Ejercicios.

1. Composición de clases. A partir de las clases Vehiculo y Cliente debemos construir la clase VehiculoAlquilado.
   Vehiculo tiene los atributos matrícula, marca, modelo, tarifa y disponible. Todos String salvo tarifa que es double y disponible boolean. Métodos: constructor, getters y setters.
   Cliente tiene los atributos DNI, nombre, apellidos y teléfono, todos String. Métodos: constructor, getters, setters y al menos getNombreCompleto.
   La clase VehiculoAlquilado tiene como atributos un vehículo, un cliente, la fecha de alquiler (LocalDate) y el número de días de alquiler. Métodos: constructor, getters/setters y getImporteTotal (numero de días de alquiler por tarifa).

2. Composición de clases. Diseña una clase llamada Cliente con las siguientes características:
   - Atributos: DNI, nombre, y teléfono.
   - Métodos: constructor, getters y setters.

   Diseña también una clase llamada CuentaBancaria como se indica:
   - Atributos: cliente (clase Cliente), saldo (double) y número de cuenta (String).
   - Métodos: constructor, getters, setters y …
     - depositar. incrementa el saldo de la cuenta en la cantidad pasada como parámetro.
     - transferencia. Mover una cantidad a otra cuenta. Ambas, cantidad y cuenta, pasadas como parámetro. La operación sólo se llevará a cabo si hay saldo suficiente. Este método devolverá true/false según se pueda realizar o no la transferencia.

   Prueba las clases: crea clientes, cuentas, haz depósitos y transferencias.

3. Composición de clase. Descarga el proyecto y completa los métodos que faltan en la clase GestionAlquileres. Descarga desde aquí.
4. Composición de clases. Descarga el proyecto y completa los métodos que faltan en la clase ColeccionCuentas. Descarga desde aquí.

5. Composición de clases. Vamos a crear un conjunto de clases que permitan almacenar exámenes y las preguntas y respuestas de los mismos. De cada examen se guarda una descripción y un listado de 20 preguntas. De cada pregunta se guarda el enunciado, una lista de 4 posibles respuestas y el número de la opción correcta.

   Crea un array de 20 exámenes, llénalo de forma automática y muestra mediante un método la lista de sus preguntas y sus respuestas de un examen. Sobrecarga el método para que muestre lo mismo pero añadiendo el texto “Correcta” al lado de la respuesta que lo sea. Estudia la imagen adjunta.

```
Introduce número de examen entre 0 y 19. Teclea 9999 para salir: 2
Descripción: Examen2
1. Pregunta 1
   a. Pregunta 1 Respuesta1
   b. Pregunta 1 Respuesta2
   c. Pregunta 1 Respuesta3-- Correcta--
   d. Pregunta 1 Respuesta4
2. Pregunta 2
   a. Pregunta 2 Respuesta1
   b. Pregunta 2 Respuesta2
   c. Pregunta 2 Respuesta3-- Correcta--
   d. Pregunta 2 Respuesta4
```

6. Herencia. Construir el siguiente esquema de clases para una empresa que se dedica al alquiler de vehículos. Crea un paquete nuevo llamado vehiculos para las clases.
    a. Clase Vehiculo.
        i. Atributos: matricula, marca, modelo, tarifa, disponible. Tarifa es double y disponible boolean. Los demás String.
        ii. Constructor.
        iii. Getters y setters.
    b. Clase Turismo. Hereda de Vehiculo.
        i. Atributos propios: puertas y automático, que serán int y boolean respectivamente.
        ii. Constructor.
        iii. Getters y setters.
    c. Clase Furgoneta. Hereda de Vehiculo.
        i. Atributos propios: capacidad en litros y carga máxima, ambos enteros.
        ii. Constructor.
        iii. Getters y setters.

    Crea un array de 10 vehículos y almacena en él objetos de todas las clases creadas. Recorre el array mediante un bucle del tipo “for-each”. Si un elemento del array está vacío vale null.
    Muestra para cada objeto el nombre de la clase a la que pertenece (Turismo o Furgoneta) mediante instanceof y usa también toString.

7. Herencia. Desarrolla un programa Java que permita la gestión de una empresa agroalimentaria que trabaja con tres tipos de productos: productos frescos, productos refrigerados y productos congelados. **Todos los productos llevan una información común: idproducto, descripcion, fecha de caducidad y número de lote**.
    A su vez, cada tipo de producto lleva alguna información específica.
    a. Productos frescos: fecha de envasado y el país de origen.
    b. Productos refrigerados: código del organismo de supervisión alimentaria.
    c. Productos congelados: temperatura de congelación recomendada.

    Crear el código de las clases Java implementando una relación de herencia desde la superclase Producto hasta las subclases Fresco, Refrigerado y Congelado. Cada clase debe disponer de constructor, getters y setters y tener un método que permita mostrar la información del objeto.

Crear una clase con el método main donde se cree un objeto de cada tipo y se muestre los datos de cada uno de los objetos creados. **La clase Producto es abstracta, no podrán instanciarse objetos de dicha clase.**
Utiliza LocalDate para las fechas.

8. Herencia. Completa la clase ColeccionProductos del siguiente ejercicio, que está basado en la estructura de clases del ejercicio anterior. Añade los siguientes métodos.
    - modificarTemperaturaCongelacion. Recibe dos parámetros, la Id de un producto congelado y la temperatura de congelación. El método buscará el producto en la lista y actualizará el valor de esa temperatura. Devolverá true o false y la operación se llevó a cabo con éxito o no, respectivamente.
    - consultarProductosPorPaisOrigen. Recibe como parámetro un nombre de país y devolverá un array de los productos Frescos de ese país.
    - consultarProductosCaducados. Comprueba si la fecha de caducidad de cada producto es anterior a la de hoy y devuelve un array de productos en los que se da esta situación.
    - obtenerProductosDelTipo. Recibe como parámetro un tipo de producto (fresco, congelado o refrigerado) y devuelve un array con los productos de ese tipo.

    En el main crea un menú de opciones citadas y prográmalo. Descarga el original **aquí**.

    Pueden serte útiles los siguientes métodos de la clase DateTime/LocalDate:
    - isBeforeNow(), devuelve true si la fecha a la que se aplica es anterior a la actual.
    - isBefore(otraFecha), devuelve true si la fecha a la que se aplica es anterior a la pasada como parámetro.

9. Interfaces. En este ejercicio vamos a simular el funcionamiento de dos estructuras, una pila y una cola. Como sabéis las pilas son estructuras de tipo LIFO, el último elemento que entra, es el primero que sale. Las colas se ajustan el modelo FIFO, el primer elemento en entrar es el primero en salir.

Crearemos dos clases llamadas Pila y Cola, ambas clases deben tener las características siguientes:

- Servirán para guardar objetos Persona (id, nombre):
    - Persona [] pila;
    - Persona [] cola;
- Tendrán un array para guardar los objetos y una variable entera para indicar la posición de llenado.
- Constructor para indicar el tamaño.
- Métodos.
    - estaVacia(). Devuelve true o false, según la estructura esté vacía o no.
    - primero(). Devuelve el primer elemento de la estructura. El que se guardó en primer lugar.
    - extraer(). Extrae el elemento correspondiente, lo devuelve y deja un hueco para otro.
    - insertar(). Inserta un objeto nuevo en la estructura si es posible. Devuelve verdadero o falso según se pueda o no llevar a cabo.

- mostrar(). Muestra el contenido de la pila/cola
- **Para asegurarnos de que ambas clases implementan estos métodos, crearemos una interfaz con los mismos, por lo que las clases Pila y Cola deben implementar esa interfaz.**
- **Probaremos el ejercicio guardando objetos de la clase Persona (dni, nombre) en esas estructuras.**
- **En el constructor pasaremos el tamaño de la estructura**

10. Interfaces. El objetivo de este ejercicio es el de crear una clase llamada GestionNumeros que tiene como atributo un array de 10 valores enteros generados al azar mediante Random.

    Esta clase debe implementar la interfaz InterfazNumeros que incluye los métodos:
    a. mostrarNumeros. Método void que muestra uno a uno los valores del array.
    b. mostrarNumerosComoString. Método void que muestra el array con el formato [valor1, valor2, … , valorN]
    c. multiplicarPor(int numero). Método void que multiplica cada valor del array por el número pasado como parámetro. El array debe ver modificado su contenido.
    d. contarValoresMayoresQue(int numero). Devuelve cuántos valores del array superan al número pasado como parámetro.
    e. contarValoresEntre(int numero1, int numero2). Devuelve cuántos valores del array están entre los dos pasados como parámetro.
    f. compararCon(int [] otroArray). Devuelve la suma de los valores del array con mayor número de elementos.
    g. getArray(int [] otroArray). Retorna el array con mayor número de elementos, el atributo del objeto o el pasado como parámetro.

11. Interfaces. Diseña una clase llamada Matriz con las características siguientes. Esta clase debe implementar una interface que le obligue a desarrollar los métodos que se indican abajo.
    - Atributos:
        - matriz de valores enteros.
    - Métodos:
        - Constructor:
            - Parámetros: número de filas, número de columnas, valor mínimo y valor máximo. Todos enteros.
            - La función del constructor es la de instanciar o crear una matriz con las filas/columnas indicadas e inicializarla con valores al azar comprendidos entre entre ese mínimo y máximo.
        - imprimirMatriz. Muestra en pantalla los valores de la matriz. No devuelve nada.
        - getMaximo. Retorna el valor máximo de los contenidos en la matriz.
        - getMedia. Retorna la media de los valores de la matriz.

    Ejemplo de cómo obtener un valor al azar comprendido entre 5 y 9: rnd.nextInt(9-5+1)+5.

    Crea varios objetos de esta clase y prueba sus métodos.

12. Diseña las clases que se citan:

- Clase Trabajo.
    - Atributos:
        - id ,entero.
        - descripción, texto
        - salario, double.
    - Métodos.
        - Constructor, getters, setters y toString.
- Clase Persona:
    - Atributos:
        - dni, String
        - nombre, String
        - puesto, Trabajo
    - Métodos:
        - Constructor, getters y setters.
- Clase PersonaDao.
    - Atributos: Array de 10 elementos de objetos Persona.
    - Métodos:
        - insertarPersona(Persona p). Inserta la persona en el array.
        - buscarPersona(String dni). Busca una persona en el array, si la encuentra la devuelve, sino devuelve null. Para buscar usa el DNI pasado como parámetro.
        - getNumeroPersonas(). Devuelve la cantidad de personas del array.
        - mostrarDatos(). Muestra el contenido del array en pantalla.
        - getNPrimerasPersonas(int n). Devuelve un array con las “n” primeras personas del array ordenadas por DNI.
        - cambiarPuestoTrabajo(String dni, Trabajo trabajo). Este método cambiar el trabajo de la persona con ese DNI. Si puede hacerlo devuelve true, en caso contrario false.
        - ordenarDatos(). Ordena los datos del array por dni.