EXPLICACIÓN DE LOS EJERCICIOS:

1-En este ejercicio creamos una clase genérica Caja<T>, al ser de tipo genérica estamos diciendo que puede trabajar con cualquier tipo de dato. Dentro de esta clase declaramos un atributo privado llamado contenido, que es del tipo genérico T y al declararlo como privado estoy diciendo que solo se pueden acceder directamente a este atributo desde dentro de esta clase . Este atributo es donde se almacena el objeto que se quiere guardar dentro de la caja. La clase tiene un constructor que recibe un parámetro de tipo T y lo asigna al atributo contenido. De esta manera, al crear una instancia de caja se inicializa con un objeto determinado. También se incluye el método getContenido() que devuelve el objeto almacenado en la caja setContenido() que permite reemplazar el objeto almacenado por otro.
Luego creamos la clase Libro con el atributo privado título y autor, que al declararlos como privados solo se puede acceder directamente a estos atributos desde esta clase. Dentro de la clase tenemos un constructor que se utiliza para crear nuevos objetos de tipo LIbro. Este constructor recibe dos parámetros, un String para el título y otro para el autor. Luego declaramos un método toString() que sirve para definir como se mostrará un objeto Libro en forma de texto.
En la clase main se pone a prueba la clase genérica Caja<T> utilizando tres tipos de datos diferentes: String, Integer y la clase Libro. Esto permite demostrar como los genéricos permiten reutilizar una misma estructura para almacenar distintos tipos de objetos sin necesidad de duplicar el código. Primero creamos una caja de tipo String que almacene un texto, imprimimos su contenido a través del getContenido(), luego actualizamos su contenido con otro texto a través del setContenido() y se imprime nuevamente su contenido actualizado. En segundo lugar creamos una caja de tipo Integer que almacene un número, imprimimos su contenido a través del getContenido(), luego actualizamos su contenido con otro número a través del setContenido() y se imprime nuevamente su contenido actualizado. En tercer lugar se utiliza una clase Libro para crear un objeto con un título y el autor, guardamos ese libro en Caja<Libro> e imprimimos el contenido de la caja gracias al método toString() del Libro que muestra su título y autor, reemplazamos el contenido con un nuevo libro a través del setContenido() y mostramos el contenido actualizado.

2-En este ejercicio creamos una clase Persona con el atributo privado nombre y edad, que al declararlos como privados solo se puede acceder directamente a estos atributos desde esta clase. Dentro de la clase tenemos un constructor que se utiliza para crear nuevos objetos de tipo Persona. Este constructor recibe dos parámetros, un String para el nombre y un int para la edad. Luego declaramos un método toString() que sirve para definir como se mostrará un objeto Persona en forma de texto.
Luego creamos un método genérico gracias al uso de <T>, lo que significa que puede trabajar con cualquier tipo de dato. Recibe un parámetro arr, que es un arreglo de tipo T[] (puede ser de enteros, cadenas, objetos, etc.). Dentro de este método creamos un bucle for each para recorrer el arreglo e imprimimos cada uno de sus elementos.
Dentro de la clase main probamos el método genérico imprimirArray con tres tipos de arreglos: Integer,String y la clase Persona. Primero creamos un arreglo de objetos Integer y llamamos al método imprimirArray para mostrar cada número. En segundo lugar creamos un arreglo de objetos String y llamamos al método imprimirArray para mostrar cada cadena. En tercer lugar se crean tres objetos persona con nombre y edad, se agrupan en un arreglo y por último llamamos al método imprimirArray para mostrar cada persona.

3-En este ejercicio creamos una clase genérica Contador<T> , dentro de esta clase creamos el método contar que recibe dos parámetros un arreglo genérico de elementos de tipo T y el valor que queremos buscar dentro del arreglo. El método devuelve un entero int que representa la cantidad de veces que aparece valorBuscado en el arreglo. Dentro de este método verificamos si el arreglo recibido es null, si lo es, lanzamos una excepción personalizada ArregloNuloException con un mensaje de error. Esto evita que el programa se rompa al intentar recorrer un arreglo que no existe, usamos throw en Java para lanzar una excepción de forma manual cuando ocurre una situación inesperada o no válida en el programa. Luego inicializamos la variable contador que se usará para contar la cantidad de veces que aparece el valorBuscado. Utilizamos un bucle for-each para recorrer los elementos del arreglo, si tanto el valorBuscado como el elemento son null se cuenta como coincidencia(esto evita errores por llamar equals() sobre null), si esto ocurre se incrementa el contador. Si valorBuscado no es null, se utiliza equals() para comparar los objetos y si esto ocurre se incrementa el contador. Por último retornamos la cantidad de coincidencias encontradas.
Luego creamos una clase ArregloNuloException que es una excepción personalizada y que extiende de RuntimeException. Dentro de esta clase creamos su constructor que permite crear una nueva excepción pasando un mensaje personalizado como texto y dentro de este llamamos al constructor de la clase padre(RuntimeException) a través del super y le pasa el mensaje. De esta manera, ese mensaje se puede recuperar más adelante con e.getMessage() si se atrapa la excepción.
Dentro de la clase main creamos un objeto Contador que va a trabajar con elementos de tipo String, definimos un arreglo cadena con cuatro elementos, luego se llama al método contar pasando el arreglo palabras y el valor "hola" como parámetro, el método contar devuelve la cantidad de veces que aparece "hola" y mostramos por pantalla la cantidad de veces que aparece esta palabra. Después creamos un objeto Contador que va a trabajar con elementos de tipo Integer, definimos un arreglo números con varios valores, luego se llama al método contar pasando el arreglo números y el valor 5 como parámetro, el método contar devuelve la cantidad de veces que aparece el 5 y mostramos por pantalla la cantidad de veces que aparece este número. Por último se prueba el comportamiento cuando el arreglo es null, llamamos al método contar pasando un arreglo nulo, como esta programada para lanzar una excepción personalizada en ese caso (ArregloNuloException), se captura con un bloque try-catch y por último Imprimimos el mensaje de error personalizado utilizando el getMessage().

4-En este ejercicio creamos la clase División, dentro de esta clase declaramos el atributo privado numerador y denominador. Al declararlo como privado solo se pueden acceder directamente desde dentro de la clase. Creamos el constructor de la clase División, que se utilizará para instanciar un nuevo objeto con valores iniciales para numerador y denominador. Creamos el método calcular que devuelve un número entero y verificamos si el denominador es cero. Si lo es, lanza una excepción aritmética (ArithmeticException) con un mensaje personalizado. Esto evita que el programa se detenga bruscamente por error de división por cero. Usamos throw para lanzar una excepción de forma manual cuando ocurre una situación inesperada o no válida en el programa. Si el denominador es distinto de cero se devuelve el resultado de la división.
Dentro de la clase main importamos la clase Scanner, que permite leer datos desde el teclado, creamos un objeto Scanner para leer datos ingresados por el usuario, le pedimos al usuario que ingrese dos números, uno para el numerador y otro para el denominador. Se

usa un método llamado leerNumero para asegurarse de que el usuario realmente escriba un número válido y creamos un objeto de la clase llamado División. Utilizamos un bloque try que intenta realizar la división, si el denominador es distinto de 0 muestra el resultado. Si el denominador es cero ocurre una excepción aritmética (ArithmeticException), que es capturada en el catch y mostramos el mensaje de error personalizado a través del getMessage(). Por último cerramos el Scanner una vez que lo terminamos de usar a través del método close().
Luego creamos el método static leerNumero, dentro de este método mostramos el mensaje para que el usuario sepa qué debe ingresar. Colocamos un while que verifica si lo que ingresó el usuario no es un entero. Si escribe letras o símbolos, lo rechaza con el mensaje "Entrada inválida". Por último devuelve el número ingresado.

5-En este ejercicio creamos la clase ValidarNombres e importamos la clase FileWriter que permite escribir textos en un archivos y la clase IOException que maneja errores de entrada/salida. Dentro de esta clase creamos el método static validarNombres que recibe como parámetros un arreglo de nombres. Dentro de este método utilizamos un bucle for each para recorrer los elementos del arreglo. Creamos un bloque try y dentro de este bloque verificamos si el nombre tiene 3 letras o menos, si se cumple esta condición se lanza una excepción personalizada con ese nombre. Usamos throw para lanzar una excepción de forma manual cuando ocurre una situación inesperada o no válida en el programa. Si el nombre tiene más de 3 letras, lo considera válido y lo muestra por consola. Si se lanza la excepción se ejecuta el bloque catch, dentro de este bloque se muestra el mensaje personalizado de error a través del getMessage() y llamamos al método registrarErrorEnLog() para guardar el error en el archivo log.txt. Creamos el método static registrarErrorEnLog que reciba un mensaje como parámetro, este método registra mensajes de error en un archivo de texto. Creamos un bloque try que usa FileWriter en modo append (true), para no sobrescribir el archivo, dentro del bloque try se escribe el mensaje recibido.
Colocamos un bloque catch que captura cualquier error de entrada/salida que ocurra cuando el programa intenta escribir en el archivo, si ocurre un error al escribir en el archivo lo informa por consola.
Luego creamos la clase NombreInvalidoException que es una excepción personalizada y que extiende de Exception. Creamos el constructor de la clase que permite crear una nueva excepción pasando un mensaje personalizado como texto y llamamos al constructor de la clase padre(Exception) a través del super y le pasa el mensaje. De esta manera, ese mensaje se puede recuperar más adelante con e.getMessage() si se atrapa la excepción.
Dentro de la clase main se declara un arreglo String que contiene una lista de nombres, luego llamamos al método estático validarNombres() de la clase ValidarNombres para verificar cada nombre del arreglo y si algún nombre tiene 3 letras o menos, se lanza una excepción personalizada y se registra en un archivo de log.

6-En este ejercicio creamos la clase InsertionSort, dentro de esta clase creamos un método estático que recibe un arreglo de double y devuelve un int que representa la cantidad de iteraciones realizadas para ordenar. Dentro de este método declaramos una variable iteraciones que suma una unidad cada vez que se hace una comparación y desplazamiento. Luego creamos un bucle for que comienza desde el segundo elemento(índice 1) y recorra el arreglo. La idea es insertar cada elemento en la parte izquierda ya ordenada. Una vez realizado esto guardamos el valor actual en temp y comenzamos a comparar con los valores a la izquierda (posición j), utilizamos un -1 que indica que la parte izquierda es la

debemos comparar. Luego creamos un while que compara si el elemento anterior (arr[j]) es mayor que temp y contamos las iteraciones cada vez que se entra al while. Si se cumple la condición del while desplazamos una posición a la derecha para hacer espacio, el j– es utilizado para que el algoritmo siga comparando con los elementos de la izquierda. Fuera de este while colocamos temp en la posición correcta dentro de la parte ya ordenada. Y por último retornamos el número de movimientos realizados en el ordenamiento.
Luego creamos la clase seleccionSort ,dentro de esta clase creamos un método estático que recibe un arreglo de double y devuelve un int que representa la cantidad de iteraciones realizadas para ordenar. Dentro de este método declaramos una variable iteraciones que suma una unidad cada vez que se hace una comparación y desplazamiento. Creamos un bucle for que recorre cada posición del arreglo menos la última (porque ya estará ordenada cuando termine), por eso se coloca el -1, dentro del bucle for declaramos una variable min que guarda el índice del valor mínimo encontrado, que inicialmente es el mismo i. Luego creamos otro for para la variable j, donde tenemos j = i + 1 que se usa para comparar con los elementos que están a la derecha del elemento actual (arr[i]), dentro de este for contamos la cantidad de comparaciones a través de iteraciones y creamos un if que compare el valor en arr[j] con el actual mínimo (arr[min]). Fuera de este for(de la variable j) guardamos el valor mínimo encontrado en una variable temporal, colocamos el valor en la posición actual (i), realizando el intercambio y ponemos el valor original de la posición actual en el lugar donde estaba el mínimo. Por último retornamos el número de movimientos realizados en el ordenamiento.
Después creamos la clase ShellSort, dentro de esta clase creamos un método estático que recibe un arreglo de double y devuelve un int que representa la cantidad de iteraciones realizadas para ordenar. Dentro de este método declaramos una variable iteraciones que suma una unidad cada vez que se hace una comparación y desplazamiento, obtenemos la longitud del arreglo. Luego creamos un bucle for y comenzamos con una brecha (gap) inicial que es la mitad del arreglo, el gap es la distancia entre elementos que se van a comparar y ordenar en cada pasada, en este for tenemos un int gap = n / 2 que nos dice que comenzamos con una brecha inicial igual a la mitad del tamaño del arreglo (n), un gap > 0 que nos dice que el bucle continúa mientras el gap sea mayor que 0. Cuando llega a 0, el arreglo ya debería estar ordenado, y un gap /= 2 que nos dice que en cada iteración del bucle externo, la brecha se divide por 2 y así se va reduciendo. Dentro de esto creamos otro for para recorrer el arreglo desde la posición "gap" hasta el final, en este for guardamos el valor actual en una variable temporal, usamos j para comparar hacia atrás con elementos separados por la brecha (gap). Creamos un while que mientras no salgamos del arreglo y el elemento a "gap" posiciones atrás sea mayor, dentro de este while contamos esta comparación como una iteración, desplazamos el elemento hacia adelante y retrocedemos "gap" posiciones para seguir comparando. Fuera de este while colocamos el valor temporal en su posición correcta dentro de la sublista ordenada. Por último retornamos el número de movimientos realizados en el ordenamiento.
Dentro de la clase main importamos la clase Arrays. Definimos un arreglo de números decimales con temperaturas a ordenar, luego hacemos tres copias del arreglo original para que cada algoritmo trabaje con los mismos datos originales, sin interferir entre sí. La función Arrays.copyOf()crea la copia del arreglo. Llamamos a los métodos ordenar() de cada clase (SeleccionSort, InsertionSort, ShellSort) pasando su copia del arreglo, cada método devuelve la cantidad de iteraciones realizadas, que se guarda en variables (iterSel, iterIns, iterShell). Después imprimimos el arreglo ordenado por el método de selección, inserción y shell, cada método con la cantidad de iteraciones que realizó.

7-En este ejercicio creamos la clase BusquedaBinaria en la que creamos el método estático que implementa la búsqueda binaria en un arreglo de enteros. Luego inicializamos los extremos del arreglo: izquierda y derecha, creamos un while que se ejecuta mientras el intervalo de búsqueda no sea vacío. Dentro del while calculamos la posición del medio para evitar desbordamiento, creamos un if y si el valor está en el medio, lo encontramos; si el valor buscado es mayor que el del medio, buscamos a la derecha; si el valor buscado es menor, buscamos a la izquierda. Por último si salimos del bucle sin encontrar el valor, retornamos -1.
Luego creamos la clase BusquedaSecuencial en la que creamos el método estático que implementa la búsqueda secuencial en un arreglo de enteros. Creamos un for y recorremos el arreglo desde el primer elemento hasta el último, si encontramos el valor buscado, devolvemos su posición (índice). Si terminamos de recorrer el arreglo y no encontramos el valor, retornamos -1.
Dentro de la clase main importamos la clase Scanner, que permite leer datos desde el teclado y creamos un objeto Scanner para leer datos ingresados por el usuario. Luego declaramos e inicializamos un arreglo ordenado de enteros, le pedimos al usuario que ingrese el número que desea buscar. Una vez ingresado el valor realizamos la búsqueda secuencial del valor en el arreglo creamos un if y evaluamos si la búsqueda secuencial encontró el valor, si lo encontró, mostramos la posición y si no lo encontró, mostramos el mensaje que no fue encontrado. Luego realizamos la búsqueda binaria del valor en el arreglo, creamos un if y evaluamos si la búsqueda binaria encontró el valor, si lo encontró, mostramos la posición y si no lo encontró, mostramos el mensaje que no fue encontrado. Por último cerramos el objeto Scanner a través del close().

8-En este ejercicio creamos la clase SumaDigitos, dentro de esta clase creamos el método estático que recibe un número entero y devuelve la suma de sus dígitos usando recursividad. Dentro de este método utilizamos un if que si el número es negativo, lo convertimos a positivo. Utilizamos otro if para el caso base que si el número es 0, la suma es 0. En la cual en una función recursiva, el caso base es la condición que detiene la recursión. Es el punto en el que la función deja de llamarse a sí misma. Si no hay un caso base, la función seguiría llamándose infinitamente, lo que provocaría un error de desbordamiento de pila. Por último tomamos el último dígito con n % 10 y llamamos recursivamente al método con el resto del número (n / 10).
Dentro de la clase main importamos la clase Scanner, que permite leer datos desde el teclado y creamos un objeto Scanner para leer datos ingresados por el usuario. Luego le pedimos al usuario que ingrese un número entero. Una vez ingresado el número llamamos al método recursivo para calcular la suma de los dígitos del número ingresado que a esto lo lo hacemos llamando a la clase SumaDigitos junto con su método sumaDigitos y mostramos el resultado de la suma de los digitos. Por último cerramos el Scanner una vez que lo terminamos de usar a través del método close().

9-En este ejercicio creamos la clase ContarVocales, dentro de esta clase creamos el método estático que cuenta la cantidad de vocales en una cadena usando recursividad. Utilizamos un if para el caso base, que nos dice que si el texto es nulo o está vacío, no hay vocales, retornamos 0. En la cual en una función recursiva, el caso base es la condición que detiene la recursión. Es el punto en el que la función deja de llamarse a sí misma. Si no hay

un caso base, la función seguiría llamándose infinitamente, lo que provocaría un error de desbordamiento de pila. Luego obtenemos el primer carácter de la cadena a través del charAt y lo convertimos a minúscula a través del toLowerCase. Luego verificamos si el primer carácter es una vocal. Si lo es, cuenta vale 1; si no, vale 0. Por último sumamos la cuenta actual con la cantidad de vocales en el resto de la cadena (llamada recursiva).
Dentro de la clase main importamos la clase Scanner, que permite leer datos desde el teclado y creamos un objeto Scanner para leer datos ingresados por el usuario. Luego le pedimos al usuario que ingrese una cadena de texto Una vez ingresado el número llamamos al método recursivo para contar las vocales en la cadena ingresada que a esto lo lo hacemos llamando a la clase ContarVocales junto con su método contarVocales y mostramos el resultado de la cantidad de vocales obtenidas de la cadena. Por último cerramos el Scanner una vez que lo terminamos de usar a través del método close().

10-En este ejercicio creamos la clase TorresDeHanoi, dentro de esta clase creamos el método recursivo que resuelve las Torres de Hanoi para n discos, en la que pasamos como parámetro a n (número de discos a mover), origen (torre de donde se mueven los discos), destino (torre a donde se quieren mover los discos) y auxiliar(torre auxiliar que se usa como soporte en los movimientos). Dentro de este método utilizamos un if para el caso base, en la cual nos dice que si solo hay un disco, simplemente se mueve de origen a destino. En la cual en una función recursiva, el caso base es la condición que detiene la recursión. Es el punto en el que la función deja de llamarse a sí misma. Si no hay un caso base, la función seguiría llamándose infinitamente, lo que provocaría un error de desbordamiento de pila. El paso uno nos dice que debemos mover los n-1 discos superiores desde origen a la torre auxiliar,usando la torre destino como auxiliar. El segundo paso nos dice que debemos mover el disco más grande (el n-ésimo) desde origen a destino. El tercer paso nos dice que debemos mover los n-1 discos que quedaron en la torre auxiliar hacia la torre destino,usando la torre origen como auxiliar.
Dentro de la clase main definimos la cantidad de discos a mover, mostramos un mensaje inicial con los movimientos a resolver. Por último llamamos al método recursivo para resolver el problema en la que pasamos n (cantidad de discos) , 'A'(torre origen), 'C' (torre destino) y 'B'(torre auxiliar).