


















- 1 Queremos hacer una aplicación para profesores y alumnos. De ambos se quiere conocer el nombre, la edad, el sexo y la nacionalidad. De los profesores se quiere conocer además el sueldo, y del alumno si es o no repetidor. Hacer lo siguiente:
  - 1.1 Realiza en java las clases que consideres oportunas teniendo en cuenta las relaciones de herencia. Realiza los constructores sin usar super. Realiza un constructor por defecto para todas las clases. Realiza también los toString. Haz un programa para probarlo.
  - 1.2 Hacer otra versión quitando el constructor por defecto del padre. Solucionarlo de dos maneras posibles:
    - Realizar una llamada explícita a un constructor de la superclase.
    - Utilizando el this(parámetros), y en caso de que no se pueda, utilizar super.

2 Rellena las siguientes tablas. Para ello, realiza ejemplos para probar cada caso.

**Tabla de visibilidad**

	private	friendly	protected	public
Misma clase				
Clase en el mismo paquete				
Clase en otro paquete				
Subclase en el mismo paquete				
Subclase en distinto paquete				

**Tabla de herencia**

	private	friendly	protected	public
Subclase en el mismo paquete				
Subclase en distinto paquete				

- 3 Crear la clase abstracta FiguraGeométrica que tendrá como método abstracto calcular\_area. Dicha clase iniciará los valores de las coordenadas x e y que serán el punto de referencia de la figura. Para el círculo será el centro de la circunferencia y para el resto, el vértice inferior izquierdo. Crear las clases Rectángulo, Círculo y Triángulo heredadas de la clase FiguraGeométrica.
- 4 Crea una clase *empleado* y una subclase *encargado*. Los encargados reciben un 10% más de sueldo base que un empleado normal aunque realicen el mismo trabajo. Implementa dichas clases con el método CalcularSueldo() para ambas clases.

5 ¿Por qué no compila el siguiente código?

```
class prueba{
    protected String nombre;
    protected int id;
    public String getIdent() {return nombre;}
    public int getIdent() {return id;}
}
```

6 Implementa la siguiente estructura de clases:

forma es una superclase que tiene cuatro subclases: círculo, cuadrado, triángulo y rombo.

La clase forma es abstracta y contiene el método abstracto toString(). La clase forma tendrá un método identidad que devuelva una cadena con la subclase a la que pertenece.

Hacer tres versiones:

6.1 Con el método getClass()

6.2 Con instanceof

6.3 Sin usar getClass() ni instanceof

Hacer un programa que cree un array con cuatro objetos, uno de cada subclase y ejecutar de todos el método identidad.

7 Este código está utilizando la estructura de clases del ejercicio anterior. Modifica la sintaxis de las líneas que dan problema y elimina aquellas líneas que aunque sean sintácticamente correctas nunca pueden funcionar.

```
class testforma {
    public static void main(String[] args) {
        Forma f=new Circulo();
        f.identidad();
        Circulo c=new Circulo();
        ((Forma)c).identidad();
        ((Circulo)f).identidad();
        Forma f2=new Forma();
        f2.identidad();
        (Forma)f.identidad();
        f=c;
        c=f;
    }
}
```

8 Averigua los errores del siguiente código:

```
public class Test {
    public int dato=0;
    public static int datostatico=0;
    public void metodo() {this.datostatico++;}
    public static void metodostatico(){
        this.datostatico++;
        datostatico++;
    }
    public static void main(String [] args){
        dato++;
        datostatico++;
        metodostatico();
        metodo();
    }
}
```

9 ¿Qué mostrará el siguiente programa por pantalla?

```
public class Bebe {
    Bebe(int i){
        this("Soy un bebe consentido");
        System.out.println("Hola, tengo " + i + " meses");
    }
    Bebe(String s){
        System.out.println(s);
    }
    void berrea(){
        System.out.println("Buaaaaaaaaa");
    }
    public static void main(String[] args){
        new Bebe(8).berrea();
    }
}
```

10 Averigua sin ejecutar el código, qué mostrará el siguiente programa por pantalla. Una vez que tengas claro lo que el programa debería de mostrar por pantalla ejecuta el código y verifica que lo que has pensado se cumple.

```
public class Bebe {
    static void pedir(){
        System.out.println(str1 + " , " + str2 + " , " + str3);
    }
    static {
        str2 = "mama pipi";
        str3 = "mama agua";
    }
    Bebe() {System.out.println("Nacimiento del bebe"); }
    static String str2, str3, str1 = "papa tengo caca";
    public static void main(String[] args) {
        System.out.println("El bebe se ha despertado y va a pedir cosas");
        System.out.println("El bebe dice: " + Bebe.str1);
        Bebe.pedir();
    }
    static Bebe bebe1 = new Bebe();
    static Bebe bebe2 = new Bebe();
    static Bebe bebe3 = new Bebe();
}
```

11 Tenemos la siguiente clase:

```
public abstract class Sorteo {
    protected int posibilidades;
    public abstract int lanzar();
}
```

Se pide:

- Crear la clase *Dado*, la cual descende de la clase *Sorteo*. La clase *Dado*, en la llamada al método *lanzar* devolverá un número aleatorio del 1 al 6.
- Crear la clase *Moneda*, la cual descende de la clase *Sorteo*. Esta clase en la llamada al método *lanzar* devolverá las palabras cara o cruz.

- 12 Averigua por qué el compilador da un mensaje de error en el siguiente código:

```
class testfinal {  
    public static void main(String[] args) {  
        final String s1=new String("Hola");  
        String s2=new String(" Mundo");  
        s1=s1+s2;  
    }  
}
```

- 13 Tenemos la siguiente clase:

```
public abstract class Vehiculo{  
    private int peso;  
    public final void setPeso(int p){peso=p;}  
    public abstract int getVelocidadActual();  
}
```

- ¿podrá tener descendencia esta clase?
- ¿se pueden sobrescribir todos sus métodos?

- 14 Tenemos una jardinería donde se venden plantas de jardín y productos de alfarería. Ambas disponen de atributos precio y descripción. Además, las plantas disponen de un atributo propio para saber si está regada, y los productos de alfarería tienen otro atributo para indicar si el producto es frágil. Ambas implementan la interfaz Mercancía:

```
interface Mercancia{  
    public double damePrecio();  
    public String dameDescripcion();  
}
```

Haz un programa para probarlo que contenga el siguiente método estático:

```
public static void dameDatos(Mercancia producto)
```

Dicho método deberá mostrar el precio y la descripción del producto.

- 15 Implementa los siguientes interfaces que heredan de la interface Mercancía del ejercicio anterior:

```
interface MercanciaViva extends Mercancia{  
    public boolean necesitaComida();  
    public boolean necesitaRiego();  
}  
interface MercanciaFragil extends Mercancia{  
    public String dameEmbalaje();  
    public double damePeso();  
}
```

Las plantas implementan la interface MercancíaViva y los productos de alfarería implementan MercancíaFragil. Haz un programa para probarlo que contenga el siguiente método estático: public static void dameDatos(Mercancia producto). Dicho método deberá utilizar los métodos de Mercancia, MercanciaViva y MercanciaFragil.

- 16 Realiza una clase *pez* la cual tendrá un atributo nombre de tipo String el cual podrá ser heredado por sus subclases. Implementa en esta clase el método clone así como el método equals.

- 17 Para la clase *pez* anterior, crea un atributo privado entero *numpeces* común a todos los objetos *pez* el cual cuente el número de peces creados. Crea un programa que compruebe que esta variable se incrementa cada vez que se crea un objeto *pez*.
- 18 Realiza una clase *huevo* que esté compuesta por dos clases internas, una clara y otra yema. Realiza un programa para probarlo.
- 19 ¿Qué resultado da el siguiente código? Analiza qué tipo de clase interna se está utilizando y haz una reflexión del resultado.

```
public class VerClaseInterna{
    public static void main(String args[]){
        Contenedor c1 = new Contenedor(34);
        Contenedor.Contenido i1 = c1.new Contenido(23);
        System.out.println(c1.muestraContenedor(i1));
        c1.numero=50;
        System.out.println(i1.muestraContenido());
        i1.numero2=25;
        System.out.println(c1.muestraContenedor(i1));
        i1.numero2=65;
        System.out.println(i1.muestraContenido());
    }
}
public class Contenedor{
    public int numero=0;
    public Contenedor(int numero){
        this.numero=numero;
    }
    public String muestraContenedor(Contenido refCont){
        return "N. contenedor= "+numero+" N. contenido= "+refCont.numero2;
    }
    public class Contenido{
        public int numero2;
        public Contenido(int numero){
            numero2=numero;
        }
        public String muestraContenido(){
            return "N.contenedor= "+numero+" N.contenido= "+numero2;
        }
    }
}
```

- 20 ¿Qué resultado da el siguiente código? Analiza qué tipo de clase interna se está utilizando y haz una reflexión del resultado.

```
public class ClaseLocal {
    public int numero=0;
    public ClaseLocal(int numero){
        this.numero=numero;
    }
    public String muestraContenido(){
        class Mostrador{
            public String muestraDato(){
                return "Número = "+numero;
            }
        }
        Mostrador m = new Mostrador();
        return m.muestraDato();
    }
}
```

```

    }
}
public class VerClaseLocal{
    public static void main(String args[]){
        ClaseLocal c1 = new ClaseLocal(346);
        System.out.println(c1.muestraContenido());
    }
}

```

- 21 ¿Qué resultado da el siguiente código? Analiza qué tipo de clase interna se está utilizando y haz una reflexión del resultado.

```

public class VerClaseAnidada{
    public static void main(Strings args[]){
        PrimerContenedor.Contenido il=new PrimerContenedor.Contenido(29);
        il.numero2=25;
        System.out.println(il.muestraContenido());
        PrimerContenedor c1 = new PrimerContenedor(34);
        System.out.println(c1.muestraContenedor(il));
    }
}

public class PrimerContenedor{
    public int numero=0;
    static public int numero3=13;
    public PrimerContenedor(int numero){
        this.numero=numero;
    }
    public String muestraContenedor(Contenido refCont){
        return "Nº contenedor= "+numero+"Nº contenido= "+refCont.numero2;
    }
    static class Contenido{
        public int numero2;
        public Contenido(int numero){
            numero2=numero;
        }
        public String muestraContenido(){
            return "Nº contenedor= "+numero3+" Nº contenido= "+numero2;
        }
    }
}

```

- 22 Una **tienda de informática** dispone de una serie de productos. De dichos productos se quiere almacenar la marca, el modelo y el stock. Los productos son los siguientes:
- Ordenadores: se quiere almacenar la velocidad del procesador y la capacidad del disco duro. Pueden ser de sobremesa o portátiles. De los portátiles, se quiere almacenar además el peso. De los de sobremesa, queremos también almacenar el monitor que se vende con dicho ordenador. Del monitor queremos almacenar la marca y la resolución. Los monitores solamente son visibles para los ordenadores de sobremesa permaneciendo ocultos para el resto de clases del diseño.
  - Dispositivos: se quiere almacenar la capacidad de la tarjeta de memoria. Pueden ser móviles o cámaras fotográficas. De las cámaras se quiere saber además si son reflex o compactas, y también si son analógicas o digitales. De los móviles se quiere saber si es de 3G o de 4G.

Realiza en java las clases que consideres oportunas teniendo en cuenta las relaciones de herencia. Realiza los constructores utilizando super. Para todas las clases, haz un método toString, clone y equals. Haz un programa para probarlo.

23 Nuestro jefe está pensando en hacer un **videojuego**. A nosotros nos ha encargado que hagamos los insectos y las plantas. Entre ambos no puede haber ninguna relación de herencia. De los insectos queremos saber el color, el número de patas y el peso(en formato decimal). De las plantas, la altura(en formato decimal) y si tiene flores.

De ambas clases tenemos que hacer el toString, el clone y el equals y poder conocer el número de objetos creados en cada clase.

Haz un programa que haga lo siguiente en este orden:

1. Crea varias plantas y varios insectos indicando el orden de creación. Ej: “Se creado la planta/insecto número x”, donde x será 1 para la primera planta y el primer insecto, x será 2 para los segundos, etc. El mensaje hazlo en el programa y para saber el orden de creación, utiliza métodos de las clases.

2. Utilizar el método equals para comparar dos plantas iguales, dos plantas diferentes, dos insectos iguales, dos insectos diferentes, una planta y un insecto. Utiliza el método toString para mostrar resultados. Ej: “El insecto color=Amarillo,numPatas=4,pesoGr=200.0 no es igual al insecto color=Rojo,numPatas=4,pesoGr=200.0”

3. Utilizar el método clone para clonar una planta y un insecto. Utiliza el método toString para mostrar resultados de los objetos y sus clones.

4. Utiliza los métodos de las clases para conocer el número de plantas y el número de insectos creados hasta el momento en la aplicación.

5. Tanto los insectos como las plantas se reproducen de la misma manera, generando un ser exactamente igual. Los insectos además comen, engordando su peso en gramos la mitad de los gramos de comida ingeridos, es decir, si un insecto se come 4 gramos de comida, engorda 2 gramos. Las plantas se riegan, y crecen en cm el doble de los litros ingeridos, es decir, si se riega una planta con 1 litro de agua, la planta crece 2 cm. Modelar dichos comportamientos utilizando herencia de interfaces. Realizar un método que tenga como parámetro un array de variables de interfaz, donde reciba varios insectos y plantas. El método recorrerá el array y donde encuentre un insecto le dará de comer tantos gramos de comida como el índice del array, es decir, si en el índice 4 hay un insecto, le dará 4 gramos de comida. Igualmente, si lo que encuentra es una planta, la regará con tantos litros de agua como el índice del array, es decir, si en el índice 5 hay una planta, la regará con 5 litros de agua. Después de haber comido o haber sido regados, se reproducirán todos los insectos y plantas almacenándolos en un segundo array en el mismo índice, es decir, si hay un insecto en el array en el índice 4, en el segundo array, en el índice 4 estará su reproducción. El método devolverá este segundo array. El programa tendrá que llamar a este método, y luego mostrar los resultados oportunos para demostrar que el método ha funcionado bien.

24 Queremos realizar una aplicación para gestionar **empleados comerciales**.

De los empleados queremos saber su nombre, el sueldo y la fecha de alta en la empresa. Como fecha de alta se tomará la fecha del sistema en el momento en que se crea el empleado.

Tenemos distintos tipos de empleados:

- Trabajadores: cobran el sueldo base.
- Encargados: cobran el sueldo base más un 20% de dicho sueldo base.
- Jefes: cobran el sueldo base más un 50% de dicho sueldo base.

De los empleados se necesita saber las horas que llevan trabajadas y las ventas. Las ventas deberán almacenarse con un número de venta y el importe de dicha venta. Las ventas solamente son visibles para los empleados permaneciendo ocultas para el resto de clases del diseño.

Realizar el clone, el equals y el toString para los empleados.

Además, los empleados implementan la interfaz Trabajar. Dicha interfaz tiene los siguientes métodos:

- incrementar\_horas\_trabajadas: este método incrementará el número de horas realizadas en su cómputo total.

- `venta_realizada`: este método almacenará la información de la nueva venta.
- `aumento_productividad`: este método incrementará el sueldo en un porcentaje realizado sobre el importe total de ventas.

25 Nos han mandado hacer una aplicación para un **puesto de feria**. El dueño dispone de de varias sorpresas que pueden ser juguetes o chucherías. De todas las sorpresas se quiere saber el nombre. Dicho nombre solamente puede contener letras. Crearle al nombre un método `set` donde si no es correcto, lance la excepción `NombreIncorrectoException`. Al construir la sorpresa, si el nombre no es correcto, se obligará al código que lo haya invocado a tratar la excepción. De los juguetes tenemos dos tipos: peluches y muñecas. De ambos se quiere saber el color, y de los peluches interesa conocer también si hablan y el tamaño: pequeño, mediano y grande. Los juguetes tienen un método abstracto para calcular el valor económico. Para las muñecas dicho valor económico son 10 euros, y para los peluches:

- Pequeños: 2 euros
- Medianos: 5 euros
- Grandes: 7 euros

Para el tema de la garantía, las muñecas llevan también la fecha de compra. Las fechas de compra posibles son el 20 de febrero del 2014 ó el 1 de marzo del 2014. Sólo son posibles estas dos fechas porque las muñecas se compraron solamente en ambos días. Al mostrar la fecha hay que hacerlo con el siguiente formato: 20-02-2014. Además, las muñecas tienen un bebé. Los bebés solamente son visibles para las muñecas permaneciendo ocultos al resto de clases del diseño. Los bebés implementan la siguiente interfaz:

- `dormir()`; Para dormirlo, tiene que estar despierto.
- `despertar()`; Para despertarlo, tiene que estar dormido.
- `comer(float litros)`; Por cada litro de biberón, el bebé engorda una cuarta parte en gramos, es decir, si toma 0.5 litro de biberón, engorda 0.125 gramos. Para poder comer, tiene que estar despierto. Los gramos del bebé se deben mostrar con 2 decimales.

26 Queremos diseñar una aplicación para gestionar **vehículos**, para lo cual dispondremos de una clase `Vehiculo`. De dicha clase heredan dos clases:

- `Triciclo`
- `Vehiculo_motor`: de dicha clase heredan la clase `Coche` y la clase `Moto`.

De los vehículos se quiere conocer la matrícula, la marca y la fecha de compra. Y si es un vehículo de motor, también se quiere conocer la cilindrada y los caballos del motor. El motor solamente es visible para los vehículos a motor permaneciendo oculto al resto de clases del diseño.

Las marcas pueden ser: SKIPE y GLOBE. La fecha se mostrará con el siguiente formato: dd/mm/aaaa.

La clase `Vehiculo` tendrá un método `CalcularNumRuedas`. Dicho método devolverá:

- 3 para los triciclos
- 4 para los coches
- 2 para las motos

La clase `Vehiculo_motor` implementa la interfaz `Arrancable`. Dicha interfaz tiene los siguientes métodos:

- `public void arrancar();`
- `public void parar();`
- `public void subir_marcha();`
- `public void bajar_marcha();`



27 Un **veterinario** nos ha pedido que le hagamos una aplicación para su clínica. Las mascotas que lleva son mamíferos y aves. Dentro de los mamíferos, lleva perros y gatos, y dentro de las aves, loros y periquitos. De todos los animales se quiere conocer el nombre, la cuota mensual y la raza. De los mamíferos, se quiere saber también qué champú utilizan para el pelo, y de los perros el grado de agresividad. De las aves interesa conocer su alimentación, es decir, si son insectívoras, herbívoras, omnívoras o carnívoras. De los loros, interesa saber si hablan o no el lenguaje humano. Los animales implementan la interfaz Comunicarse que tiene los siguientes métodos: hablar y enfadarse. El veterinario considera que si un animal está enfadado es porque está enfermo, entonces le hace análisis más exhaustivos. En el método hablar, el perro dice “guau”, el gato “miau”, etc. También se quieren conocer las crías de los animales con sus nombres y la fecha de nacimiento. Dicha fecha se mostrará de la siguiente manera: 28-febrero-2012.

Todos los animales tienen un código compuesto por:

- un carácter con la inicial en mayúscula del tipo de animal:
  - P: perro
  - G: gato
  - L: loro
  - R: periquito
- un número de tres dígitos que se va incrementando por cada tipo de animal. Es decir, para el primer perro la clave será P001, para el segundo perro P002,...G001 para el primer gato, G002 para el segundo...

Dicha clave se genera automáticamente cada vez que se introduce un nuevo animal.