## HERENCIA

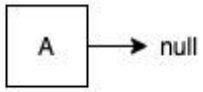
| HERENCIA NORMAL | HERENCIA ABSTRACTA | INTERFACES |
|---|---|---|
| Las clases hijas deben poder sobreescribir cualquier método de la clase padre (aunque NO es obligatorio) | Las clases hijas deben sobreescribir todos los métodos de la clase padre que sean abstractos (ya que tan sólo está declarada la cabecera de los métodos). También podemos tener métodos que no sean abstractos y cuya implementación figure | TODOS los métodos deben ser abstractos e implementados por las clases que la implementen |
| Podemos tener atributos de clase | NO podemos podemos tener atributos de clase,<br>solo constantes | NO podemos tener atributos de clase, solo constantes |

## LISTAS ENLAZADAS

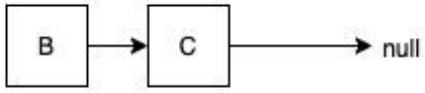
**Ejercicio:** se trata de realizar las dos funciones que se hallan especificadas en la plantilla *T3_Cadenas.java* Para ello se dispone de la susodicha plantilla, de un téster para comprobar la corrección de los resultados y de la definición del nodo con el que se va a trabajar. Se pide el fichero de la plantilla con las dos funciones resueltas (y con las auxiliares que cada cual convenga).

ITERACIÓN 1

cad --> se convierte en anterior



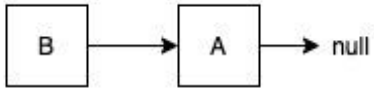

siguiente --> se convierte en cad




```
public NodoCaracter invertida (NodoCaracter cad) {
  NodoCaracter anterior=null, siguiente;
  while(cad!=null) {
     siguiente=cad.siguiente;
     cad.siguiente=anterior;
     anterior=cad;
     cad=siguiente;
  }
  return anterior;
}
```

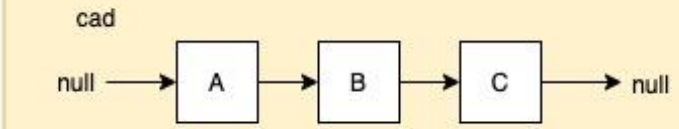
ITERACIÓN 2

cad --> se convierte en anterior



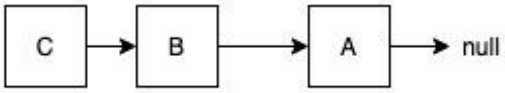

siguiente --> se convierte en cad


**EJEMPLO:**




ITERACIÓN 3

cad --> se convierte en anterior



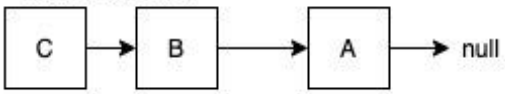

siguiente --> se convierte en cad

null

ITERACIÓN 4

Como cad == null --> PARO

Retorno anterior



**EXÁMENES**

**MAYO 2019**

Problema 1:

```
public void añade(DatosEconomicos de) throws FueraDeTiempo {
  // Si la lista no está vacía y el año del dato <de> que quiero introducir es menor que el
  // último de la lista, entonces lanzo la excepción FueraDeTiempo
  if(lista.size()!=0  && de.año()<lista.get(lista.size()-1).año())
        throw new FueraDeTiempo();
  // En caso contrario, añado el elemento al final de la lista
  lista.add(lista.size(),de);
}
```

Problema 2:

a)
```
DatosEconomicos grecia = new DatosEconomicos(30,"GRECIA",184714,0.9,2018);
DatosEconomicos españa = new DatosEconomicos(34,"ESPAÑA",1208248,1.5,2018);
DatosEconomicos italia = new DatosEconomicos(39,"ITALIA",1753949,1.1,2018);
```

b)
```
HistoriaEconomica gei = new HistoriaEconomica();
try {
        añade(grecia);
        añade(españa);
        añade(italia);
} catch (FueraDeTiempo e) {
        e.getMessage(); // e.printStackTrace() // System.out.println("ERROR: " + e);
}
```

Problema 3:

```
public int numeroDatos(int codigo){
        int cont = 0;
        for(int i=0; i<lista.size();i++){
                if(lista.get(i).codigoPais()==codigo)
                        cont++;
        }
        return cont;
}
```

Problema 4:

```
public boolean hayIPCSuperior(double porcentaje) {
	boolean existe = false;
	for(int i=0; i<lista.size() && !existe; i++){
		if(lista.get(i).ipc()>porcentaje)
			existe = true;
	}
	return existe;
}
```

Problema 5:

```
public int paisPIBMaximo(int año1, int año2) {
	int codigoPais = -1;
	if(año1<=año2 && lista.size()>0){
		int pibMax = -1; // pongo -1 por inicializarlo de alguna forma con un valor “nulo”
		for(int i=0; i<lista.size();i++){
			DatosEconomicos dato = lista.get(i);
			if(dato.año()>=año1 && dato.año()<=año2
				&& pibMax<dato.pib()) {
					pibMax = dato.pib();
					codigoPais = dato.codigoPais();
			}
		}
	}
	return codigoPais;
}
```