará código a una aplicación existente. Debe determinar si el código se ejecuta en un entorno multithread y, en ese caso, activarle la protección de thread.

Supuestos

Ha revisado las secciones sobre el uso de la clase `Thread` y la palabra clave `synchronized` de esta lección.

Resumen

Abirá un proyecto en el que se adquieren camisas de una tienda. Se le proporcionará el código de lectura de archivos. Su tarea es agregar el código de manejo de excepciones adecuado.

Tareas

1. Abra el proyecto `Synchronized` como proyecto principal.
 - a. Seleccione `File > Open Project`.
 - b. Vaya a `D:\labs\12-Threading\practices`.
 - c. Seleccione `Synchronized` y active la casilla de control "Open as Main Project".
 - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyectos, pero evite abrir y revisar las clases proporcionadas en este momento. Intentará detectar si esta aplicación es multithread. Para ello, observará el comportamiento del código especificado.
3. Cree una clase `PurchasingAgent` en el paquete `com.example`.
4. Termine la clase `PurchasingAgent`.
 - a. Agregue un método `purchase`.

```
public void purchase() {}
```

- b. Termine el método `purchase`. El método `purchase()` debería:

- Obtener una referencia `Store`. Tenga en cuenta que la clase `Store` implanta el patrón de diseño Singleton.

```
Store store = Store.getInstance();
```

- Comprar una camisa (`Shirt`).
- Verificar que la tienda tiene al menos una camisa en existencias.

```
store.getShirtCount()
```

- Utilizar la tienda para autorizar una compra con tarjeta de crédito. Utilice el número de cuenta de tarjeta de crédito "1234" y un importe de compra de 15,00. Se devuelve un resultado `boolean`.

```
store.authorizeCreditCard("1234", 15.00)
```

- Si hay camisas en existencias y se ha autorizado la compra con la tarjeta de crédito, debe tomar una camisa de la tienda.

```
Shirt shirt = store.takeShirt();
```

- Imprimir el objeto `shirt` y un mensaje de que la acción se ha realizado correctamente si se ha adquirido la camisa, o bien un mensaje de fallo si no se ha adquirido.

- Ejecute el proyecto varias veces. Observe que la tienda solo contiene una camisa. Pueden aparecer varias posibles variaciones de la salida. Podría ver:
 - Dos mensajes de que la acción se ha realizado correctamente y dos objetos `shirt` (la salida puede que aparezca en orden variable).
 - Dos mensajes de que la operación se ha realizado correctamente, un objeto `shirt` y un valor `null`.
 - Dos mensajes de que la operación se ha realizado correctamente, un objeto `shirt` y una excepción.
 - Un mensaje de que la operación se ha realizado con éxito, un objeto `shirt` y un mensaje de fallo (comportamiento deseado, pero el menos probable).
- Descubra cómo se está usando la clase `PurchasingAgent`.
 - Utilice un constructor y una sentencia `print` para saber cuántas instancias de la clase `PurchasingAgent` se están creando al ejecutar la aplicación.
Recordatorio: en ocasiones, los objetos se crean por solicitud y, en otros casos, varias solicitudes pueden compartir un objeto. Las variaciones del modelo afectan al código que debe tener protección de `thread`.
 - En el método `purchase`, utilice el método `Thread.currentThread()` para obtener una referencia al `thread` que está ejecutando actualmente el método `purchase()`. Utilice una sola sentencia `print` para imprimir el nombre y el ID del `thread` en ejecución.
 - Ejecute el proyecto y observe la salida.
- Abra la clase `Store` y agregue un retraso al método `authorizeCreditCard`.
 - Obtenga un número aleatorio que oscile entre 1 y 3, el número de segundos de retraso. Imprima un mensaje que indique cuántos segundos se retrasará la ejecución.

```
int seconds = (int) (Math.random() * 3 + 1);
```

- Utilice el método `static` adecuado en la clase `Thread` para retrasar la ejecución entre 1 y 3 segundos.

Tarea opcional: ¿Qué ocurre si el retraso se interrumpe? ¿Cómo puede asegurarse de que la ejecución se retrasa el número deseado de segundos? O bien ¿se debería realizar otra acción?

8. Ejecute el proyecto varias veces. Debe aparecer un rastreo de pila para una excepción `java.util.NoSuchElementException`. Busque la línea en el método `com.example.PurchasingAgent.purchase` que aparece en el rastreo de pila. Revise la acción que se está produciendo en esa línea.
9. Utilice un bloque de código `synchronized` para crear un comportamiento previsible.
 - Modifique el método `purchase` en la clase `PurchasingAgent` para que incluya un bloque de código sincronizado.
Nota: no funcionará si se agrega `synchronized` a la firma de método o se usa un bloque `synchronized` que use el monitor del objeto `this`.
10. Ejecute el proyecto. Ahora debe aparecer el comportamiento deseado. En la ventana de salida, debería ver un mensaje de que la operación se ha realizado con éxito, un objeto `shirt` y un mensaje de fallo.

Práctica 12-1: Nivel detallado: Sincronización de acceso a datos compartidos

Visión general

En esta práctica, agregará código a una aplicación existente. Debe determinar si el código se ejecuta en un entorno multithread y, en ese caso, activarle la protección de thread.

Supuestos

Ha revisado las secciones sobre el uso de la clase `Thread` y la palabra clave `synchronized` de esta lección.

Resumen

Abirá un proyecto en el que se adquieren camisas de una tienda. Se le proporcionará el código de lectura de archivos. Su tarea es agregar el código de manejo de excepciones adecuado.

Tareas

1. Abra el proyecto `Synchronized` como proyecto principal.
 - a. Seleccione `File > Open Project`.
 - b. Vaya a `D:\labs\12-Threading\practices`.
 - c. Seleccione `Synchronized` y active la casilla de control "Open as Main Project".
 - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyectos, pero evite abrir y revisar las clases proporcionadas en este momento. Intentará detectar si esta aplicación es multithread. Para ello, observará el comportamiento del código especificado.
3. Cree una clase `PurchasingAgent` en el paquete `com.example`.
4. Termine la clase `PurchasingAgent`.
 - a. Agregue un método `purchase`. El método `purchase()` debería:
 - Obtener una referencia `Store`. Tenga en cuenta que la clase `Store` implanta el patrón de diseño `Singleton`.
 - Comprar una camisa (`Shirt`).
 - Verificar que la tienda tiene al menos una camisa en existencias.
 - Utilizar la tienda para autorizar una compra con tarjeta de crédito. Utilice el número de cuenta de tarjeta de crédito "1234" y un importe de compra de 15,00. Se devuelve un resultado `boolean`.
 - Si hay camisas en existencias y se ha autorizado la compra con la tarjeta de crédito, debe tomar una camisa de la tienda.
 - Imprimir el objeto `shirt` y un mensaje de que la acción se ha realizado correctamente si se ha adquirido la camisa, o bien un mensaje de fallo si no se ha adquirido.

```
public class PurchasingAgent {  
  
    public void purchase() {  
        Store store = Store.getInstance();
```

```

        if (store.getShirtCount() > 0 &&
store.authorizeCreditCard("1234", 15.00)) {
            Shirt shirt = store.takeShirt();
            System.out.println("The shirt is ours!");
            System.out.println(shirt);
        } else {
            System.out.println("No shirt for you");
        }
    }
}

```

5. Ejecute el proyecto varias veces. Observe que la tienda solo contiene una camisa. Pueden aparecer varias posibles variaciones de la salida. Podría ver:

- Dos mensajes de que la acción se ha realizado correctamente y dos objetos shirt (la salida puede que aparezca en orden variable).

```

Adding a shirt to the store.
Total shirts in stock: 1
The shirt is ours!
The shirt is ours!
Shirt ID: 1
Description: Polo
Color: Rainbow
Size: Large

Shirt ID: 1
Description: Polo
Color: Rainbow
Size: Large

```

- Dos mensajes de que la operación se ha realizado correctamente, un objeto shirt y un valor null.

```

Adding a shirt to the store.
Total shirts in stock: 1
The shirt is ours!
The shirt is ours!
null
Shirt ID: 1
Description: Polo
Color: Rainbow
Size: Large

```

- Dos mensajes de que la operación se ha realizado correctamente, un objeto shirt y una excepción.

```
Adding a shirt to the store.
Total shirts in stock: 1
The shirt is ours!
Shirt ID: 1
Description: Polo
Color: Rainbow
Size: Large
Exception in thread "Thread-0" java.util.NoSuchElementException
```

- Un mensaje de que la operación se ha realizado con éxito, un objeto shirt y un mensaje de fallo (comportamiento deseado, pero el menos probable).

```
Adding a shirt to the store.
Total shirts in stock: 1
The shirt is ours!
Shirt ID: 1
Description: Polo
Color: Rainbow
Size: Large

No shirt for you
```

6. Descubra cómo se está usando la clase PurchasingAgent.

- En la clase PurchasingAgent, utilice un constructor y una sentencia print para descubrir cuántas instancias de la clase PurchasingAgent se están creando al ejecutar la aplicación.

```
public PurchasingAgent() {
    System.out.println("Creating a purchasing agent");
}
```

Recordatorio: en ocasiones, los objetos se crean por solicitud y, en otros casos, varias solicitudes pueden compartir un objeto. Las variaciones del modelo afectan al código que debe tener protección de thread.

- En el método purchase, utilice el método Thread.currentThread() para obtener una referencia al thread que está ejecutando actualmente el método purchase(). Utilice una sola sentencia print para imprimir el nombre y el ID del thread en ejecución.

```
Thread t = Thread.currentThread();
System.out.println("Thread:" + t.getName() + "," + t.getId());
```

- Ejecute el proyecto y observe la salida.

7. Abra la clase `Store` y agregue un retraso al método `authorizeCreditCard`.

- `Math.random()` se usa para obtener un número aleatorio que oscile entre 1 y 3, el número de segundos de retraso.

```
int seconds = (int) (Math.random() * 3 + 1);
System.out.println("Sleeping for " + seconds + " seconds");
try {
    Thread.sleep(seconds * 1000);
} catch (InterruptedException e) {
    System.out.println("Interrupted");
}
```

8. Ejecute el proyecto varias veces. Debe aparecer un rastreo de pila para una excepción `java.util.NoSuchElementException`. Busque la línea en el método `com.example.PurchasingAgent.purchase` que aparece en el rastreo de pila. La llamada a `store.takeShirt()` está generando la excepción.

Nota: el retraso incluido en el paso anterior aumenta la probabilidad de que las llamadas a métodos `PurchasingAgent.purchase` simultáneas consideren ambas que se puede obtener una camisa, pero en otro momento. Tomar la camisa a casi el mismo tiempo suele producir algunos de los otros errores que se muestran en el paso 5.

9. Utilice un bloque de código `synchronized` para crear un comportamiento previsible.

- Modifique el método `purchase` en la clase `PurchasingAgent` para que incluya un bloque de código sincronizado.

```
synchronized (store) {
    if (store.getShirtCount() > 0 &&
        store.authorizeCreditCard("1234", 15.00)) {
        Shirt shirt = store.takeShirt();
        System.out.println("The shirt is ours!");
        System.out.println(shirt);
    } else {
        System.out.println("No shirt for you");
    }
}
```

Nota: no funcionará si agrega `synchronized` a la firma de método o se usa un bloque `synchronized` que use el monitor del objeto `this`.

10. Ejecute el proyecto. Ahora debe aparecer el comportamiento deseado. En la ventana de salida, debería ver un mensaje de que la operación se ha realizado con éxito, un objeto `shirt` y un mensaje de fallo.

```
Adding a shirt to the store.
Total shirts in stock: 1
The shirt is ours!
Shirt ID: 1
Description: Polo
Color: Rainbow
Size: Large

No shirt for you
```

Práctica 12-2: Nivel de resumen: Implantación de un programa multithread

Visión general

En esta práctica, creará un nuevo proyecto e iniciará un nuevo thread.

Supuestos

Ha revisado las secciones sobre el uso de la clase `Thread`.

Resumen

Crearé un proyecto que imprima de forma lenta un número en aumento. Se utilizará un nuevo thread para aumentar e imprimir el número. La aplicación debe esperar a que se pulse Intro para interrumpir los threads.

Tareas

1. Cree un nuevo proyecto `ThreadInterrupted` como proyecto principal.
 - a. Seleccione `File > New Project`.
 - b. Seleccione `Java` en `Categories` y `Java Application` en `Projects`. Haga clic en el botón `Next`.
 - c. Introduzca la siguiente información en el cuadro de diálogo “`Name and Location`”:
 - Project Name: `ThreadInterrupted`
 - Project Location: `D:\labs\12-Threading\practices`.
 - (activada) Create Main Class: `com.example.ThreadInterruptedMain`
 - (activada) Set as Main Project
 - d. Haga clic en el botón `Finish`.
2. Cree una clase `Counter` en el paquete `com.example`.
3. Termine la clase `Counter`. La clase `Counter` debe:
 - Implantar la interfaz de `Runnable`.
 - En el método `run`:
 - Cree una variable `int` denominada `x` e inicialícela en cero.
 - Cree un bucle que se repetirá hasta que se interrumpa el thread en ejecución.
 - Dentro del bucle, imprima y aumente el valor de `x`.
 - Dentro del bucle, establezca un retraso de 1 segundo. Realice una devolución desde el método `run` o salga del bucle si el thread se ha interrumpido durante el retraso.

4. Agregue lo siguiente al método `main` en la clase `ThreadInterruptedMain`:
 - Cree una instancia de `Counter`.
 - Cree un thread y transfiera a su constructor el objeto `Counter` que se puede ejecutar.
 - Inicie el thread.
5. Ejecute el proyecto. Debe aparecer una secuencia de números en aumento con un retraso de un segundo entre cada número. Observe que, si bien el método `main` ha terminado, la aplicación se sigue ejecutando.
6. Pare el proyecto.
 - a. Abra el menú Run.
 - b. Haga clic en Stop Build/Run.

Nota: también puede parar una compilación/ejecución haciendo clic en el cuadro de color rojo que aparece junto al lado izquierdo de la ventana de salida.

7. Modifique el método `main` de la clase `ThreadInterruptedMain`.

- Después de iniciar el thread, espere a que se pulse Intro en la ventana de salida. Puede utilizar el siguiente código:

```
try(BufferedReader br = new BufferedReader(new
InputStreamReader(System.in))) {
    br.readLine();
} catch (IOException e) {}
```

Nota: puede que tenga que corregir las importaciones y actualizar las propiedades del proyecto para que soporten las funciones de JDK 7.

- Imprima un mensaje que indique si el thread está o no activo.
 - Interrumpa el thread.
 - Establezca un retraso de un segundo (para permitir que termine el tiempo de thread) y, a continuación, imprima un mensaje que indique si el thread está o no activo.
8. Ejecute el proyecto. Debe aparecer una secuencia de números en aumento con un retraso de un segundo entre cada número. Pulse Intro mientras esté seleccionada la ventana de salida para cerrar la aplicación de forma correcta.

Práctica 12-2: Nivel detallado: Implantación de un programa multithread

Visión general

En esta práctica, creará un nuevo proyecto e iniciará un nuevo thread.

Supuestos

Ha revisado las secciones sobre el uso de la clase `Thread`.

Resumen

Crearé un proyecto que imprima de forma lenta un número en aumento. Se utilizará un nuevo thread para aumentar e imprimir el número. La aplicación debe esperar a que se pulse Intro para interrumpir los threads.

Tareas

1. Cree un nuevo proyecto `ThreadInterrupted` como proyecto principal.
 - a. Seleccione `File > New Project`.
 - b. Seleccione `Java` en `Categories` y `Java Application` en `Projects`. Haga clic en el botón `Next`.
 - c. Introduzca la siguiente información en el cuadro de diálogo “`Name and Location`”:
 - `Project Name`: `ThreadInterrupted`
 - `Project Location`: `D:\labs\12-Threading\practices`
 - (activada) `Create Main Class`: `com.example.ThreadInterruptedMain`
 - (activada) `Set as Main Project`
 - d. Haga clic en el botón `Finish`.
2. Cree una clase `Counter` en el paquete `com.example`.
3. Termine la clase `Counter`. La clase `Counter` debe:
 - Implantar la interfaz de `Runnable`.
 - En el método `run`:
 - Cree una variable `int` denominada `x` e inicialícela en cero.
 - Cree un bucle que se repetirá hasta que se interrumpa el thread en ejecución.
 - Dentro del bucle, imprima y aumente el valor de `x`.
 - Dentro del bucle, establezca un retraso de 1 segundo. Realice una devolución desde el método `run` o salga del bucle si el thread se ha interrumpido durante el retraso.


```
int x = 0;
while(!Thread.currentThread().isInterrupted()) {
    System.out.println("The current value of x is: " + x++);
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        return;
    }
}
```

4. Agregue lo siguiente al método `main` en la clase `ThreadInterruptedMain`:

- Cree una instancia de `Counter`.
- Cree un thread y transfiera a su constructor el objeto `Counter` que se puede ejecutar.
- Inicie el thread.

```
Runnable r = new Counter();
Thread t = new Thread(r);
t.start();
```

5. Ejecute el proyecto. Debe aparecer una secuencia de números en aumento con un retraso de un segundo entre cada número. Observe que, si bien el método `main` ha terminado, la aplicación se sigue ejecutando.

6. Pare el proyecto.

- Abra el menú `Run`.
- Haga clic en `Stop Build/Run`.

Nota: también puede parar una compilación/ejecución haciendo clic en el cuadro de color rojo que aparece junto al lado izquierdo de la ventana de salida.

7. Modifique las propiedades del proyecto para permitir la sentencia `try-with-resources`.

- Haga clic con el botón derecho en el proyecto `ThreadInterrupted` y haga clic en `Properties`.
- En el cuadro de diálogo `Project Properties`, seleccione la categoría `Sources`.
- En la lista desplegable `Source/Binary Format`, seleccione `JDK 7`.
- Haga clic en el botón `OK`.

8. Modifique el método `main` de la clase `ThreadInterruptedMain`.

- Después de iniciar el thread, espere a que se pulse `Intro` en la ventana de salida. Puede utilizar el siguiente código:

```
try(BufferedReader br = new BufferedReader(new
InputStreamReader(System.in))) {
    br.readLine();
} catch (IOException e) {}
```

- Agregue las sentencias de importación necesarias.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

- Imprima un mensaje que indique si el thread está o no activo.

```
System.out.println("Thread is alive:" + t.isAlive() );
```

- Interrumpa el thread.

```
t.interrupt();
```

- Establezca un retraso de un segundo (para permitir que termine el tiempo de thread) y, a continuación, imprima un mensaje que indique si el thread está o no activo.

```
try {
    Thread.sleep(1000);
} catch (InterruptedException e) {
}
System.out.println("Thread is alive:" + t.isAlive());
```

9. Ejecute el proyecto. Debe aparecer una secuencia de números en aumento con un retraso de un segundo entre cada número. Pulse Intro mientras esté seleccionada la ventana de salida para cerrar la aplicación de forma correcta.

Prácticas de la lección 13: Simultaneidad

Capítulo 13

Prácticas de la lección 13: Visión general

Visión general de las prácticas

En estas prácticas, utilizará el paquete `java.util.concurrent` y los subpaquetes del lenguaje de programación Java.

(Opcional) Práctica 13-1: Uso del paquete `java.util.concurrent`

Visión general

En esta práctica, modificará un proyecto existente para utilizar un elemento `ExecutorService` del paquete `java.util.concurrent`.

Supuestos

Ha revisado las secciones sobre el uso del paquete `java.util.concurrent`.

Resumen

Crearé un cliente de red multithread que leerá rápidamente el precio de una camisa de varios servidores. En lugar de tener que crear manualmente threads, utilizaré un elemento `ExecutorService` del paquete `java.util.concurrent`.

Tareas

1. Abra el proyecto `ExecutorService` como proyecto principal.
 - a. Seleccione `File > Open Project`.
 - b. Vaya a `D:\labs\13-Concurrency\practices`.
 - c. Seleccione `ExecutorService` y active la casilla de control "Open as Main Project".
 - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Ejecute la clase `NetworkServerMain` del paquete `com.example.server` haciendo clic con el botón derecho en ella y seleccionando `Run File`.
4. Abra la clase `NetworkClientMain` del paquete `com.example.client`.
5. Ejecute el paquete de la clase `NetworkClientMain` haciendo clic con el botón derecho en la clase y seleccionando `Run File`. Observe la cantidad de tiempo que se tarda en consultar todos los servidores por orden.
6. Cree una clase `NetworkClientCallable` en el paquete `com.example.client`.
 - Agregue un constructor y un campo para recibir y almacenar una referencia `RequestResponse`.
 - Implante la interfaz de `Callable` con un tipo genérico de `RequestResponse`.

```
public class NetworkClientCallable implements
    Callable<RequestResponse>
```

 - Rellene el método `call` mediante un elemento `java.net.Socket` y un elemento `java.util.Scanner` para leer la respuesta del servidor. Almacene el resultado en el objeto `RequestResponse` y devuélvalo.

Nota: puede que desee usar una sentencia `try-with-resources` para asegurarse de que los objetos `Socket` y `Scanner` están cerrados.
7. Modifique el método `main` de la clase `NetworkClientMain` para consultar los servidores por orden mediante un elemento `ExecutorService`.
 - a. Comente el contenido del método `main`.

- b. Obtenga un elemento `ExecutorService` que vuelva a utilizar un pool de threads almacenados en caché.
- c. Cree un objeto `Map` que se utilizará para unir una solicitud a una futura respuesta.

```
Map<RequestResponse, Future<RequestResponse>> callables = new  
HashMap<>();
```

- d. Codifique un bucle que creará un elemento `NetworkClientCallable` para cada una de las solicitudes de red.
 - Los servidores se deben estar ejecutando en el host local, en los puertos del 10000 al 10009.
 - Envíe cada elemento `NetworkClientCallable` a `ExecutorService`. Almacene cada elemento `Future` en el objeto `Map` creado en el paso 7c.
 - e. Cierre `ExecutorService`.
 - f. Espere a que finalicen todos los threads de `ExecutorService` durante 5 segundos.
 - g. Realice el bucle de los objetos `Future` almacenados en el objeto `Map` creado en el paso 7c. Imprima la respuesta de los servidores o un mensaje de error con los detalles del servidor si se ha producido un problema en la comunicación con un servidor.
8. Ejecute la clase `NetworkClientMain` haciendo clic con el botón derecho en la clase y seleccionando Run File. Observe la cantidad de tiempo que se tarda en consultar todos los servidores simultáneamente.
9. Cuando haya terminado de probar el cliente, asegúrese de seleccionar el separador de salida de `ExecutorService` y de cerrar la aplicación del servidor.

(Opcional) Práctica 13-2: Uso del marco Fork-Join

Visión general

En esta práctica, modificará un proyecto existente para que use el marco Fork-Join.

Supuestos

Ha revisado las secciones sobre el uso del marco Fork-Join.

Resumen

Se le proporciona un proyecto existente en el que ya se usa el marco Fork-Join para procesar los datos obtenidos en una matriz. Antes de procesar la matriz, se inicializa con números aleatorios. Actualmente, la inicialización tiene un único thread. Debe usar el marco Fork-Join para inicializar la matriz con números aleatorios.

Tareas

1. Abra el proyecto `ForkJoinFindMax` como proyecto principal.
 - a. Seleccione `File > Open Project`.
 - b. Vaya a `D:\labs\13-Concurrency\practices`.
 - c. Seleccione `ForkJoinFindMax` y active la casilla de control "Open as Main Project".
 - d. Haga clic en el botón `Open Project`.
2. Amplíe los directorios de proyecto.
3. Abra la clase `Main` del paquete `com.example`.
 - Revise el código en el método `main`. Anote cómo divide el método `compute` la matriz `data` si el recuento de elementos que procesar es demasiado grande.
3. Abra la clase `FindMaxTask` del paquete `com.example`.
 - Revise el código en la clase. Anote el bucle `for` que se usa para inicializar la matriz `data` con números aleatorios.
4. Cree una clase `RandomArrayAction` en el paquete `com.example`.
 - a. Agregue cuatro campos.

```
private final int threshold;
private final int[] myArray;
private int start;
private int end;
```

- b. Agregue un constructor que reciba parámetros y guarde sus valores en los campos definidos en el paso anterior.

```
public RandomArrayAction(int[] myArray, int start, int end, int
threshold)
```

- c. Amplíe la clase `RecursiveAction` del paquete `java.util.concurrent`.

Nota: se usa `RecursiveAction` cuando se necesita un elemento `ForkJoinTask` sin valores de retorno.

- d. Agregue el método `compute`. Observe que, a diferencia del método `compute` de un elemento `RecursiveTask`, el método `compute` de `RecursiveAction` devuelve `void`.

```
protected void compute() { }
```

- e. Inicie el método `compute`. Si el número de elementos que procesar está por debajo del umbral, debe inicializar la matriz.

```
for (int i = start; i <= end; i++) {
    myArray[i] = ThreadLocalRandom.current().nextInt();
}
```

Nota: se usa `ThreadLocalRandom` en lugar de `Math.random()` porque `Math.random()` no se escala cuando se ejecuta simultáneamente con varios threads y eliminaría cualquier ventaja de aplicar el marco Fork-Join a esta tarea.

- f. Finalice el método `compute`. Si el número de elementos que procesar está por encima o es igual que el umbral, debe encontrar el punto intermedio en la matriz y crear dos nuevas instancias de `RandomArrayAction` para cada una de las secciones de la matriz que procesar. Inicie cada uno de los elementos `RandomArrayAction`.

Nota: al iniciar un elemento `RecursiveAction`, puede utilizar el método `invokeAll` en lugar de la combinación `fork/join/compute` que se suele ver con un elemento `RecursiveTask`.

```
RandomArrayAction r1 = new RandomArrayAction(myArray, start,
midway, threshold);
RandomArrayAction r2 = new RandomArrayAction(myArray, midway +
1, end, threshold);
invokeAll(r1, r2);
```

5. Modifique el método `main` de la clase `Main` para que use la clase `RandomArrayAction`.
 - a. Comente el bucle `for` dentro del método `main` que inicializa la matriz `data` con valores aleatorios.
 - b. Tras la línea que crea `ForkJoinPool`, cree un nuevo elemento `RandomArrayAction`.
 - c. Utilice `ForkJoinPool` para llamar al elemento `ForkJoinPool`.
6. Para ejecutar el proyecto `ForkJoinFindMax`, haga clic con el botón derecho en él y seleccione *Run*.

Nota: si tiene un sistema con varias CPU, puede utilizar `System.currentTimeMillis()` para crear una referencia de las soluciones secuenciales y Fork-Join.

Prácticas de la lección 14: Creación de aplicaciones de base de datos con JDBC

Capítulo 14

Prácticas de la lección 14: Visión general

Visión general de las prácticas

En estas prácticas, trabajará con la base de datos JavaDB (Derby) y creará, leerá, actualizará y suprimirá datos de una base de datos SQL mediante la API JDBC Java.

Práctica 14-1: Nivel de resumen: Trabajo con la base de datos Derby y JDBC


Visión general

En esta práctica, iniciará la base de datos JavaDB (Derby), cargará algunos datos de ejemplo mediante un script y escribirá una aplicación para leer el contenido de una tabla de base de datos de empleados, e imprimirá los resultados en la consola.

Tareas

1. Cree la base de datos Employee mediante el script SQL proporcionado en el directorio de recursos.
 - a. Abra la ventana Services seleccionando Windows > Services o pulsando Ctrl-5.
 - b. Amplíe la carpeta Databases.
 - c. Haga clic con el botón derecho en JavaDB y seleccione Start Server.
 - d. Vuelva a hacer clic con el botón derecho en JavaDB y seleccione Create Database.
 - e. Introduzca la siguiente información:

Descripción de la página/ventana	Opciones o valores
Database Name	EmployeeDB
User Name	public
Password	tiger
Confirm Password	tiger

- f. Haga clic en OK.
- g. Haga clic con el botón derecho en la conexión que ha creado:
`jdbc:derby://localhost:1527/EmployeeDB[public on PUBLIC]` y seleccione Connect.
- h. Seleccione File > Open File.
- i. Navegue a `D:\labs\resources` y abra el script `EmployeeTable.sql`. El archivo se abrirá en una ventana SQL Execute.
- j. Seleccione la conexión que ha creado en la lista desplegable y haga clic en el icono Run-SQL  o pulse Ctrl-Mayús-E para ejecutar el script.
- k. Amplíe la conexión `EmployeeDB`. Verá que el esquema `PUBLIC` ya se ha creado. Amplíe el esquema `PUBLIC` y observe la tabla `Employee`.
- l. Vuelva a hacer clic con el botón derecho en la conexión y seleccione Execute Command para abrir otra ventana SQL. Introduzca el comando:
`select * from Employee`
y haga clic en el icono Run-SQL para ver el contenido de la tabla `Employee`.

2. Abra el proyecto `SimpleJDBCExample` y ejecútelo.
 - a. Deben aparecer todos los registros de la tabla `Employee`.
3. (Opcional) Agregue un comando SQL para añadir un nuevo registro `Employee`.
 - a. Modifique la clase `SimpleJDBCExample` para agregar un nuevo registro `Employee` a la base de datos.
 - b. La sintaxis para agregar una fila en una base de datos SQL es:
`INSERT INTO <table name> VALUES (<column 1 value>, <column 2 value>, ...)`
 - c. Utilice el método `executeUpdate` de `Statement` para ejecutar la consulta. ¿Cuál es el tipo de retorno para este método? ¿De qué valor debe ser el tipo de retorno? Realice una prueba para asegurarse de que el valor del retorno es el correcto.

Práctica 14-1: Nivel detallado: Trabajo con la base de datos Derby y JDBC


Visión general

En esta práctica, iniciará la base de datos JavaDB (Derby), cargará algunos datos de ejemplo mediante un script y escribirá una aplicación para leer el contenido de una tabla de base de datos de empleados, e imprimirá los resultados en la consola.

Tareas

1. Cree la base de datos Employee mediante el script SQL proporcionado en el directorio de recursos.
 - a. Abra la ventana Services seleccionando Windows > Services o pulsando Ctrl-5.
 - b. Amplíe la carpeta Databases.
 - c. Haga clic con el botón derecho en JavaDB y seleccione Start Server.
 - d. Vuelva a hacer clic con el botón derecho en JavaDB y seleccione Create Database.
 - e. Introduzca la siguiente información:

Descripción de la página/ventana	Opciones o valores
Database Name	EmployeeDB
User Name	public
Password	tiger
Confirm Password	tiger

- f. Haga clic en OK.
- g. Haga clic con el botón derecho en la conexión que ha creado:
`jdbc:derby://localhost:1527/EmployeeDB[public on PUBLIC]` y seleccione Connect.
- h. Seleccione File > Open File.
- i. Navegue a `D:\labs\resources` y abra el script `EmployeeTable.sql`. El archivo se abrirá en una ventana SQL Execute.
- j. Seleccione la conexión que ha creado en la lista desplegable y haga clic en el icono Run-SQL  o pulse Ctrl-Mayús-E para ejecutar el script.
- k. Amplíe la conexión `EmployeeDB`. Verá que el esquema `PUBLIC` ya se ha creado. Amplíe el esquema `PUBLIC`, amplíe `Tables` y, a continuación, amplíe la tabla `Employee`.
- l. Vuelva a hacer clic con el botón derecho en la conexión y seleccione Execute Command para abrir otra ventana SQL. Introduzca el comando:
`select * from Employee`
y haga clic en el icono Run-SQL para ver el contenido de la tabla `Employee`.

2. Abra el proyecto `SimpleJDBCExample` y ejecútelo.
 - a. Seleccione `Windows > Projects` o pulse `Ctrl-1`.
 - b. Seleccione `File > Open Project`.
 - c. Seleccione `D:\labs\12-JDBC\practices\SimpleJDBCExample`.
 - d. Seleccione "Open as Main Project".
 - e. Haga clic en OK.
 - f. Amplíe `Source Packages`, pruebe los paquetes y busque la clase `SimpleJDBCExample.java`.
 - g. Ejecute el proyecto: haga clic con el botón derecho en el proyecto y seleccione `Run` o haga clic en el icono `Run` o pulse `F6`.
 - h. Deben aparecer todos los registros de la tabla `Employee`.
3. (Opcional) Agregue un comando SQL para añadir un nuevo registro `Employee`.
 - a. Modifique la clase `SimpleJDBCExample` para agregar un nuevo registro `Employee` a la base de datos.
 - b. La sintaxis para agregar una fila en una base de datos SQL es:
`INSERT INTO <table name> VALUES (<column 1 value>, <column 2 value>, ...)`
 - c. Utilice el método `executeUpdate` de `Statement` para ejecutar la consulta. ¿Cuál es el tipo de retorno para este método? ¿De qué valor debe ser el tipo de retorno? Realice una prueba para asegurarse de que el valor del retorno es el correcto.
 - d. El código se puede parecer al siguiente:

```
query = "INSERT INTO Employee VALUES (200, 'Bill',  
'Murray', '1950-09-21', 150000)";  
if (stmt.executeUpdate(query) != 1) {  
    System.out.println ("Failed to add a new employee record");  
}
```

Nota: si vuelve a ejecutar la aplicación, devolverá una excepción, porque esta clave ya existe en la base de datos.

Práctica 14-2: Nivel de resumen: Uso del patrón de objeto de acceso a datos

Visión general

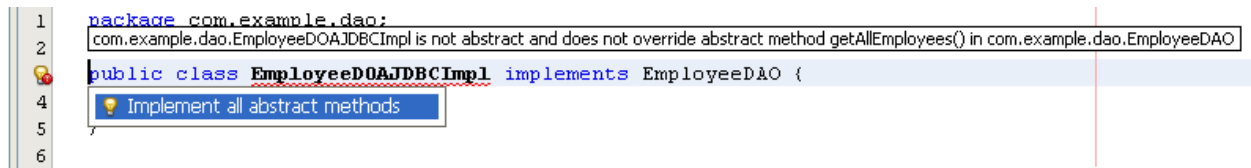
En esta práctica, tomará la aplicación Memory del DAO Employee existente y refactorizará el código para utilizar JDBC en su lugar. La solución de la lección “Excepciones y afirmaciones” ahora se llama `EmployeeDAOJDBC`. Tendrá que crear una clase `EmployeeDAOJDBCImpl` para sustituir a la clase `EmployeeDAOMemoryImpl` y modificar `EmployeeDAOFactory` para que devuelva una instancia de la nueva clase de implantación en lugar de la versión de Memory.

No será necesario que modifique las demás clases. En este ejemplo se muestra cómo una aplicación de objeto de acceso a datos con un buen diseño puede usar una clase de persistencia alternativa sin que se produzca ningún cambio significativo.

Tareas

1. Abra y examine el proyecto `EmployeeDAOJDBC` en la carpeta `D:\labs\12-JDBC\practices`.
 - a. En el paquete `com.example.test`, verá la clase `EmployeeTestInteractive`, que contiene el método `main` y que proporciona una interfaz de usuario basada en la consola. Mediante esta interfaz de usuario, podrá crear nuevos registros, leer todos los registros, actualizar un registro y suprimir un registro de la base de datos Employee. Observe cómo el método `main` crea una instancia de un objeto de acceso a datos (DAO).
 - b. En el paquete `com.example.model`, observe la clase `Employee`, que es un POJO (Plain Old Java Object) que encapsula todos los datos de un solo registro y fila de empleados en la tabla Employee. Tenga en cuenta que esta clase no incluye métodos `set`, sino solo métodos `get`. Una vez que se cree un objeto `Employee`, no se podrá cambiar. Es inmutable.
 - c. Amplíe el paquete `com.example.dao`. Busque la clase `EmployeeDAO` para ver los métodos que se espera que una implantación de esta interfaz implante. Cada uno de estos métodos devuelve una excepción `DAOException`. Tenga en cuenta que esta interfaz amplía `AutoCloseable`. Por tanto, tendrá que proporcionar un método `close()` para ser compatible con `AutoCloseable`.
 - d. Observe la clase `EmployeeDAOFactory`. Verá que esta clase tiene un método, `getFactory()`, que devuelve una instancia de un elemento `EmployeeDAOMemoryImpl`.
 - e. Observe la clase `EmployeeDAOMemoryImpl`. Esta clase es el arma secreta del patrón DAO. Se trata de la clase que `EmployeeDAOJDBCFactory` devuelve como instancia del método `createEmployeeDAO`. Se trata de la clase que se sustituirá con una implantación de JDBC.
2. Cree una nueva clase, `EmployeeDAOJDBCImpl`, que implante `EmployeeDAO` en el paquete `com.example.dao`.
 - a. Observe que la clase tiene un error.

3. Implante las firmas de método que define `EmployeeDAO`.
 - a. Haga clic en cualquier lugar de la línea que muestre un error (aparece con una bombilla con un punto rojo):
 - b. Pulse la combinación de teclas Alt-Intro para ver sugerencias para corregir el error en esta clase. Debería ver lo siguiente:



Nota: puede que los números de línea sean diferentes de los que se muestran en esta imagen.

- c. La clase debe implantar todos los métodos de la interfaz porque se trata de una clase concreta (no abstracta). Puede hacer que NetBeans proporcione todos los cuerpos de métodos. Para ello, pulse la tecla Intro para aceptar la sugerencia “Implement all abstract methods”.
 - d. Observará que el error del archivo desaparece de forma inmediata y que NetBeans ha proporcionado todas las firmas de métodos según las declaraciones de interfaz de `EmployeeDAO`.
 - e. Su siguiente tarea es agregar un constructor y rellenar los cuerpos de los métodos.
4. Agregue una variable de instancia privada, `con`, para que incluya una instancia de objeto `Connection`.
5. Escriba un constructor de nivel de paquete para la clase. El constructor de esta clase creará una instancia de un objeto `Connection` que los métodos de esta clase podrán reutilizar a lo largo de la duración de la aplicación. Asegúrese de capturar una excepción `SQLException`.
6. Escriba el cuerpo de un método para `add`. El método `add` crea un nuevo registro en la base de datos del objeto `Employee` transferido como parámetro. Recuerde que el comando SQL para crear un nuevo registro en la base de datos es: `INSERT INTO <table> VALUES (...)`.

Nota: utilice comillas simples para las cadenas y la fecha.

- a. Vuelva a emitir cualquier excepción `SQLException` detectada como excepción del tipo `DAOException`.
7. Escriba el cuerpo de un método para `findById`. Los métodos `update` y `delete` usan este método, que sirve para localizar un solo registro que se desea mostrar. Recuerde que el comando SQL para leer un solo registro es: `SELECT * FROM <table> WHERE <pk>=<value>`.
 - a. Vuelva a emitir cualquier excepción `SQLException` detectada como excepción del tipo `DAOException`.

8. Escriba el cuerpo de un método para `update`. El método `update` actualiza un registro existente en la base de datos del objeto `Employee` transferido como parámetro. Recuerde que el comando SQL para crear un nuevo registro en la base de datos es: `UPDATE <table> SET COLUMNNAME=<value>, COLUMNNAME=<value>, ... WHERE <pk>=<value>`.

Nota: asegúrese de agregar comillas simples a las cadenas y valores de datos.

 - a. Vuelva a emitir cualquier excepción `SQLException` detectada como excepción del tipo `DAOException`.
9. Escriba el cuerpo de un método para `delete`. El método `delete` prueba si un empleado existe en la base de datos mediante el método `findById` para, a continuación, suprimir el registro si existe. Recuerde que el comando SQL para suprimir un registro de la base de datos es: `DELETE FROM <table> WHERE <pk>=<value>`.
 - a. Vuelva a emitir cualquier excepción `SQLException` detectada como excepción del tipo `DAOException`.
10. Escriba el cuerpo del método para `getAllEmployees`. Este método devuelve una matriz de registros `Employee`. La consulta SQL para devolver todos los registros es muy sencilla: `SELECT * FROM <table>`.
 - a. Vuelva a emitir cualquier excepción `SQLException` detectada como excepción del tipo `DAOException`.
11. Escriba el cuerpo del método para `close`. Este método se define mediante la interfaz de `AutoCloseable`. Con este método se debe cerrar explícitamente el objeto `Connection` que ha creado en el constructor.
 - a. En lugar de volver a emitir la excepción `SQLException`, solo tiene que notificarla.
12. Guarde la clase. Corrija las importaciones que falten y los errores de compilación si aún no lo ha hecho.
13. Actualice `EmployeeDAOFactory` para que devuelva una instancia del nuevo objeto `EmployeeDAOJDBCImpl`.


```
return new EmployeeDAOJDBCImpl();
```
14. Agregue la clase del controlador JDBC Derby al proyecto, pero agregando el archivo `derbyclient.jar` al objeto Libraries del proyecto.
 - a. Haga clic con el botón derecho en la carpeta Libraries del proyecto y seleccione Add Jar/Folder.
 - b. Navegue hasta `D:\Program Files\Java\jdk1.7.0\db\lib`.
 - c. Seleccione `derbyclient.jar`.
 - d. La opción Absolute Path debería estar activada.
 - e. Haga clic en Open.
15. Guarde la clase actualizada y, si no tiene errores, compile y ejecute el proyecto. Esta aplicación tiene una función interactiva que permite consultar la base de datos y leer uno o todos los registro, buscar un empleado por ID, así como actualizar y suprimir un registro de empleado.

Práctica 14-2: Nivel detallado: Uso del patrón de objeto de acceso a datos

Visión general

En esta práctica, tomará la aplicación Memory del DAO Employee existente y refactorizará el código para utilizar JDBC en su lugar. La solución de la lección “Excepciones y afirmaciones” ahora se llama `EmployeeDAOJDBC`. Tendrá que crear una clase `EmployeeDAOJDBCImpl` para sustituir a la clase `EmployeeDAOMemoryImpl` y modificar `EmployeeDAOFactory` para que devuelva una instancia de la nueva clase de implantación en lugar de la versión de Memory.

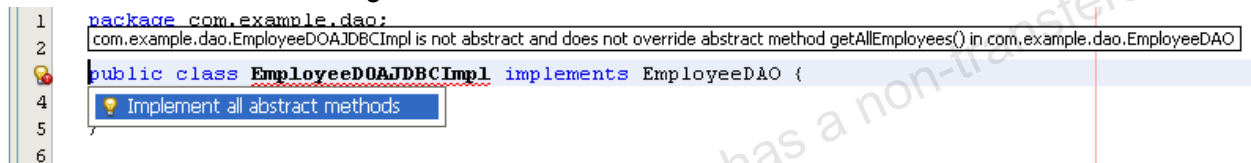
No será necesario que modifique las demás clases. En este ejemplo se muestra cómo una aplicación de objeto de acceso a datos con un buen diseño puede usar una clase de persistencia alternativa sin que se produzca ningún cambio significativo.

Tareas

1. Abra y examine el proyecto `EmployeeDAOJDBC` en la carpeta `D:\labs\12-JDBC\practices`.
 - a. En el paquete `com.example.test`, verá la clase `EmployeeTestInteractive`, que contiene el método `main` y que proporciona una interfaz de usuario basada en la consola. Mediante esta interfaz de usuario, podrá crear nuevos registros, leer todos los registros, actualizar un registro y suprimir un registro de la base de datos Employee. Observe cómo el método `main` crea una instancia de un objeto de acceso a datos (DAO).
 - b. En el paquete `com.example.model`, observe la clase `Employee`, que es un POJO (Plain Old Java Object) que encapsula todos los datos de un solo registro y fila de empleados en la tabla Employee. Tenga en cuenta que esta clase no incluye métodos `set`, sino solo métodos `get`. Una vez que se cree un objeto `Employee`, no se podrá cambiar. Es inmutable.
 - c. Amplíe el paquete `com.example.dao`. Busque la clase `EmployeeDAO` para ver los métodos que se espera que una implantación de esta interfaz implante. Cada uno de estos métodos devuelve una excepción `DAOException`. Tenga en cuenta que esta interfaz amplía `AutoCloseable`; por tanto, tendrá que proporcionar un método `close()` para ser compatible con `AutoCloseable`.
 - d. Observe la clase `EmployeeDAOFactory`. Verá que esta clase tiene un método, `getFactory()`, que devuelve una instancia de un elemento `EmployeeDAOMemoryImpl`.
 - e. Observe la clase `EmployeeDAOMemoryImpl`. Esta clase es el arma secreta del patrón DAO. Se trata de la clase que `EmployeeDAOJDBCFactory` devuelve como instancia del método `createEmployeeDAO`. Se trata de la clase que se sustituirá con una implantación de JDBC.

2. Cree una nueva clase denominada `EmployeeDAOJDBCImpl` en el paquete `com.example.dao`.
 - a. Haga clic con el botón derecho en el paquete `com.example.dao` y seleccione New Java Class.
 - b. Introduzca el nombre de clase `EmployeeDAOJDBCImpl` y haga clic en Finish.
 - c. Cambie esta clase para que implante `EmployeeDAO`.
 - d. Observe que se produce un error.
3. Implante las firmas de método que define `EmployeeDAO`.

- a. Haga clic en cualquier lugar de la línea que muestre un error (aparece con una bombilla con un punto rojo):
- b. Pulse la combinación de teclas Alt-Intro para ver sugerencias para corregir el error en esta clase. Debería ver lo siguiente:



Nota: puede que los números de línea sean diferentes de los que se muestran en esta imagen.

- c. La clase debe implantar todos los métodos de la interfaz porque se trata de una clase concreta (no abstracta). Puede hacer que NetBeans proporcione todos los cuerpos de métodos. Para ello, pulse la tecla Intro para aceptar la sugerencia "Implement all abstract methods".
 - d. Observará que el error del archivo desaparece de forma inmediata y que NetBeans ha proporcionado todas las firmas de métodos según las declaraciones de interfaz de `EmployeeDAO`.
 - e. Su siguiente tarea es agregar un constructor y rellenar los cuerpos de los métodos.
4. Agregue una variable de instancia privada, `con`, para que incluya una instancia de objeto `Connection`.


```
private Connection con = null;
```
 5. Escriba un constructor para la clase. El constructor de esta clase creará una instancia de un objeto `Connection` que los métodos de esta clase podrán reutilizar a lo largo de la duración de la aplicación.
 - a. Escriba el constructor para que use acceso de nivel de paquete. Esto permitirá que solo las clases del paquete creen una instancia de esta clase (como `EmployeeDAOFactory`).

```
EmployeeDAOJDBCImpl() {
```

- b. Abra la conexión mediante la URL, el nombre y la contraseña JDBC de la aplicación SimpleJDBCExample:

```
String url = "jdbc:derby://localhost:1527/EmployeeDB";
String username = "public";
String password = "tiger";
```

- c. En un bloque try (no en uno try-with-resources, ya que desea mantener esta conexión abierta hasta que salga de la aplicación), cree una instancia de un objeto Connection y capture cualquier excepción. Si no se puede conectar, salga de la aplicación.

Nota: lo ideal sería que indicara al usuario que no se ha podido realizar la conexión y que vuelva a intentar realizar la conexión una serie de veces antes de salir.

```
try {
    con = DriverManager.getConnection(url, username, password);
} catch (SQLException se) {
    System.out.println("Error obtaining connection with the
database: " + se);
    System.exit(-1);
}
```

6. Escriba el cuerpo de un método para add. El método add crea un nuevo registro en la base de datos del objeto Employee transferido como parámetro. Recuerde que el comando SQL para crear un nuevo registro en la base de datos es: INSERT INTO <table> VALUES (...).

- a. Suprima el código fijo que ha creado NetBeans para el método add.
b. Observe los demás métodos de la clase. Todos empiezan creando una instancia de un objeto Statement en una sentencia try-with-resources:

```
try (Statement stmt = con.createStatement()) {
}
```

- c. Dentro del bloque try, cree una consulta para insertar los valores transferidos en la instancia Employee a la base de datos. La cadena de salida debe ser parecida a la siguiente:

```
String query = "INSERT INTO EMPLOYEE VALUES (" + emp.getId()
    + ", ' " + emp.getFirstName() + ', ' "
    + "' " + emp.getLastName() + ', ' "
    + "' " +
    new java.sql.Date(emp.getBirthDate().getTime()) + ', ' "
    + emp.getSalary() + ")";
```

Observe el uso de comillas simples para las cadenas y la fecha.

- d. Como no espera que la consulta devuelva un resultado, el método de clase `Statement` adecuado que usar es `updateQuery`. Asegúrese de realizar una prueba para ver si la sentencia se ha ejecutado correctamente. Para ello, observe el resultado entero del método. Por ejemplo:

```
if (stmt.executeUpdate(query) != 1) {
    throw new DAOException("Error adding employee");
}
```

- e. Al final del bloque `try`, capture cualquier excepción `SQLException` detectada y encapsúela en la excepción `DAOException` para que las maneje la aplicación que realiza la llamada. Por ejemplo:

```
catch (SQLException se) {
    throw new DAOException("Error adding employee in DAO", se);
}
```

7. Escriba el cuerpo de un método para `findById`. Los métodos `update` y `delete` usan este método, que sirve para localizar un solo registro que se desea mostrar. Recuerde que el comando SQL para leer un solo registro es: "SELECT * FROM <table> WHERE <pk>=<value>".

- a. Suprima el código fijo que ha creado NetBeans para el método `findById`.
b. Cree una instancia de un objeto `Statement` en un bloque `try-with-resources`:

```
try (Statement stmt = con.createStatement()) {
    ...
}
```

- c. Dentro del bloque `try`, escriba una sentencia de consulta para que incluya el ID de entero transferido como argumento al método y ejecute la consulta; para ello, devuelva una instancia de `ResultSet`:

```
String query = "SELECT * FROM EMPLOYEE WHERE ID=" + id;
ResultSet rs = stmt.executeQuery(query);
```

- d. Pruebe la instancia `ResultSet` para ver si hay valores null con el método `next()` y devuelva el resultado como un nuevo objeto `Employee`:

```
if (!rs.next()) {
    return null;
}
return (new Employee(rs.getInt("ID"),
    rs.getString("FIRSTNAME"),
    rs.getString("LASTNAME"),
    rs.getDate("BIRTHDATE"),
    rs.getFloat("SALARY")));
```

- e. Al final del bloque try, capture cualquier excepción `SQLException` detectada y encapsúela en la excepción `DAOException` para que las maneje la aplicación que realiza la llamada. Por ejemplo

```
catch (SQLException se) {
    throw new DAOException("Error finding employee in DAO", se);
}
```

8. Escriba el cuerpo de un método para `update`. El método `update` actualiza un registro existente en la base de datos del objeto `Employee` transferido como parámetro. Recuerde que el comando SQL para crear un nuevo registro en la base de datos es: "UPDATE <table> SET COLUMNNAME=<value>, COLUMNNAME=<value>, ... WHERE <pk>=<value>".

Nota: asegúrese de agregar comillas simples a las cadenas y valores de datos.

- a. Suprima el código fijo que ha creado NetBeans para el método `update`.
b. Cree una instancia de un objeto `Statement` en un bloque try-with-resources:

```
try (Statement stmt = con.createStatement()) {
    }
}
```

- c. Dentro del bloque try, cree la consulta SQL UPDATE desde el objeto `Employee` transferido:

```
String query = "UPDATE EMPLOYEE "
    + "SET FIRSTNAME='" + emp.getFirstName() + "',"
    + "LASTNAME='" + emp.getLastName() + "',"
    + "BIRTHDATE='" + new
java.sql.Date(emp.getBirthDate().getTime()) + "',"
    + "SALARY=" + emp.getSalary()
    + "WHERE ID=" + emp.getId();
```

- d. Puede que desee realizar una prueba para ver que la actualización se ha realizado correctamente mediante la evaluación del valor de retorno del método `executeUpdate`:

```
if (stmt.executeUpdate(query) != 1) {
    throw new DAOException("Error updating employee");
}
```

- e. Al final del bloque try, capture cualquier excepción `SQLException` detectada y encapsúela en la excepción `DAOException` para que las maneje la aplicación que realiza la llamada. Por ejemplo

```
catch (SQLException se) {
    throw new DAOException("Error updating employee in DAO",
se);
}
```

9. Escriba el cuerpo de un método para `delete`. El método `delete` prueba si un empleado existe en la base de datos mediante el método `findById` para, a continuación, suprimir el registro si existe. Recuerde que el comando SQL para suprimir un registro de la base de datos es: "DELETE FROM <table> WHERE <pk>=<value>".

- a. Suprima el código fijo que ha creado NetBeans para el método `delete`.
- b. Llame al método `findById` con el `id` transferido como parámetro y, si el registro ha devuelto `null`, devuelva una nueva excepción `DAOException`.

```
Employee emp = findById(id);
if (emp == null) {
    throw new DAOException("Employee id: " + id + " does not exist to delete.");
}
```

- c. Cree una instancia de un objeto `Statement` en un bloque `try-with-resources`:

```
try (Statement stmt = con.createStatement()) {

}
```

- d. Dentro del bloque `try`, cree la consulta SQL DELETE y pruebe el resultado devuelto para asegurarse de que se ha modificado un solo registro. En caso contrario, devuelva una nueva excepción `DAOException`:

```
String query = "DELETE FROM EMPLOYEE WHERE ID=" + id;
if (stmt.executeUpdate(query) != 1) {
    throw new DAOException("Error deleting employee");
}
```

- e. Al final del bloque `try`, capture cualquier excepción `SQLException` detectada y encapsúlela en la excepción `DAOException` para que las maneje la aplicación que realiza la llamada. Por ejemplo:

```
catch (SQLException se) {
    throw new DAOException("Error deleting employee in DAO",
se);
}
```

10. Escriba el cuerpo del método para `getAllEmployees`. Este método devuelve una matriz de registros `Employee`. La consulta SQL para devolver todos los registros es muy sencilla: "SELECT * FROM <table>".

- a. Suprima el código fijo que ha creado NetBeans para el método `getAllEmployees`.
- b. Cree una instancia de un objeto `Statement` en un bloque `try-with-resources`:

```
try (Statement stmt = con.createStatement()) {

}
```

- c. Dentro del bloque try, cree y ejecute la consulta para que devuelva todos los registros de empleados:

```
String query = "SELECT * FROM EMPLOYEE";
ResultSet rs = stmt.executeQuery(query);
```

- d. La forma más fácil de crear una matriz de empleados que devolver es usar un objeto Collection, ArrayList, para, a continuación, convertir el objeto ArrayList en una matriz. Itere con ResultSet y agregue cada uno de los registros a ArrayList. En la sentencia de retorno, utilice el método toArray para convertir la recopilación en una matriz:

```
ArrayList<Employee> emps = new ArrayList<>();
while (rs.next()) {
    emps.add(new Employee(rs.getInt("ID"),
                           rs.getString("FIRSTNAME"),
                           rs.getString("LASTNAME"),
                           rs.getDate("BIRTHDATE"),
                           rs.getFloat("SALARY")));
}
return emps.toArray(new Employee[0]);
```

- e. Al final del bloque try, capture cualquier excepción SQLException detectada y encapsúlela en la excepción DAOException para que las maneje la aplicación que realiza la llamada. Por ejemplo:

```
catch (SQLException se) {
    throw new DAOException("Error getting all employees in DAO",
                           se);
}
```

11. Escriba el cuerpo del método para close. Este método se define mediante la interfaz de AutoCloseable. Con este método se debe cerrar explícitamente el objeto Connection que ha creado en el constructor.

- Suprima el código fijo que ha creado NetBeans para el método close.
- En un bloque try (debe usar un bloque try, porque Connection.close devuelve una excepción que se debe capturar o volver a emitir), llame al método close en la instancia de objeto Connection, con. En lugar de volver a emitir la excepción, solo tiene que notificarla.

```
try {
    con.close();
} catch (SQLException se) {
    System.out.println ("Exception closing Connection: " + se);
}
```

12. Guarde la clase. Corrija las importaciones que falten y los errores de compilación si aún no lo ha hecho.

13. Actualice `EmployeeDAOFactory` para que devuelva una instancia del nuevo objeto `EmployeeDAOJDBCImpl`.

```
return new EmployeeDAOJDBCImpl();
```

14. Agregue la clase del controlador JDBC Derby al proyecto, agregando el archivo `derbyclient.jar` al objeto Libraries del proyecto.
- Haga clic con el botón derecho en la carpeta Libraries del proyecto y seleccione Add Jar/Folder.
 - Navegue hasta `D:\Program Files\Java\jdk1.7.0\db\lib`.
 - Seleccione `derbyclient.jar`.
 - La opción Absolute Path debería estar activada.
 - Haga clic en Open.
15. Guarde la clase actualizada y, si no tiene errores, compile y ejecute el proyecto. Esta aplicación tiene una función interactiva que permite consultar la base de datos y leer uno o todos los registros, buscar un empleado por ID, así como actualizar y suprimir un registro de empleado.

Carlos Jaramillo (cajaramillo@gmail.com) has a non-transferable
license to use this Student Guide.

Prácticas de la lección 15: Localización

Capítulo 15

Prácticas de la lección 15: Visión general

Visión general de las prácticas

En estas prácticas, creará una aplicación de fecha similar al ejemplo usado en la lección. Para cada práctica, se le proporcionará un proyecto de NetBeans. Realice el proyecto como se indica en las instrucciones.

Práctica 15-1: Nivel de resumen: Creación de una aplicación de fecha localizada

Visión general

En esta práctica, creará una aplicación basada en texto que muestra las fechas y horas de distintas formas. Creará los grupos de recursos necesarios para localizar la aplicación para francés, chino simplificado y ruso.

Supuestos

Ha asistido a la clase teórica de esta lección. Tiene acceso a la documentación de la API de JDK 7.

Resumen

Crearé una aplicación de fecha basada en texto que mostrará la siguiente información de fecha para hoy:

- Fecha por defecto
- Fecha larga
- Fecha corta
- Fecha completa
- Hora completa
- Día de la semana
- Y un día y hora personalizados que mostrarán: el día de la semana, la fecha larga, la era, la hora y la zona horaria.

Localizaré la aplicación para que muestre esta información en chino simplificado y ruso. El usuario debe ser capaz de cambiar de un idioma a otro.

Aquí se muestra la salida de la aplicación en inglés.

```
=== Date App ===
Default Date is: Aug 1, 2011
Long Date is: August 1, 2011
Short Date is: 8/1/11
Full Date is: Monday, August 1, 2011
Full Time is: 10:13:56 AM MDT
Day of week is: Monday
My custom day and time is: Monday August 1, 2011 AD 10:13:56
Mountain Daylight Time

--- Choose Language Option ---
1. Set to English
2. Set to French
3. Set to Chinese
4. Set to Russian
q. Enter q to quit
Enter a command:
```

Tareas

Abra el proyecto `Localized-Practice01` de NetBeans y realice los siguientes cambios:

1. Edite el archivo `DateApplication.java`.
2. Cree un grupo de mensajes para ruso y chino simplificado.
 - El texto traducido de los menús se puede encontrar en el archivo `MessagesText.txt` del directorio `practices`.
3. Agregue código para que se muestren los formatos de fecha especificados (indicados con comentarios) y texto localizado.
4. Agregue código para cambiar el objeto `Locale` según los datos introducidos por el usuario.
5. Ejecute el archivo `DateApplication.java` y verifique que funciona de la forma prevista.

Práctica 15-1: Nivel detallado: Creación de una aplicación de fecha localizada

Visión general

En esta práctica, creará una aplicación basada en texto que muestra las fechas y horas de distintas formas. Creará los grupos de recursos necesarios para localizar la aplicación para francés, chino simplificado y ruso.

Supuestos

Ha asistido a la clase teórica de esta lección. Tiene acceso a la documentación de la API de JDK 7.

Resumen

Crearé una aplicación de fecha basada en texto que mostrará la siguiente información de fecha para hoy:

- Fecha por defecto
- Fecha larga
- Fecha corta
- Fecha completa
- Hora completa
- Día de la semana
- Y un día y hora personalizados que mostrarán: el día de la semana, la fecha larga, la era, la hora y la zona horaria.

Localizaré la aplicación para que muestre esta información en chino simplificado y ruso. El usuario debe ser capaz de cambiar de un idioma a otro.

Aquí se muestra la salida de la aplicación en inglés.

```
=== Date App ===
Default Date is: Aug 1, 2011
Long Date is: August 1, 2011
Short Date is: 8/1/11
Full Date is: Monday, August 1, 2011
Full Time is: 10:13:56 AM MDT
Day of week is: Monday
My custom day and time is: Monday August 1, 2011 AD 10:13:56
Mountain Daylight Time

--- Choose Language Option ---
1. Set to English
2. Set to French
3. Set to Chinese
4. Set to Russian
q. Enter q to quit
Enter a command:
```

Tareas

Abra el proyecto `Localized-Practice01` de NetBeans y realice los siguientes cambios:

1. Edite el archivo `DateApplication.java`.
2. Abra el archivo `MessagesText.txt` que se encuentra en el directorio `practices` de esta práctica en un editor de texto.
3. Cree un archivo de grupo de mensajes para el texto en ruso denominado `MessagesBundle_ru_RU.properties`.
 - Haga clic con el botón derecho en el proyecto y seleccione `New > Other > Other > Properties File`.
 - Haga clic en `Next`.
 - Introduzca `MessagesBundle_ru_RU` en el campo `File Name`.
 - Haga clic en `Browse`.
 - Seleccione el directorio `src`.
 - Haga clic en `Select Folder`.
 - Haga clic en `Finish`.
 - Pegue el texto de ruso localizado en el archivo y guárdelo.
4. Cree un archivo de grupo de mensajes para el texto de chino simplificado denominado `MessagesBundle_zh_CN.properties`.
 - Haga clic con el botón derecho en el proyecto y seleccione `New > Other > Other > Properties File`.
 - Haga clic en `Next`.
 - Introduzca `MessagesBundle_zh_CN` en el campo `File Name`.
 - Haga clic en `Finish`.
 - Pegue el texto de chino simplificado localizado en el archivo y guárdelo.

5. Actualice el código que define la configuración regional según los datos introducidos por el usuario.

```
public void setEnglish(){
    currentLocale = Locale.US;
    messages = ResourceBundle.getBundle("MessagesBundle",
currentLocale);
}

public void setFrench(){
    currentLocale = Locale.FRANCE;
    messages = ResourceBundle.getBundle("MessagesBundle",
currentLocale);
}

public void setChinese(){
    currentLocale = Locale.SIMPLIFIED_CHINESE;
    messages = ResourceBundle.getBundle("MessagesBundle",
currentLocale);
}

public void setRussian(){
    currentLocale = ruLocale;
    this.messages =
ResourceBundle.getBundle("MessagesBundle", currentLocale);
}
```

6. Agregue el código que muestra la información de fecha al método printMenu.

```
df = DateFormat.getDateInstance(DateFormat.DEFAULT,
currentLocale);
pw.println(messages.getString("date1") + " " +
df.format(today));
df = DateFormat.getDateInstance(DateFormat.LONG,
currentLocale);
pw.println(messages.getString("date2") + " " +
df.format(today));
df = DateFormat.getDateInstance(DateFormat.SHORT,
currentLocale);
pw.println(messages.getString("date3") + " " +
df.format(today));
df = DateFormat.getDateInstance(DateFormat.FULL,
currentLocale);
pw.println(messages.getString("date4") + " " +
df.format(today));
df = DateFormat.getTimeInstance(DateFormat.FULL,
currentLocale);
```

```
        pw.println(messages.getString("date5") + " " +
df.format(today));
        sdf = new SimpleDateFormat("EEEE", currentLocale);
        pw.println(messages.getString("date6") + " " +
sdf.format(today));
        sdf = new SimpleDateFormat("EEEE MMMM d, y G kk:mm:ss
zzzz", currentLocale);
        pw.println(messages.getString("date7") + " " +
sdf.format(today));
```

7. Ejecute el archivo `DateApplication.java` y verifique que funciona de la forma prevista.

Práctica 15-2: Nivel de resumen: Localización de una aplicación JDBC (opcional)

Visión general

En esta práctica, localizará la aplicación JDBC que ha creado en las prácticas de la lección “Creación de aplicaciones de base de datos con JDBC”.

Supuestos

Ha asistido a la clase teórica de esta lección. Ha realizado las prácticas de la lección “Creación de aplicaciones de base de datos con JDBC”.

Resumen

Localizará la aplicación JDBC de la lección anterior. Identificará cualquier objeto que muestre información de un menú u objeto y lo cambiará para que se muestren los mensajes localizados en lugar del texto estático.

Localizará la aplicación para que muestre esta información en inglés, francés y ruso. El usuario debe ser capaz de cambiar de un idioma a otro.

Tareas

En este ejercicio práctico, tendrá varias opciones de proyecto. En primer lugar, puede usar los archivos de proyecto de la lección 14, práctica 2, “Uso del patrón de objeto de acceso a datos” y, simplemente, continuar con este proyecto. También puede abrir el proyecto `Practice02` de esta lección. Realice los siguientes pasos:

1. Abra el archivo `EmployeeTestInteractive.java`. Examine el código fuente y determine cuáles de los mensajes impresos en la consola se deben convertir en grupos de recursos. Observe que no toda la salida de texto está incluida en este archivo de clase.

Nota: no tiene que incluir mensajes de error en el grupo. Solo se deben incluir los mensajes de petición de datos e informativos.

2. Es necesario realizar un pequeño cambio en la interfaz de usuario.

El aspecto actual de la interfaz principal es parecido a este:

```
[C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit:
```

Al cambiar la interfaz de usuario por lo siguiente se facilita la traducción solo de las palabras del menú.

```
[C] - Create | [R] - Read | [U] - Update | [D] - Delete | [L] -  
List | [S] - Set Language | [Q] - Quit:
```

De esta forma se separan los comandos de un carácter de las palabras. Para la solución, solo se han traducido las palabras. Por supuesto, podría traducir ambos. Observe que se ha agregado una nueva opción para definir el idioma.

3. Cree un grupo de mensajes para inglés, francés y ruso.
 - El texto traducido de los menús se puede encontrar en el archivo `MessagesText02.txt` del directorio `practices`.
4. Agregue un objeto `ResourceBundle` a cualquier objeto que muestre información relacionada con el menú. Sustituya el texto estático por una llamada al grupo de recursos y obtenga el mensaje de cadena adecuado.
5. Examine todo el código fuente relacionado con la fecha. Asegúrese de que la información de fecha se imprime con el formato localizado adecuado.
6. Cuando haya terminado, ejecute `EmployeeTestInteractive.java` y asegúrese de que todos los menús se han localizado.
7. También podrá realizar las siguientes mejoras:
 - Localizar todos los mensajes de error de la aplicación.
 - Localizar las opciones de un solo carácter del menú principal.