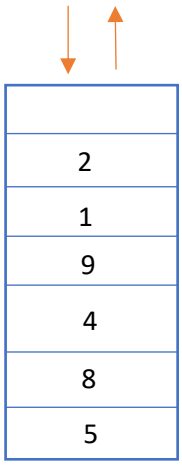
3ª Evaluación PRG 23/24 DAM
Nombre:

1) En Java existe una clase llamada Stack para poder usar una pila. Una pila es una estructura LIFO (last-input first-output). La pila la podemos ver como una caja donde vamos metiendo cosas pero se van sacando por arriba siempre.



En la pila anterior de números enteros primero se ha introducido el 5 luego el 8, 4, 9 ,1 ,2. Para sacarlos primero saldrá el 2, luego el 1, 9, 4, 8, 5. No es posible introducir un número en una posición concreta no sacar un número que no sea el último introducido. Imaginad una pila de palets en un almacén, no podemos quitar el tercero ni ponerlo en esa posición si encima hay más palets. Siempre entran por arriba y siempre sale el último introducido.
Se pide:

- Crear una clase Pila, usando un vector, que simule el funcionamiento de una pila que puede almacenar cualquier tipo de Objetos.
- La Pila implementará la siguiente interface:

```
public interface Stack{
      boolean push(Object o);// Inserta un elemento en la pila.
      Object pop();          // saca un elemento de la pila.
      boolean isEmpty();     // indica si la pila está vacía o no.
      int size();            //Devuelve el tamaño de la pila.
                             // elementos que hay
      String toString();     //muestra el contenido de la pila.
}
```

  Si la pila está llena push devuelve false
  Si la pila está vacía pop devuelve null
  La pila se inicializará a un número entero x

- Para probar el funcionamiento crear dos pilas de 5 elementos y rellenarla con 4 Objetos de distintos tipos (Coche, Zanahoria, Perro, Melón). Los objetos tendrán las propiedades y métodos que estiméis oportunos.
  - Mostrar el contenido de las pilas.
  - Comprobar si dos objetos son iguales usando el pop() para sacarlos.

Se valorará la claridad, POO, encapsulación, el uso adecuado de las estructuras. Cualquier cosa "extraña" comentadla. (3pto)

2) Un centro de estudios necesita un programa para llevar el control de las aulas de informática. Todas las aulas tienen un nombre, un puesto para el ordenador del profesor y puestos para los alumnos (no todas las aulas tienen el mismo número de ordenadores de alumnos pero sí el del profesor). De todas las aulas se quiere saber los ordenadores instalados en ellas. Un ordenador tiene las siguientes características: Procesador, memoria instalada, capacidad del disco, tipo de disco (mecánico/SSD) e identificador del PC. En un aula se puede desinstalar un PC para reparación y luego volver a ponerlo, por tanto, es posible que en un momento dado el aula no tenga todos los PC.
   Existen dos tipos de aulas, las normales que tienen solo las características anteriores y las Premium que además cuentan con proyector y aire acondicionado. De estos últimos solo se quiere saber si están operativos.
   Cuando un profesor detecta un fallo en un PC de un aula crea un ticket de avería. El centro desea tener un historial de todos los tickets. En un ticket debe de aparecer un identificador único, el aula afectada, el ordenador afectado (tipo ordenador) y si el ticket está resuelto o pendiente.
   El centro quiere saber si dos ordenadores, aunque sean de aulas distintas, son iguales. Dos ordenadores son iguales cuando todas sus características lo son a excepción del identificador.
   Por otro lado, se quiere mostrar toda la información de las aulas ordenada, de forma que primero saldrán las aulas normales y dentro de estás primero las que menos equipos tienen instalados, en el momento de la petición, luego las Premium y dentro de estás primero las que tienen menos equipos instalados, en el momento de la petición.

   Crear un programa en Java que resuelva la situación. Cread un main que demuestre el funcionamiento. No es necesario mucho código solo que quede claro que sabéis lo que estáis haciendo.
   Se valorará el uso de las estructuras adecuadas, su tipo, la encapsulación, polimorfismo, POO (5pto)

3) Dado el siguiente código indica **si compila o no**. **Razona tu respuesta**. En caso de que creas que sí, ¿qué veríamos por pantalla? **(se penalizarán respuestas sí/no)** (2pto)

```
    public abstract class Persona {
        private String nombre;
        private String apellidos;
        private int edad;
        Persona(String nombre,String apellidos,int edad)
        {
                        this.nombre=nombre;
                        this.apellidos=apellidos;
                        this.edad=edad;
        }
        public String toString(){
                return "nombre:"+nombre+", apellidos:"+apellidos+", edad:"+edad;
        }
    }
**********************************************************************
    public class Alumno extends Persona {
        private String ciclo;
        private int curso;
        Alumno(String nombre,String apellidos,int edad,String ciclo,int curso)
        {
                        super(nombre,apellidos,edad);
                        this.ciclo=ciclo;
                        this.curso=curso;
        }
        public int getCurso(){
                return curso;
        }
        public String toString(){
                return super.toString()+", ciclo:"+ciclo+", curso:"+curso;
        }
    }
**********************************************************************
    public class Gestion {
        public static void main (String args[]) {
                 Persona [] centro=new Persona[5];
                 centro[0]=new Alumno("pepe","gonzalez",23,"DAM",1);
                 for(int i=0;i<centro.length;i++)
                        System.out.println(centro[i].getCurso());

        }
    }
```

API

- Cuando hacemos un método en una clase no solo puedo operar con las propiedades de la misma, también con los parámetros que le llegan.

- Interface Comparable:

> public interface **Comparable<T>**
> **int compareTo(T o)**
> Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

- Método equals:

> **public boolean equals(Object obj)**
> Indicates whether some other object is "equal to" this one.

- Crear una interface:
  public interface *nombre*

-Crear clase abstracta
  public abstract class *nombre*

-Heredar e implementar:
  extends y ***implements***

-Una clase abstracta no necesita solucionar los métodos de una interface si esta tiene hijos que lo hacen.

-Si una clase implementa una interface y a su vez extiende una clase que también implementa la interface, esa clase no necesita implementar los métodos ya que los hereda. Lo que si puede es sobreescribirlos.

Algoritmo de la burbuja aplicado a un vector de enteros

```
for(int i = 1; i < vec.length; i++) {
        for(int j = 0; j < (vec.length - i); j++) {
                if(vec[j] > vec[j+1]){
                        aux = vec[j];
                        vec[j] = vec[j+1];
                        vec[j+1] = aux;
                }
        }
}
```