|  |  |  |
| --- | --- | --- |
| de | Carátula para entrega de prácticas | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Ing. Jorge Ángel Hernández López |
| *Asignatura:* | Programación Orientada a Objetos |
| *Grupo:* | 04 |
| *No. de práctica(s):* | 06 |
| *Integrante(s):* | Barrera Treviño José Gerardo  Muñoz San Agustin Victoria Monserrat  Leocadio Chávez Rodrigo  Overa Bravo Cynthia Carolina  Velasco Garcia Santiago |
| *No. de lista o brigada:* | Brigada 01 |
| *Semestre:* | 2024-1 |
| *Fecha de entrega:* | 03 de Octubre de 2023 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_\_\_

# **Introducción:**

Para la realización de este programa utilizamos elementos y temas como:

* Paquetes:

Los paquetes es un mecanismo utilizado por java que tiene como finalidad facilitar el modularidad de un código, el cual puede en su interior contener una p más de una definición de interfaz y/o de clases, distribuyéndose habitualmente como un archivo, Para poder usar los elementos que están dentro de un paquete se necesita importar dicho módulo de código utilizando la sentencia “Import”.

* Importar paquetes

Los paquetes de java contienen colecciones distintas para que el código funcione y sea útil que necesitaremos abordar el desarrollo de un nuevo proyecto, he aquí que utilizamos la sentencia “Import” que funciona para importar en el ámbito actual las definiciones de otro paquete y poder utilizarlas creando objetos y accediendo a las clases, etc.

La cláusula import puede utilizarse para importar un elemento concreto de un paquete, facilitando el nombre de este seguido de un punto y el identificador de dicho elemento. Por ejemplo, para importar la clase Math del paquete java.lang, pudiendo así acceder a la constante PI y las funciones matemáticas que aporta, bastaría con la siguiente línea:

import java.lang.Math;

* Paquetes propios

En el contexto de Java, los "paquetes propios" generalmente se refieren a paquetes o packages que has creado tú mismo como parte de tu programa o aplicación Java. Un paquete es una forma de organizar clases relacionadas en Java. Proporciona un espacio de nombres para evitar conflictos de nombres de clases y ayuda a estructurar tu código de manera más organizada y modular.

Cuando se desarrolla una aplicación Java, se puede organizar las clases en paquetes para mantener una jerarquía de directorios y agrupar clases relacionadas. Estos paquetes pueden ser considerados como "propios" porque se han creado específicamente para dicho proyecto. Los paquetes propios ayudan a mantener el código más limpio y facilitan su mantenimiento.

* Compilación y ejecución en clases en paquetes

Cuando se trabaja con clases en paquetes en Java, el proceso de compilación y ejecución es ligeramente diferente al de las clases que no están en paquetes. A continuación, se presentan los pasos generales para compilar y ejecutar clases en paquetes en Java:

**Compilación**

1. Abre una terminal y navega al directorio raíz del proyecto `miapp`.

2. Ejecuta el compilador de Java (`javac`) para compilar las clases en el paquete `miapp`. Puede usar un comodín (`\*`) para compilar todas las clases en el paquete:

javac -d bin src/miapp/\*.java

3. Esto compilará las clases `Clase1.java` y `Clase2.java` y las colocará en el directorio `bin/miapp`.

**Ejecución:**

1. Una vez que las clases están compiladas, se pueden ejecutar desde el directorio raíz del proyecto. Asegúrese de que el directorio raíz esté en el classpath para que Java pueda encontrar las clases en el paquete:

java -cp bin miapp.Clase1

2. Esto ejecutará la clase `Clase1` ubicada en el paquete `miapp`. Si hay una clase con un método `main` y se desea ejecutarlo, asegúrese de que la clase `Clase1` tenga un método `main` definido y que esté configurado correctamente como punto de entrada.

Es importante ajustar los comandos y rutas de acuerdo con la estructura del proyecto y los nombres de las clases y paquetes. La estructura de directorios y los nombres de los paquetes y clases en el proyecto pueden variar, por lo que los comandos deben adaptarse según corresponda.

* Distribución de aplicación

La distribución de una aplicación Java implica empaquetar y proporcionar todos los recursos y archivos necesarios para que la aplicación se pueda ejecutar en otros sistemas. A continuación, se presenta una descripción general de los pasos comunes para distribuir una aplicación Java:

**Compilación**: Asegúrese de compilar todas las clases Java en archivos de bytecode (.class) utilizando el compilador de Java (javac). Puede usar el comando javac para compilar las clases, como se mencionó en la respuesta anterior.

**Empaquetado:** Empaque las clases y recursos en un archivo JAR (Java ARchive). Un archivo JAR es una forma común de distribuir aplicaciones Java y puede contener todas las clases, recursos y metadatos. Para crear un archivo JAR, utilice el comando jar de la siguiente manera:

jar cf MiApp.jar -C bin .

Esto creará un archivo JAR llamado MiApp.jar que contiene todas las clases y recursos en el directorio bin.

**Manifiesto JAR:** Si la aplicación requiere configuración adicional, como indicar la clase principal para ejecutar, puede incluir un archivo MANIFEST.MF en el JAR. El MANIFEST.MF debe especificar la clase principal utilizando la entrada Main-Class. Por ejemplo:

Main-Class: miapp.ClasePrincipal

Esto le dice a Java cuál es la clase principal que debe ejecutarse cuando se inicia el JAR.

**Empaquetado de dependencias:** Si la aplicación depende de bibliotecas externas (por ejemplo, archivos JAR de bibliotecas de terceros), debe incluir estas bibliotecas en la distribución. Puede colocar estos archivos JAR en un directorio específico dentro de la distribución.

**Documentación y recursos adicionales**: Incluya cualquier documentación o recursos adicionales que sean necesarios para que los usuarios comprendan y utilicen la aplicación, como archivos README, licencias, archivos de configuración, etc.

**Creación de un instalador o paquete distribuible:** Si desea que la aplicación sea fácilmente instalable en sistemas operativos específicos, puede crear un instalador o un paquete distribuible para esos sistemas. Esto puede incluir la creación de archivos de instalación para Windows (como un archivo .exe o .msi) o paquetes para sistemas basados en Unix (como archivos .deb o .rpm).

**Distribución:** Distribuya la aplicación empaquetada y los recursos adicionales a los usuarios finales. Esto puede hacerse a través de descargas en línea, medios físicos (como CD o DVD) o cualquier otro método adecuado para la audiencia.

I**nstrucciones de instalación y uso:** Proporcione a los usuarios instrucciones claras sobre cómo instalar y utilizar la aplicación.

* Documentación

La documentación se refiere a la información escrita que describe el propósito, funcionamiento, uso y detalles técnicos de las clases, métodos, interfaces y otros elementos del código fuente de un programa Java. La documentación sirve como una guía para los desarrolladores que trabajan en el código y para aquellos que utilizan las clases y métodos en sus propios programas. La documentación es esencial para comprender y utilizar eficazmente las bibliotecas y componentes Java.

La documentación en Java se crea principalmente mediante comentarios especiales conocidos como "comentarios de documentación" o "JavaDoc comments". Estos comentarios comienzan con /\*\* y se pueden agregar justo antes de clases, métodos, campos y otros elementos del código fuente. Los comentarios de documentación siguen una convención específica y pueden incluir:

Descripción: Una descripción concisa del propósito y la funcionalidad del elemento.

Parámetros: Si se trata de un método, los comentarios de documentación pueden incluir una lista de parámetros, sus tipos y descripciones.

Valor de retorno: Para métodos, se puede especificar qué valor retorna el método y una descripción de su significado.

Excepciones lanzadas: Si un método lanza excepciones, estas se pueden listar junto con una descripción de cuándo y por qué se lanzan.

Ejemplos de uso: Se pueden proporcionar ejemplos de cómo utilizar el elemento en el código.

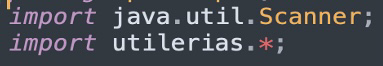
# **Desarrollo**

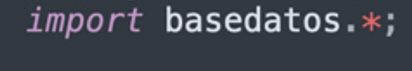
Generar un sistema de autenticación en Java, dividido en 3 paquetes de la siguiente manera:

* Principal: Clases que soliciten los datos al usuario para realizar una autenticación.
* Base datos: Clases que tengan a los usuarios registrados que serán válidos para el sistema.
* Utilerías: Clases utilizadas para realizar la autenticación al sistema

Biblioteca usada para leer valores



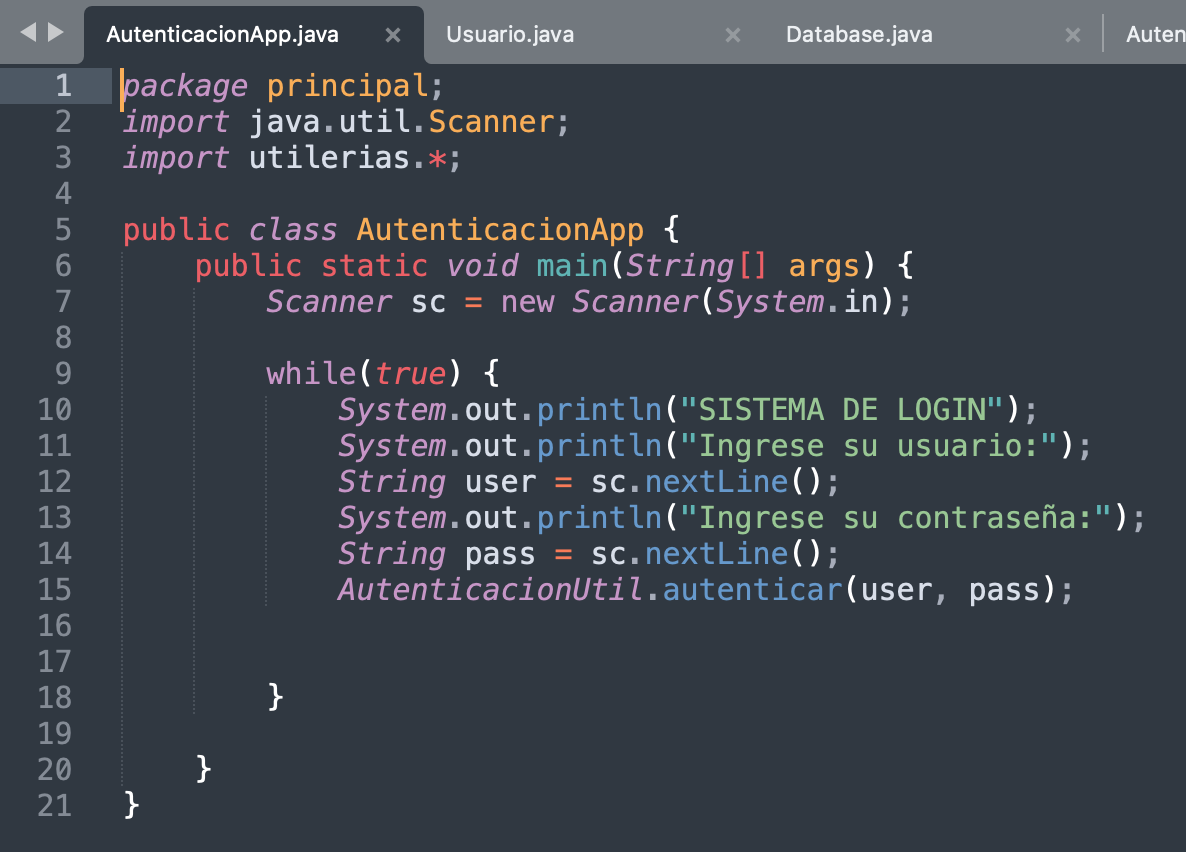




**Capturas del código del programa**

Paquete 1.

Principal: Clases que soliciten los datos al usuario para realizar una autenticación

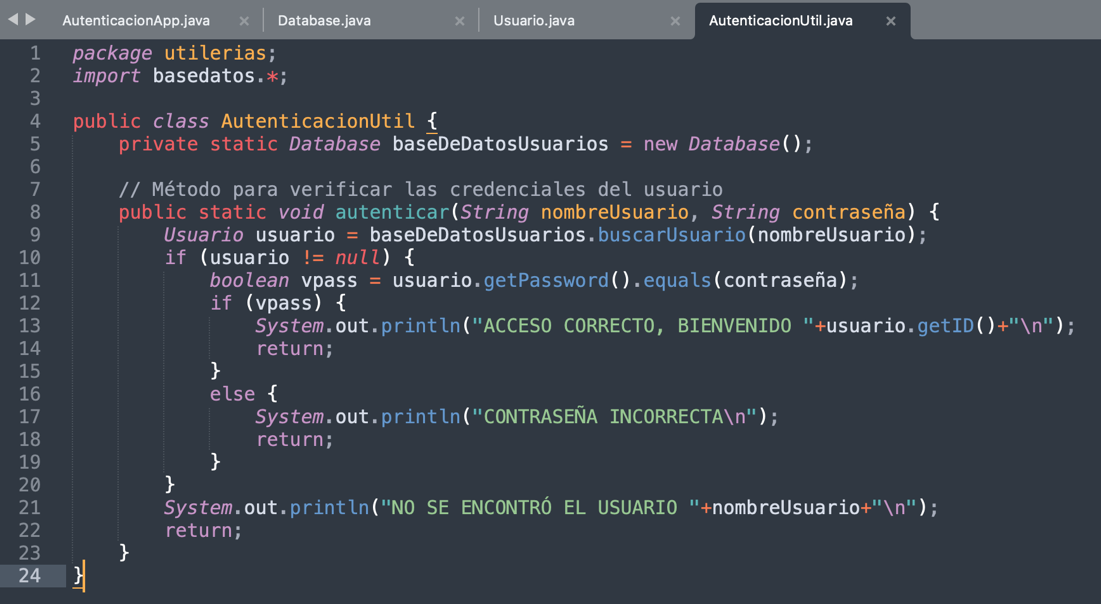


Paquete 2.  
Basedatos: Clases que tengan a los usuarios registrados que serán válidos para el sistema

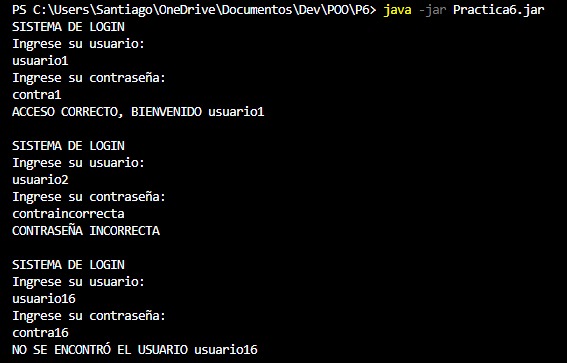




Paquete 3.   
Utilerias: Clases utilizadas para realizar la autenticación al sistema



Compilación del código (ejemplo)



# **Conclusiones**

José Gerardo Barrera Treviño: José Gerardo:

En Java, los paquetes desempeñan un papel fundamental en la organización y estructura del código. Permiten la agrupación lógica de clases y recursos relacionados en espacios de nombres separados, evitando así conflictos de nombres y facilitando la administración del código. Además, esta estructura se refleja en la jerarquía de directorios del sistema de archivos, lo que simplifica la localización y el mantenimiento de los componentes del proyecto.

La importación de paquetes es esencial para acceder a clases y miembros de otros paquetes en el código Java. Mediante la declaración "import", se permite al compilador y al intérprete de Java encontrar las clases necesarias sin requerir el uso del nombre completo de la clase en cada referencia. Esto mejora la legibilidad y la mantenibilidad del código, ya que aclara el origen de las clases utilizadas.

En conjunto, la utilización eficiente de paquetes, importaciones de paquetes, documentación y paquetes propios contribuye significativamente a un desarrollo de software ordenado, comprensible y colaborativo en el entorno de Java. Estos conceptos son esenciales para garantizar la calidad, escalabilidad y mantenibilidad de los proyectos de software en este lenguaje de programación.

Muñoz San Agustin Victoria Monserrat:

Intenté hacer el código por mi cuenta, pero me costó bastante a la hora de implementar las subclases en la clase principal ya que me emocionaba que no era visible cuando las tenía declaradas como públicas, pero al ver videos y leer texto pude reconocer mis errores. Creo que esta práctica hace mucho uso de fundamentos de programación ya que para poder crear este programa debes generar diferentes folders que guarden los archivos .java y al final dar un archivo que englobe los demás, creo que es sencillo si logras comprender el funcionamiento public y private ya que son muy importantes en este desarrollo, además de que hace alusión a temas visto anteriormente durante el curso.

Algo también a destacar, son los import y package, si entendías eso lo demás estaba facilísimo ya que era pura implementación de estas dos funciones, además de que en la compilación por medio de consola era necesario conocer cómo se podían compilar juntos sin necesidad de acceder a uno por uno.

Leocadio Chávez Rodrigo:

Para la creación de cada uno de los paquetes lo que hicimos fue una implementación parecida a las que ya habíamos utilizado en prácticas anteriores así que esto se llevó a cabo sin ningún problema y de manera relativamente sencilla. El uso de paqueterías fue una de las cosas que más nos llevó tiempo, ya que teníamos que tener una correcta organización del código para que no hubiera problemas y fuera más fácil la compilación de estas ya que la creación de un archivo JAR requiere una organización precisa de los paquetes y una especificación adecuada en el archivo MANIFEST.MF para tener una correcta ejecución y así poder lograr el objetivo de la práctica.

Olvera Bravo Cynthia Carolina:

En esta práctica hicimos uso de paquetes, incluyéndolos con la palabra “import” y para poder implementarlos de manera correcta fue esencial ordenar el código de manera correcta y a mi parecer fue una de las cosas que se me hizo más difícil de lograr pues me pareció un reto el imaginar el orden de toda la estructura. Una vez realizada de manera correcta la estructura el crear el archivo JAR para su ejecución fue más sencillo pues ya teníamos el orden de los paquetes que íbamos a utilizar y simplemente recordamos el comando para su ejecución que era tan difícil de aprender una vez que lo entiendes. Entonces así comprendí que la implementación de paquetes hace más fácil su exportación, pero para ello la organización es parte esencial del proceso. Además de paquetes implementamos conceptos que ya habíamos utilizado anteriormente sin problema alguno.

# Velasco García Santiago:

Durante el desarrollo de esta práctica, aprendimos la importancia de la organización cuidadosa de la estructura de directorios y paquetes en un proyecto Java al crear y ejecutar archivos JAR. También observamos que la especificación precisa de la clase principal en el archivo MANIFEST.MF es esencial para que el archivo JAR se ejecute correctamente. La práctica nos recordó la importancia de comprender las opciones y argumentos de las herramientas que utilizamos, así como la necesidad de depurar y solucionar problemas cuando se encuentran desafíos en el proceso. Tener bien organizada nuestra estructura de directorios nos permite simplificar la creación de archivos JAR y garantizar una gestión más eficiente del proyecto.

# Referencias

s.a. (s.f.). Paquetes en Java – Package e import. Edu4JAVA. Recuperado el 1 de octubre de 2023 de <http://www.edu4java.com/es/java/paquetes-package-import.html>

Wang, P. S. (2000). Java con Programación orientada a Objetos y aplicaciones en la www. International Thomson.

Rodriguez, A. (s.f). Documentar proyectos Java con Javadoc. Comentarios, símbolos , tags (deprecated, param, etc). aprende a programar.com. Recuperado el 1 de octubre de 2023 de <https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=646:documentar-proyectos-java-con-javadoc-comentarios-simbolos-tags-depre%20cated-param-etc-cu00680b&catid=68&Itemid=188>