# RESUMEN DEL CAPÍTULO

Este tema es una continuación del Capítulo 4. En este tema se verán más conceptos de la Programación Orientada a Objetos como es el polimorfismo, overloading, casting, clases anidadas, etc. También se estudiarán clases de utilidad como son los wrappers y clases para el manejo de la fecha y la hora. Otros conceptos como las clases y métodos abstractos y finales también se abordan en este capítulo.

# EJERCICIOS RESUELTOS

**1. Crea una clase *empleado* y una subclase *encargado*. Los encargados reciben un 10% más de sueldo ba se que un empleado normal aunque realicen el mismo trabajo. Implementa dichas clases en el paquet sobrescribe y sobrescribe el método *getSueldo()* para ambas clases. Variables miembro no deberán d poder ser accesibles desde el exterior.**

**Solución:**

```
package sobreescribe;

public class Empleado {
protected int sueldobase;
public int getSueldo(){
    return sueldobase;
}
}

package sobreescribe;

public class Encargado extends Empleado {
public int getSueldo (){
   return sueldobase * 1,1;
}
}
```

**2.** Compilará el siguiente código:

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

**En caso de que no compile expón las razones.**

**Solución:**

En el código anterior parece que se quiere hacer una especie de sobrecarga del método *getIdent()* pero la clas no compilará dado que el compilador se va a encontrar dos métodos con el mismo nombre y con la misma lista d parámetros, con lo cual, para él va a ser una implementación repetida. Concretamente con el compilador Geany dar el siguiente error:

```
prueba.java:5: getIdent() is already defined in prueba
```

Recuerda que para la sobrecarga de un método era necesario cambiar la lista de argumentos del mismo.

**3.** Realiza una función que dada la fecha de nacimiento de una persona indique cuántos años tiene.

**Solución:**

```
public static int edad(String fecha_nac) {
   //Importante: fecha_nac tiene que tener el formato dd/MM/yyyy
   java.util.Date hoy = new Date(); //Fecha actual
   String[] tokens = fecha_nac.split("/");
     Calendar cal = new GregorianCalendar(Integer.parseInt(tokens[2]), Integer
parseInt(tokens[1])-1, Integer.parseInt(tokens[0]));
   //Se resta 1 porque los meses comienzan en 0
   java.sql.Date fecha = new java.sql.Date(cal.getTimeInMillis());
   long diferencia = ( hoy.getTime() - fecha.getTime() )/(24 * 60 * 60 * 1000);
   //Se divide por los milisegundos que tiene un día. Se obtiene la diferencia en días

   return (int)diferencia/365;
}
```

**4.** Tenemos una clase con un método *metodox()* que debe devolver un valor entero y da problemas al compilar. Part del cuerpo de la clase es el siguiente:

```
int dato;
.....
public int metodox(){
```

```
        return dato * 1.1;
    }
```

**Se pide la reescritura del método utilizando un *wrapper Double* que solvente el problema d compilación.**

**Solución:**

```
    public int metodox(){
        Double d = new Double(dato*1.1);
        return d.intValue();
    }
```

**5. Realiza una clase huevo que esté compuesta por dos clases internas, una clara y otra yema. Crea d métodos *hazYema()* y *hazClara()* que generen objetos de las clases yema y clara respectivamente. Reali un método main en el que se creen objetos de cada una de las clases.**

**Solución:**

```
public class huevo {
    class yema {
        yema() { System.out.println("Inicializando yema"); }
    }
    class clara {
        clara() { System.out.println("Inicializando clara"); }
    }
    huevo() { System.out.println("inicializando huevo"); }

    yema hazYema() {
        return new yema();
    }
    clara hazClara() {
        return new clara();
    }
    public static void main(String[] args) {
        huevo h = new huevo();
        yema y = h.hazYema();
        clara c = h.hazClara();
    }
}
```

- **6.** Averigua si son verdaderas o falsas las siguientes afirmaciones:
  - Una clase abstracta es una clase que no se puede instanciar.
  - Una clase abstracta es una clase que se usa únicamente para definir superclases.
  - Los métodos de las clases abstractas no tienen implementación.
  - En la declaración de una interfaz solamente pueden aparecer declaraciones de método y atributos pero nu implementación de métodos.
  - Las interfaces no encapsulan datos.
  - Las interfaces pueden definir constantes simbólicas.

  La solución a este ejercicio se encuentra al final de los ejercicios propuestos.

- **7.** Implementa la siguiente estructura de clases.

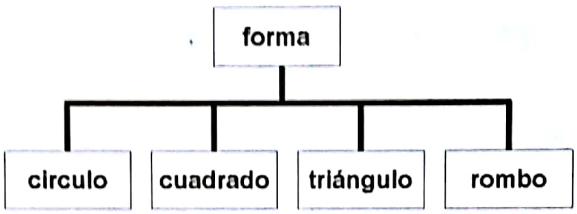


*Figura 5.4. Jerarquía de clases desdencientes de la clase forma*

**Esta jerarquía deberá de tener las siguientes características:**

- **La clase forma deberá de ser abstracta.**
- **La clase forma tendrá el método abstracto toString().**
- **La clase forma tendrá un método identidad que muestre el identificador interno de la clase.**
- **Círculo, cuadrado, triángulo y rombo descienden de la clase forma.**
- **Estas clases implementarán el método abstracto de la clase padre.**

**Solución:**

```
abstract class forma {
    void identidad() { System.out.println(this); }
    abstract public String toString();
}

class circulo extends forma {
    public String toString() { return "círculo"; }
}

class cuadrado extends forma {
```

```
    public String toString() { return " cuadrado "; }
}

class triangulo extends forma {
    public String toString() { return "triángulo"; }
}

class rombo extends forma {
    public String toString() { return "rombo"; }
}
```

- **8. En el siguiente ejercicio puedes ver cómo se utiliza el upcasting y downcasting, pero...**

```
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();
    }
}
```

- Para completar el ejercicio deberás de hacer lo siguiente:
  - Modificar la sintaxis de las líneas que dan problema.
  - Eliminar aquellas líneas que aunque sean sintácticamente correctas nunca pueden funcionar.

**Solución:**

```
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();
//Las clases abstractas nunca pueden ser instanciadas
```

```
            ((forma)f).identidad();
        }
    }
```

**Qué mostrará el programa por pantalla tras los cambios?**

**Solución:**

círculo

círculo

círculo

círculo

# EJERCICIOS PROPUESTOS

**1.** Averigua sin ejecutar el código qué mostrará el siguiente programa por pantalla:

```
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();
}
```

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.

**2.** ¿Compilará el siguiente programa?

En caso afirmativo averigua sin ejecutar el código qué mostrará por pantalla:

```
public class bebe {
    static void pedir(String... argumentos) {
        for(String str : argumentos) System.out.println(str);
    }
    public static void main(String[] args) {
        pedir("mama pipi", "mama caca", "mama agua");
        pedir(new String[]{"papa jugar", "mama me aburro", "papa sed", "papa dormir
"mama tengo hambre"});
    }
}
```

**3.** Tenemos la siguiente clase:

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

Se pide:

- Crear la clase *dado*, la cual desciende de la clase *sorteo*. La clase *lado*, en la llamada *lanzar()* mostrará un nu mero aleatorio del 1 al 6.
- Crear la clase *moneda*, la cual desciende de la clase *sorteo*. Esta clase en la llamada al método *lanzar()* mostrar las palabras *cara* o *cruz*.

**4.** Realiza una clase *conversor* que tenga las siguientes características:

- Toma como parámetro en el constructor un valor entero.
- Tiene un método getNumero que dependiendo del parámetro devolverá el mismo número en el siguient formato:

| Parámetro | Formato |
|---|---|
| B | Binario (String) |
| H | Hexadecimal (String) |
| O | Octal (String) |

- Realiza un método *main* en la clase para probar todo lo anterior.

**5.** Realiza una clase *conversorfechas* que tenga los siguientes métodos:

- *String* **normalToAmericano**(*String*). Este método convierte una fecha en formato normal dd/m formato americano mm/dd/yyyy
- *String* **americanoToNormal**(*String*). Este método realiza el paso contrario, convierte fechas en americano a formato normal.

**6.** Estoy intentando hacer lo siguiente en mi programa:

```
final String s1=new String("Hola");
String s2=new String(" Mundo");
s1=s1+s2;
```

- El compilador muestra un mensaje de error. ¿Qué podrá estar sucediendo?

**7.** 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?
- Razona tus respuestas.

**8.** Realiza una clase *Test* en con un método *main* que tome por teclado dos números y muestre la suma, mu ción, división y módulo. En el caso de que el segundo número sea 0, el programa deberá de atrapar las exce que se puedan producir. Para la resolución de este problema necesitarás utilizar *wrappers*.

**9.** Diseña una clase con un método que permita averiguar la última cifra de un número introducido por Para la resolución de este problema deberás utilizar wrappers de tipos numéricos.

**Solución ejercicio resuelto número 6:**

- (Verdadera) Una clase abstracta es una clase que no se puede instanciar.
- (Falsa) Una clase abstracta es una clase que se usa únicamente para definir superclases.
- (Falsa) Los métodos de las clases abstractas no tienen implementación.
- (Falsa) En la declaración de un interfaz solamente pueden aparecer declaraciones de método y at pero nunca implementación de métodos.
- (Verdadera) Las interfaces no encapsulan datos.
- (Verdadera) Las interfaces pueden definir constantes simbólicas.