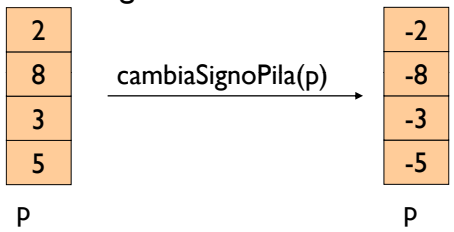
# Ejercicios Tema 7

Ejercicios Adaptados de Apuntes y Exámenes de EDA
Germán Moltó
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia



## Cambio de Signo Pila

1. Diseñar un método recursivo en una clase EjerciciosPila que, dada una Pila de *Integer*, la transforme de manera que tenga los mismos datos que la Pila original pero cambiados de signo.

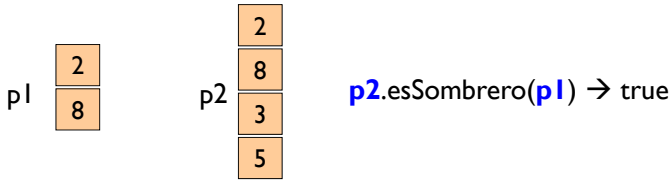



- public static void **cambiaSignoPila**(Pila<Integer> p)
- ¿Cómo se implementaría ese método para poder invocarlo sobre una instancia de Pila?



## Pila Sombrero

- Se dice que una Pila dada p es sombrero de otra si todos los datos de p aparecen en la otra en el mismo orden y ocupando las posiciones más próximas a su tope.
  - Una pila vacía es sombrero de cualquier otra pila.




• Se pide: Enriquézcase la interfaz Pila con un método recursivo para calcular si una pila dada es sombrero de la pila sobre la que se invoca el método

public boolean **esSombrero**(Pila p);

• Por simplicidad asumir que p2 tiene más elementos que p1



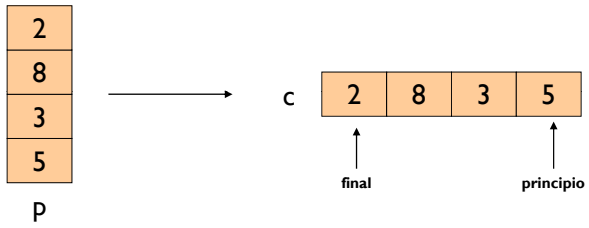
## Inversión de una Cola

- Diséñese un método que invierta recursivamente una Cola dada.


- El método se implementará en una clase diferente de aquella que implementa la interfaz Cola.
- ¿Cuál es el coste del algoritmo?

## Pila2Cola

- Se desea ampliar la funcionalidad de una Pila con una nueva operación que transforme una Pila en una Cola tal que su último dato sea el situado en el tope de la Pila.

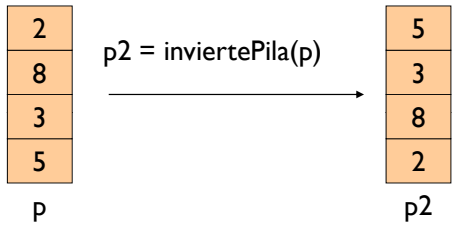



- Enriquézcase el interfaz Pila con una operación toCola que realice tal operación.



## Inversión de Pila

- Implementar un método **iterativo** que, a partir de una Pila dada, obtenga una nueva Pila con los elementos situados en orden inverso. Se permite la modificación de la pila original.




- Asumiendo que la implementación se realiza en la clase EjerciciosPila, ¿Cómo se invocaría al método desde una supuesta clase TestEjerciciosPila del mismo paquete para invertir una pila de 10 Integer?



## Cálculo de Talla de Pila

- Enriquecer el interfaz Pila con un nuevo método para calcular la talla de la Pila.
- El nuevo método no debe destruir la Pila.



## Ampliando Funcionalidad de ListaConPI

- Enriquece el modelo ListaConPI con las siguientes operaciones:
  - void **localiza**(E x): Sitúa el punto de interés sobre la primera aparición del elemento buscado.
  - int **talla**(): Devuelve el número de elementos de la ListaConPI.
  - void **vaciar**(): Elimina todos los elementos de la ListaConPI.

## Lista con Punto de Interés: Anterior (1/2)

- Dada la siguiente definición de ListaConPI …

```
package modelos;
public interface ListaConPI<E> {
        void insertar(E x);
        void borrar() throws ElementoNoEncontrado;
        void inicio();
        void fin();
        void siguiente() throws ElementoNoEncontrado;
        E recuperar() throws ElementoNoEncontrado;
        boolean esFin();
        boolean esVacia();
    }
```

- … donde la excepción ElementoNoEncontrado se produce al invocar una operación con el PI situado al final de la lista.



## Lista con Punto de Interés: Anterior (2/2)

1. Añadir al interfaz ListaConPI aquellos métodos que se consideren necesarios para permitir iterar **descendentemente** sobre una Lista con Punto de Interés, indicando brevemente qué hacen.

2. Utilizando únicamente las operaciones del nuevo interfaz, escribir un método toString de una supuesta clase que implementa el interfaz ListaConPI para que muestre en orden inverso al de su inserción los elementos de la lista.



## Lista con Punto de Interés: eliminarTodos

- Dada la interfaz Java ListaConPI tal y como ha sido definido en teoría y la clase:

```
public class ListaPlus<E> {
        private ListaConPI<E> lpi;
        public ListaPlus(){ lpi = new LEGListaConPI<E>();}
        public void eliminarTodos(E x)  throws
        ElementoNoEncontrado{...}
...
}
```

- Diseñar el método **eliminarTodos** que borra todas las apariciones de un cierto dato x de una Lista.
  - Si x no es un dato de la lista, lanza la excepción.



## Lista con Punto de Interés: Lista Ordenada

```
public interface ListaOrdenada<E extends Comparable<E>>{
    void insertarOrdenado(E x);
    E borrar(E x);
    boolean esVacia();
}
public class ListaOrdenadaPI<E extends Comparable<E>> implements
    ListaOrdenada<E>{
    private ListaConPI<E> lpi;
    public ListaOrdenadaPI(){ lpi = new LEGListaConPI<E>();}
    public void insertarOrdenado(E x) { ... }
    public E borrar(E x) { ... }
    public boolean esVacia(){ ... }
}
```

- Se pide implementar el método insertarOrdenado de la clase ListaOrdenadaPI

## Desapilar Todos Rango

- Se desea añadir al modelo de Pila un nuevo método desapilarTodosRango que elimina todos los elementos de la Pila que pertenecen a un cierto intervalo [x,y]. Obviamente, los elementos deberán ser de tipo Comparable.
- Para ello se va a extender el modelo con una nueva interfaz llamada PilaExtendida
- Dado que se dispone de la implementación del modelo con la clase *ArrayPila* se desea también extender esta clase añadiendo la nueva operación. Se pide:
  1. Implementar la Interfaz PilaExtendida
  2. Implementar la clase *ArrayPilaExtendida* utilizando única y exclusivamente los métodos de la interfaz *Pila*



## Navegador de Internet (1/2)

- En un navegador de Internet, se puede guardar el historial de páginas web visitadas en una pila. De esta manera se puede volver al enlace de la página anterior con una simple operación de desapilar().
- Dados el interfaz pila y la clase PaginaWeb

```
package modelos;
public interface Pila<E> {
        void apilar(E x);
        E desapilar();
        E tope();
        boolean esVacia();
   }
```

```
public class PaginaWeb{
  protected String direccionURL;
  public PaginaWeb(String dirURL){...}
  public String toString(){...}
  public boolean equals (Object x){...}
        ...
}
```



## Navegador de Internet (2/2)

1. Diseñar un método recursivo **pagWebVisitada** que reciba como parámetros una **Pila<PaginaWeb>** p y una **PaginaWeb** w para consultar el orden en el que la página web ha sido visitada en el historial, con respecto a la página actual (el tope de la Pila p).
   - Si w coincide con el tope de p el método debe devolver un valor 0, si coincide con el anterior al tope, un 1 y así sucesivamente. Si la página web w no se encuentra en el historial que contiene p, el método deberá indicarlo devolviendo un –1.
2. Estudiar la complejidad temporal del método anterior
   - Talla del problema, instancias significativas, relaciones de recurrencia y cotas de complejidad.



## Implementación de Pila usando ListaConPI

- Proporciona la implementación del modelo Pila en una clase denominada **PilaListaConPi**, dados únicamente:
  - **Pila**: Modelo de Pila
  - **ListaConPI** : Modelo de Lista con Punto de Interés
  - **LEGListaConPI**: Implementación de ListaConPI
- Adicionalmente, añadir el método toString a la clase diseñada.
  - Este método debe mostrar los elementos de la Pila en el único órden en el podrían ser extraídos de la misma.

## Cancelar en Cola

- Sea la interfaz Java Cola tal y como la hemos visto en clase de teoría, se pide:
- Utilizando única y exclusivamente los métodos de esta interfaz, diseñar un método iterativo *cancelar* que elimine todas las apariciones de un cierto dato x de la Cola sobre la que se aplica y sin modificar el orden de sus datos.



## Borrar iguales en Pila

- Se pide diseñar un método recursivo que, in-situ, borre todos los datos de una Pila iguales a uno dado d, sin modificar su orden actual y devuelva como resultado el número de datos que ha borrado.
  - No se podrá utilizar en el diseño métodos que no sean los que define la interfaz Pila que figura a continuación

```
public int  eliminarIguales(E d)
```



## Pasar Mayores de Pila a Cola

- Diseña un método estático y genérico que elimine los elementos mayores que uno dado de una Pila y los devuelva en una Cola que contenga esos elementos.
- La Pila debe mantener el mismo orden de sus elementos (salvo los elementos eliminados).