

TEMA 1

Conceptos de Java para Estructuras de Datos Herencia y polimorfismo

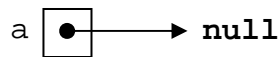
EJERCICIOS RESUELTOS

Ejercicio 1.- Indica el contenido de los vectores *a* y *b* tras las siguientes instrucciones:

```
int a[];  
int b[] = new int[3];  
a = new int[3];  
a[0] = 1;  
b[1] = a[0];  
a = b;  
b[0] = a[1];
```

Solución:

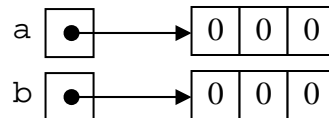
`int a[];`



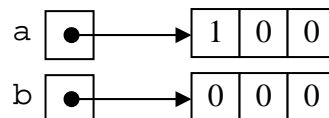
`int b[] = new int[3];`



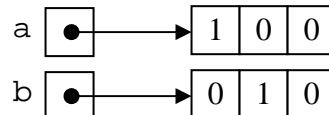
`a = new int[3];`



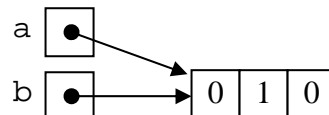
`a[0] = 1;`



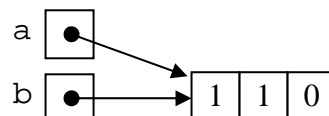
`b[1] = a[0];`



`a = b;`



`b[0] = a[1];`



Ejercicio 2.- Implementar los métodos *area* y *perímetro* en la clase *Circulo*, haciendo uso del siguiente atributo de la clase *Math*:

```
public static final double PI
```

Solución:

```
public double area() {  
    return Math.PI * radio * radio;  
}
```

```
public double perimetro() {  
    return Math.PI * 2 * radio;  
}
```

Ejercicio 3.- Implementar una clase que permita gestionar un conjunto de círculos (como máximo 10 círculos)

- Los círculos se guardarán en un vector (con tamaño máximo 10)
- El constructor debe crear el vector vacío
- Consultores: leer el número de círculos insertados y poder recuperar un círculo del vector
- Modificadores: insertar un círculo en el vector (si hay menos de 10)
- Métodos *toString()* y *equals()*

Solución:

```
public class ColeccionCirculos {  
    /* Atributos */  
    private int numCirculos;  
    private Circulo elArray[];  
    private static final int TAM_MAXIMO = 10;  
  
    /* Constructor */  
    public ColeccionCirculos() {  
        elArray = new Circulo[TAM_MAXIMO];  
        numCirculos = 0;  
    }  
  
    /* Consultores */  
    public int leerNumeroCirculos() {  
        return numCirculos;  
    }  
  
    public Circulo recuperarCirculo(int posicion) {  
        if (posicion >= 0 && posicion < numCirculos)  
            return elArray[posicion];  
        else  
            return null;  
    }  
  
    /* Modificadores */  
    public void insertar(Circulo nuevoCirculo) {  
        if (numCirculos < TAM_MAXIMO)  
            elArray[numCirculos++] = nuevoCirculo;  
    }  
}
```

```

    /* toString */
    public String toString() {
        String res = "Colección de " + numCirculos + " círculos ";
        for (int i = 0; i < numCirculos; i++)
            res += "(" + elArray[i].toString() + ")";
        return res;
    }

    /* equals */
    public boolean equals(Object x) {
        ColeccionCirculos cX = (ColeccionCirculos) x;
        boolean eq = (numCirculos == cX.numCirculos);
        for (int i = 0; i < numCirculos && eq; i++)
            eq = cX.recuperarCirculo(i).equals(elArray[i]);
        return eq;
    }
}

```

Ejercicio 4.- Sean las siguientes clases:

```

public class Animal {
    public void sonido() {
        System.out.println("Grunt");
    }
}

public class Muflon extends Animal {
    public void sonido() {
        System.out.println("MOOOO!");
    }
}

public class Armadillo extends Animal {}

```

¿Qué instrucciones del siguiente programa no son correctas?

```

public class Test1Animal {
    public static void main(String[] args) {
        adoptaAnimal(new Armadillo());
        Object o = new Armadillo();
        Armadillo a1 = new Animal();
        Armadillo a2 = new Muflon();
    }
    private static void adoptaAnimal(Animal a) { }
}

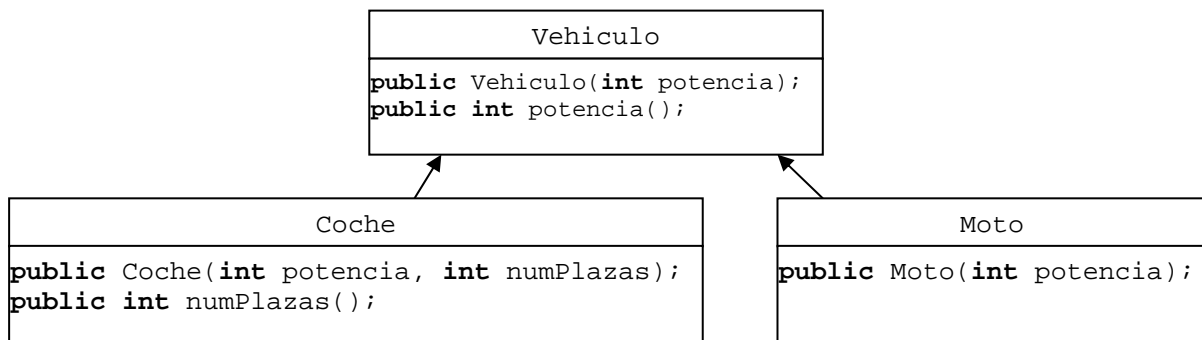
```

Solución:

Armadillo a1 = new Animal();
 No es correcta porque *Armadillo* es una subclase de *Animal*.

Armadillo a2 = new Muflon();
 No es correcta porque *Armadillo* y *Muflon* son clases hermanas

Ejercicio 5.- Dada la siguiente jerarquía de clases:



Diseñar una clase **Garaje** que:

- En el constructor se indique el número total de plazas de garaje
- En cada plaza se pueda guardar tanto un coche como una moto
- Tenga una función que devuelva la cuota mensual de una plaza:
 - Si en dicha plaza hay un coche, la cuota se calcula como la potencia multiplicada por el número de plazas
 - Si en dicha plaza hay una moto, la cuota se calcula como la potencia multiplicada por 2
 - Si no hay ningún vehículo en la plaza, la cuota es 0

Solución:

```
public class Garaje {
    /* Atributos */
    private Vehiculo plazas[];

    /* Constructor */
    public Garaje(int numPlazas) {
        this.plazas = new Vehiculo[numPlazas];
    }

    public void guardarVehiculo(Vehiculo v, int plaza) {
        if (plaza >= 0 && plaza < plazas.length)
            plazas[plaza] = v;
    }

    public int leerCuota(int plaza) {
        int cuota = 0;
        if (plaza >= 0 && plaza < plazas.length && plazas[plaza] != null) {
            if (plazas[plaza] instanceof Coche) { /* Es un coche */
                Coche c = (Coche) plazas[plaza];
                cuota = c.potencia() * c.numPlazas();
            } else /* Es una moto */
                cuota = 2 * plazas[plaza].potencia();
        }
        return cuota;
    }
}
```

Ejercicio 6.- Corrige el código de las siguientes clases para que el método *mostrarReparto* funcione correctamente:

```
public abstract class Persona {
    private String nombre;
    public Persona(String nombre) { this.nombre = nombre; }
}

public class Actor extends Persona {
    private String pelicula;
    public Actor(String nombre, String pelicula) {
        this.nombre = nombre; this.pelicula = pelicula;
    }
}

public class Peliculas {
    public static void mostrarReparto(Actor lista[], String pelicula) {
        for (int i = 0; i < lista.length; i++)
            if (lista[i].pelicula == pelicula)
                System.out.println(lista[i].toString());
    }
}
```

Solución:

```
public abstract class Persona {
    protected String nombre; // Para poder acceder al nombre en la clase Actor
    public Persona(String nombre) {
        this.nombre = nombre;
    }
    // Sería recomendable implementar también los métodos toString y equals
}

public class Actor extends Persona {
    private String pelicula;
    public Actor(String nombre, String pelicula) {
        super(nombre); // Es necesario invocar al constructor de la clase base
        this.pelicula = pelicula; // ya que la clase Persona no tiene constructor vacío
    }
    public String getPelicula() { // Haca falta un consultor para la película
        return pelicula;
    }
    public String toString() { // Requerido para obtener una descripción correcta del actor
        return "Actor de nombre " + nombre + " y película " + pelicula;
    }
    // Sería recomendable implementar también el método equals
}

public class Peliculas {
    public static void mostrarReparto(Actor lista[], String pelicula) {
        for (int i = 0; i < lista.length; i++)
            if (pelicula.equals(lista[i].getPelicula()))
                System.out.println(lista[i].toString());
    }
}
```

Ejercicio 7.- Diseñar la clase *OperacionesArray* que:

- Tenga un método estático que busque un elemento en un *array* y devuelva su posición (devolverá -1 si el elemento no está en el *array*)
- Tenga un método estático que permita eliminar un elemento dado de un *array*. Si el elemento no está no hace nada.

Nota: se asume que todos los datos del array están al principio y el resto de posiciones (a partir del último dato válido) apuntan a *null*.

Solución:

```
public class OperacionesArray {

    public static int buscar(Object x, Object v[]) {
        int pos = -1;
        for (int i = 0; i < v.length && v[i] != null && pos == -1; i++)
            if (v[i].equals(x)) pos = i;
        return pos;
    }

    public static void eliminar(Object x, Object v[]) {
        int pos = buscar(x, v);
        if (pos != -1) {
            while (pos < v.length - 1 && v[pos] != null) {
                v[pos] = v[pos + 1];
                pos++;
            }
            v[pos] = null;
        }
    }
}
```