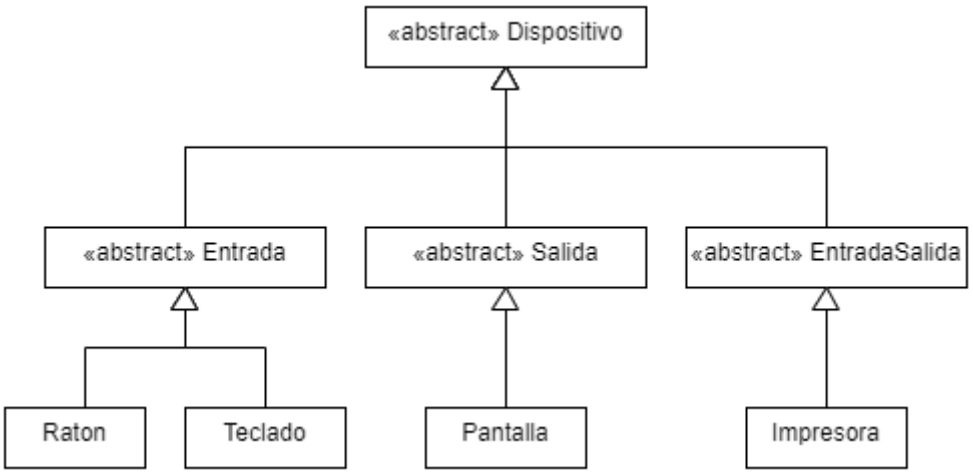
CUESTIONES Y EJERCICIOS

CONVERSIONES DE REFERENCIAS

1. Partimos de la siguiente jerarquía de clases:



Y en un método *main* tecleamos el siguiente código:

```
Raton r = new Raton();
Teclado t = new Teclado();
Pantalla p = new Pantalla();
Impresora i = new Impresora();
Entrada e = t;
Dispositivo d = p;
```

Indica si las siguientes instrucciones son correctas o no y, en caso de no serlo, indica cómo deberíamos corregirlas, siempre que sea posible:

a. Entrada ent = r;
b. Dispositivo dis = e;
c. Salida sal = t;
d. Impresora imp = (Impresora) t;
e. EntradaSalida es = i;
f. Salida s = (Salida) d;
g. Pantalla pant = d;
h. Teclado tec = e;
i. Raton rat = (Raton) t;

TIPOS DE COLECCIONES DE DATOS

2. Rellena la siguiente tabla con la estructura de datos (lista, pila, cola, conjunto o diccionario) que consideras más oportuna para cada caso. Piensa en los argumentos en los que te basas para dar tu respuesta:

Situación	Estructura de datos
Tienes que almacenar una colección de matrículas de vehículos que están en "busca y captura" por la policía. La idea es que se pueda buscar rápidamente si una matrícula está contenida o no en la colección.	
Tienes que almacenar una colección de canciones de Spotify para que sean reproducidas en orden.	
Tienes que almacenar una colección de nombres de usuario y respectivas contraseñas, de forma que puedas acceder rápidamente a la contraseña almacenada dado un nombre de usuario concreto.	
Tienes que almacenar las peticiones de páginas web que recibe tu servidor web para que no se pierda ninguna solicitud y poder atenderlas en estricto orden de llegada.	
Tienes que almacenar las pulsaciones de teclado que realiza un usuario en tu aplicación para poder volcarlas a un cuadro de texto en la pantalla	
Tienes que almacenar el uso de los carritos de un supermercado sabiendo que éstos se encajan unos en otros y que solo se puede sacar el último que se introdujo en la colección	
Tienes que almacenar los eventos culturales de una ciudad en orden cronológico	

LISTAS

3. Vamos a crear una clase llamada *PruebaListas* con un método main que realice las siguientes acciones:
   - Creamos un objeto de tipo ArrayList y se guarda en una referencia de tipo List llamada *meses*.
   - A continuación, llenamos la lista con los nombres de los 12 meses del año empezando desde "Enero".
   - Ahora toca hacerle consultas a la lista e imprimir "las respuestas". Tenemos que averiguar si está vacía o no. Cuántos elementos contiene y si contiene el mes de "Henero"
   - Acto seguido vamos a recorrer la lista con un iterador e imprimir en pantalla cada uno de los valores.
   - Después vamos a ordenar la lista y volver a imprimir los valores de la lista ordenada utilizando un iterador.
   - Por último, vamos a vaciar la lista y volverla a imprimir con utilizando un iterador.

4. Vamos a crear una clase llamada *NombresDePerros* con un método main que realice las siguientes acciones:
   - Se crea una lista de tipo ArrayList y se guarda en una referencia de tipo List llamada *nombres*.
   - Guardaremos en la lista los siguientes valores: “Toby”, “Rocky”, “Max”, “Pancho”, “Coco”, “Chispa”
   - A continuación, escribiremos en pantalla: “Actualmente conozco X nombres para perros. Te los digo: ” y se imprimirán los nombres de la lista, uno debajo de otro, recorriéndola mediante un iterador.
   - Acto seguido, escribiremos “También te los puedo decir en orden alfabético: “. Entonces ordenamos la lista y la volvemos a recorrer usando un iterador.
   - Ahora nos interesa que el usuario nos de algunos nombres más. Para ello, le diremos al usuario que nos enseñe más nombres de perros y que cuando se canse añada el nombre “FIN” para terminar. Los nombres que vaya introduciendo los iremos añadiendo a la lista.
   - Una vez el usuario ha terminado de introducir nombre entonces ordenaremos la lista alfabéticamente y escribiremos: “Gracias. Ahora conozco X nombres de perros. Te los digo en orden: “ y se imprimirán los nombres recorriendo la lista con un iterador.

5. Vamos a crear una nueva clase llamada *ManipulandoCadenas* que tenga un método main que realice las siguientes acciones:
   - Se crea una lista de tipo ArrayList y se guarda en una referencia de tipo List llamada *cadenas.*
   - A continuación, cargaremos la lista con las siguientes cadenas: “Vertical”, “Horizontal”, “Izquierda”, “Derecha”, “Adelante”, “Atrás”, “Curvo”, “Recto”, “Arriba”, “Abajo”.
   - Ahora queremos imprimir las palabras de la lista en mayúsculas. OJO: no queremos cambiar las palabras de la lista, solo que al imprimirlas se vean en mayúsculas.
   - Acto seguido, vamos a ordenar alfabéticamente las palabras y volverlas a imprimir.
   - Ahora queremos mostrar solo las palabras que empiecen por la letra “A”.
   - Ahora queremos mostrar solo las palabras que terminen en la letra “o”.
   - Ahora queremos mostrar solo las palabras que contengan la letra “e”
   - Ahora queremos mostrar solo las palabras de 5 letras
   - Por último, mostramos solo las palabras con más de 5 letras y que empiecen por “A”

   Consulta los métodos que te hagan falta de la clase String en la documentación oficial de Java: https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

6. Creamos una clase *FiltrandoListas* con un método *main* en el que vamos a hacer las siguientes acciones:
   - Se crea una lista de tipo ArrayList y se guarda en una referencia de tipo List.
   - A continuación, cargaremos la lista con las siguientes cadenas: “Vertical”, “Horizontal”, “Izquierda”, “Derecha”, “Adelante”, “Atrás”, “Curvo”, “Recto”, “Arriba”, “Abajo”. (COPIALO DEL EJERCICIO ANTERIOR, ES LO MISMO).
   - Ordenamos la lista y la imprimimos utilizando su método toString()
   - Ahora tenemos que aprender algo nuevo. La clase Iterator tiene un método **remove()** que elimina de la lista el último elemento extraído mediante un **next()**. Con esta nueva utilidad vamos a manipular la lista del siguiente modo:
     - Recorre la lista y elimina las palabras que terminen en “o”. Acto seguido muestra el contenido de la lista.
     - Recorre de nuevo la lista y elimina las palabras que contengan la letra “t”. Muestra el contenido de la lista al terminar.
     - Recorre de nuevo la lista y elimina las palabras que contengan la letra “a” y que tengan más de 5 letras. Muestra el contenido al terminar… ¿Nos queda alguna palabra en la lista?

7. Creamos una clase Persona como la siguiente:

Persona *implementa* **Comparable**
- nombre: texto - dni: texto - edad: entero
+ Persona (nombre: texto, dni: texto, edad: entero) + getters y setters + toString() *// Overriden* + compareTo (Object obj)

   El criterio de orden lo vamos a establecer según la edad. Los más jóvenes primero y después los mayores.

   A continuación, crea otra clase llamada *PruebaOrdenPersonas* con un método *main* que cree una lista y 4 objetos Persona. Los almacene en la lista y muestre la lista antes de ordenarla y después de ordenarla.

   ¿Qué habría que cambiar si quisiéramos invertir el orden y que los más mayores aparecieran primero y después los más jóvenes?

8. Saca una copia del proyecto anterior y modifícalo para que se ordene por nombre en orden alfabético ascendente.

9. Saca una copia del proyecto anterior y modifícalo para que se ordene por nombre en orden ascendente y, en el caso de que dos personas tengan el mismo nombre, entonces ordenará por edad descendentemente.

10. Saca una copia del proyecto anterior y modifícalo para que se ordene por edad ascendente y, en el caso de que dos personas tengan la misma edad, entonces ordenará por nombre en orden alfabético ascendente.

CONJUNTOS

11. Vamos a crear una clase llamada PruebaConjuntos con un método main que realice las siguientes acciones:
    - Creamos un objeto de tipo HashSet y lo guardamos en una referencia de tipo Set llamada *candidatos*.
    - A continuación, guardamos en el conjunto los siguientes elementos. “Juan”, “Ana”, “Pedro”, “Rosa” y “María”.
    - Ahora toca hacerle consultas al conjunto e imprimir “las respuestas”. Tenemos que averiguar si está vacía o no. Cuántos elementos contiene y si contiene al candidato “Pedro”.
    - Acto seguido vamos a recorrer el conjunto con un iterador e imprimir en pantalla cada uno de los valores.
    - A continuación, vamos a crear otro HashSet y lo vamos a guardar en la referencia *seleccionados*.
    - Ahora recorremos el conjunto *candidatos* y añadimos los nombres que contengan la letra ‘a’ al conjunto *seleccionados.*
    - Por último, recorremos el conjunto *seleccionados* con un iterador y mostramos el contenido de cada uno de los elementos.

12. Vamos a crear una clase llamada *OperacionesConjuntos* con un método main que realice las siguientes acciones:
    - Creamos los siguientes conjuntos:
        - buscaPolicia = ["12345678A", "12345678B", "12345678C", "12345678D"]
        - buscaGuardia = ["12345678E", "12345678F", "12345678G", "12345678D"]
    - A continuación, tienes que crear y mostrar los siguientes conjuntos de resultados:
        - Todos los dnis buscados por la Policía o por la Guardia Civil
        - Todos los dnis buscados por la Policía y por la Guardia Civil
        - Todos los dnis buscados por la Policía, pero no por la Guardia Civil
        - Todos los dnis buscados por la Guardia Civil, pero no por la Policía

13. La empresa "Lotería Doña Paquita" vende un montón de números distintos para el sorteo de "El Gordo de Navidad". Debes realizar un programa que le pida al usuario los números que ha vendido la empresa (siendo el -1 la marca para terminar de introducir números). A continuación, le pedirá al usuario el número de "El Gordo". Finalmente se imprimirá un mensaje en la pantalla que te indique si la empresa ha vendido el número premiado.

14. Vamos a crear una clase llamada *ConjuntoTelefonos* con un método main que realice las siguientes acciones:
    - Se crean dos conjuntos llamados *permitidos* y *prohibidos.*
    - A continuación, se le dice al usuario que introduzca los números de teléfonos desde los que puede recibir llamadas y los vamos guardando en *permitidos*. Cuando el usuario introduzca un número igual o menor que cero, el programa dejará de pedirle números.
    - Acto seguido, vamos a hacer lo mismo con los números de teléfono desde los que el usuario no quiere recibir llamadas y los vamos guardando en *prohibidos*. Al igual que antes pararemos cuando el número introducido sea igual o menor que cero.
    - Por último, simularemos tres llamadas al usuario del siguiente modo:
        - Le pediremos al usuario que teclee el número de teléfono desde el que le llaman.
        - El programa lo buscará en los dos conjuntos y ofrecerá los siguientes posibles resultados:
            - "El xxxxxx está PERMITIDO, puedes atender la llamada"
            - "El xxxxxx está PROHIBIDO, cuelga"
            - "El xxxxxx no es un número conocido, haz lo que creas"

15. Crea una clase Persona con las siguientes propiedades: DNI, nombre, apellidos, edad, si está casada, si tiene trabajo, número de teléfono y dirección. A continuación, crea los getters y setters correspondientes, también el toString(). Crea, además, un constructor que incluya todas las propiedades.

    Ahora tienes que decidir qué propiedad/propiedades vas a usar para generar el criterio de igualdad y el código hash de la clase y, a continuación, genera los métodos correspondientes.

    Por último, crea otra clase llamada *PruebaSetPersona* que tenga un método *main* que realice las siguientes acciones:

    - Crea tres objetos persona. Dos de ellos deben satisfacer tu criterio de igualdad y otro objeto debe ser distinto.
    - Imprime el código hash de los tres objetos. Dos de ellos deberían coincidir y otro no.
    - Ahora crea un conjunto y guarda los tres objetos en él.
    - Por último, recorre el conjunto con un iterador e imprime cada uno de los objetos.

16. En una biblioteca se desea guardar información sobre los recursos bibliográficos que posee. Un recurso se caracteriza por las siguientes propiedades: ISBN (número entero único), título, lista de autores por orden de relevancia, el número de ejemplares que posee la biblioteca. Por otro lado, de los autores nos interesa almacenar su nombre, apellidos y nacionalidad.

    Debes crear las clases Autor y Recurso, dotándolas de constructores que admitan todas las propiedades. Sobrescribe también el método toString() de ambas.

    Además, tienes que establecer el criterio de igualdad y de código hash para ambas clases teniendo en cuenta que debes escoger propiedades inmutables. Para garantizar que las propiedades no cambien vamos, además, a tomar dos medidas:

    - Vamos a declararlas como *final*
    - Vamos a crear solo los getters de estas propiedades, no los setters. Para el resto de las propiedades “mutables”, creamos tanto el getter como el setter.

    Por último, debes crear una clase *PruebaRecursos* que tenga un método main que realice las siguientes acciones:

    - Crearemos un HashSet y lo almacenaremos en la referencia *recursos*.
    - Creamos el recurso de título “Los pilares de la tierra” del autor británico “Ken Follet” con ISBN 9788401328510 y del que la biblioteca ha comprado 10 ejemplares. Lo almacenamos en *recursos.*
    - Creamos el recurso de título “La catedral del mar” del autor español “Ildefonso Falcones” con ISBN 9788499088044 y del que la biblioteca ha comprado 8 ejemplares. Lo almacenamos en *recursos.*
    - Creamos el recurso de título “Los mejores cuentos para leer a media noche” de los autores Howard Phillips Lovecraft (estadounidense) y Alexandre Dumas (francés), con ISBN 9788417244712 y del que la biblioteca ha comprado 3 ejemplares. Lo almacenamos en *recursos.*
    - Por último, utilizamos un iterador para imprimir el contenido del conjunto *recursos.*

17. Vamos a crear una clase Cliente que tenga las siguientes propiedades:

Cliente
- idCliente: entero - nombre: texto - apellidos: texto
+ Cliente (idCliente: entero, nombre: texto, apellidos: texto) + getters y setters + toString() devuelve texto + equals(Object obj)

A continuación, vamos a crear una clase *PruebaMapaCliente* que cree un HashMap y lo almacene en una referencia llamada *clientes* de tipo Map.

Crearemos 3 clientes y los almacenaremos en el mapa utilizando como clave el idCliente y como valor el propio objeto.

Acto seguido, recorreremos el diccionario usando el conjunto de claves como iterador e imprimiremos cada uno de los clientes.

A continuación, vamos a borrar el primer cliente que guardamos en la estructura.

Además, vamos a actualizar el nombre de uno de los clientes. Para ello recuperaremos el objeto que queramos actualizar, le modificaremos el nombre con un método *set* y volveremos a guardarlo.

Por último, recorreremos de nuevo el diccionario usando el conjunto de claves como iterador e imprimiremos cada uno de los clientes.

NOTA: Fíjate que el idCliente es un int y que el método put(Object key, Object value) espera un Object como clave, sin embargo funciona perfectamente ¿qué está pasando?

18. Crea una clase *MapaHabitaciones* que cree un mapa llamado *estadoHabitaciones* que va a almacenar el estado de ocupación (true = ocupada, false = libre) de las habitaciones de un hotel. Realizaremos las siguientes acciones.
    - Partimos del siguiente estado de ocupación de las habitaciones del hotel: 101 -> false, 102 -> true, 103 -> true, 104 -> false, 105 -> true, 201 -> true, 202 -> false, 203 -> false, 204 -> true, 205 -> false.
    - Acto seguido presentaremos un menú al usuario en el que se le den las siguientes opciones:

      0 – Salir

      1 – Ver estado de ocupación

      2 – Ocupar una habitación

      3 – Liberar una habitación
    - La opción 1 imprimirá el estado de ocupación de las habitaciones del hotel con el siguiente formato: "Habitación XXX: ocupada/libre"
    - La opción 2 preguntará qué habitación se desea ocupar y la ocupará, si no estaba ocupada previamente, en ese caso debe emitir un mensaje de error.
    - La opción 3 preguntará qué habitación se desea liberar y la liberará, si no estaba libre previamente, en ese caso debe emitir un mensaje de error.

19. Suponiendo que tenemos creadas las siguientes clases: Cliente, Pedido, LineaPedido, Articulo indica cómo se escribirían las siguientes instrucciones utilizando la notación de tipo de datos genéricos:
    a. Declaración e inicialización de un conjunto de Pedido
    b. Declaración e inicialización de una lista de Articulo
    c. Declaración e inicialización de un diccionario que nos permita almacenar Articulo usando como clave de búsqueda el nombre del artículo.
    d. Declaración e inicialización de un iterador del conjunto declarado en el apartado a)
    e. Declaración e inicialización de un iterador de la lista declarada en el apartado b) y, además, las instrucciones para recorrerla.
    f. Declaración e inicialización de un iterador de las claves del diccionario declarado en el apartado c) y, además, las instrucciones para recorrerlo.

20. Crea una clase Vehículo caracterizada por: marca, modelo, matrícula, año de fabricación y precio. La clase debe definir los métodos: equals, hashCode y toString. A continuación, crea una clase Concesionario que caracteriza por un nombre, una dirección y un conjunto de vehículos utilizando la notación de tipos de datos genéricos. El concesionario debe ofrecer los siguientes métodos:

    - Un constructor que tenga como parámetros el nombre y la dirección.
    - Añadir un vehículo al concesionario.
    - Actualiza un vehículo.
    - Borrar un vehículo.
    - Muestra todos los vehículos del concesionario
    - Calcula la suma total de los precios de todos los vehículos del concesionario.

    Por último, crea una clase PruebaConcesionario que tenga un método *main* y que pruebe todos los métodos del concesionario.

21. Crea una clase capaz de albergar un tipo genéricos llamada Funda. Podremos guardar un objeto en la funda, sacarlo y preguntar si la funda está vacía o no.

    También crearemos una clase abstracta llamada Dispositivo que tendrá las siguientes propiedades: marca, modelo, tamaño (en pulgadas) y estado (encendido/apagado). Esta clase tendrá un constructor que reciba la marca, modelo y tamaño del dispositivo. Además, se debe sobrescribir el método toString();

    A continuación, vamos a crear dos clases llamadas *Smartphone* y *Tablet* que extienden de Dispositivo.

    Por último, crearemos una clase PruebaFunda que cree dos fundas y pruebe los métodos de la clase utilizando para ello dos objetos:

    - Un Smartphone de 6 pulgadas de la marca “Samsung” y modelo “Galaxy 8”
    - Una Tablet de 10 pulgadas de la marca “Huawei” y modelo “P10”

22. Crea una clase *Camion* que puede transportar una carga de tipo genérico. Podremos cargar el camión, descargarlo, preguntarle si está cargado o no y cuántos kilos de carga lleva.

    Para ellos vamos a crear una interfaz llamada *Pesable* que solo tendrá un método con la siguiente firma: *public double getPeso()*

    Después, crearemos dos clases llamadas *Ganado* y *Madera* que implementen la interfaz *Pesable* y que representarán dos tipos de carga de nuestro *Camion*. Ambas clases tendrán un constructor en el que se indique como parámetro el número de kilos que pesará la carga. Además, deben implementar el método *getPeso*.

    Por último, crearemos una clase *PruebaCamion* que cree dos camiones y pruebe los métodos de la clase utilizando para ello dos objetos: uno de Ganado de 2000 kilos y otro de Madera de 5000 kilos.