

#### Punto 4.1 (ejercicio 1)

### Clase Cuenta:

```
package ejeercicio1;

public class Cuenta {

    protected float saldo;

    protected int númeroConsignaciones = 0;

    protected int númeroRetiros = 0;

    protected float tasaAnual;

    protected float comisiónMensual = 0;

    public Cuenta(float saldo, float tasaAnual) {

        this.saldo = saldo;

        this.tasaAnual = tasaAnual;

    }

    public void consignar(float cantidad) {

        saldo = saldo + cantidad; /* Se actualiza el saldo con la cantidad
        consignada */

        // Se actualiza el número de consignaciones realizadas en la cuenta
        númeroConsignaciones = númeroConsignaciones + 1;

    }

    public void retirar(float cantidad) {

        float nuevoSaldo = saldo - cantidad;

        /* Si la cantidad a retirar no supera el saldo, el retiro no se puede
        realizar */

        if (nuevoSaldo >= 0) {

            saldo -= cantidad;

            númeroRetiros = númeroRetiros + 1;

        } else {

            System.out.println("La cantida a retirar excede el saldoactual.");

        }

    }

}
```

```

    }

    /**
     * Método que calcula interés mensual de la cuenta a partir de la tasa
     * anual aplicada
     */
    public void calcularInterés() {
        float tasaMensual = tasaAnual / 12; /* Convierte la tasa anual en
        mensual */

        float interesMensual = saldo * tasaMensual;

        saldo += interesMensual; /* Actualiza el saldo aplicando el interés
        mensual */
    }

    /**
     * Método que genera un extracto aplicando al saldo actual una
     * comisión y calculando sus intereses
     */
    public void extractoMensual() {
        saldo -= comisiónMensual;

        calcularInterés();
    }
}

```

## Clase Cuenta Ahorros

```
package ejeercicio1;
```

```
public class CuentaAhorros extends Cuenta {
```

```
    private boolean activa;
```

```
/**
```

```
* Constructor de la clase CuentaAhorros
```

```
* @param saldo Parámetro que define el saldo de la cuenta de ahorros
```

\* @param tasa Parámetro que define la tasa anual de interés de la

\* cuenta de ahorros

\*/

```
public CuentaAhorros(float saldo, float tasa) {
```

```
    super(saldo, tasa);
```

```
    if (saldo < 10000) /* Si el saldo es menor a 10000, la cuenta no  
        se activa */
```

```
        activa = false;
```

```
    else
```

```
        activa = true;
```

```
}
```

```
/**
```

\* Método que recibe una cantidad de dinero a retirar y actualiza el

\* saldo de la cuenta

\* @param saldo Parámetro que define la cantidad a retirar de una

\* cuenta de ahorros

\*/

```
public void retirar(float cantidad) {
```

```
    if (activa) // Si la cuenta está activa, se puede retirar dinero
```

```
        super.retirar(cantidad); /* Invoca al método retirar de la clase
```

```
        padre */
```

```
}
```

```
/**
```

\* Método que recibe una cantidad de dinero a consignar y actualiza

\* el saldo de la cuenta

\* @param saldo Parámetro que define la cantidad a consignar en

\* una cuenta de ahorros

\*/

```
public void consignar(float cantidad) {
```

```

if (activa) // Si la cuenta está activa, se puede consignar dinero
super.consignar(cantidad); /* Invoca al método consignar de
la clase padre */
}
/**
 * Método que genera el extracto mensual de una cuenta de ahorros
 */
public void extractoMensual() {
/* Si la cantidad de retiros es superior a cuatro, se genera una
comisión mensual */
if (númeroRetiros > 4) {
comisiónMensual += (númeroRetiros - 4) * 1000;
}
super.extractoMensual(); // Invoca al método de la clase padre
/* Si el saldo actualizado de la cuenta es menor a 10000, la
cuenta no se activa */
if ( saldo < 10000 )
activa = false;
}
/**
 * Método que muestra en pantalla los datos de una cuenta de
ahorros
 */
public void imprimir() {
System.out.println("Saldo = $ " + saldo);
System.out.println("Comisión mensual = $ " + comisiónMensual);
System.out.println("Número de transacciones = " +(númeroConsignaciones + númeroRetiros));
System.out.println();
}

```

```
}
```

## Clase Cuenta Corriente:

```
package ejeercicio1;
```

```
public class CuentaCorriente extends Cuenta {  
    /* Atributo que define un sobregiro de la cuenta que surge cuando el  
    saldo de la cuenta es negativo */  
    float sobregiro;  
    /**  
    * Constructor de la clase CuentaCorriente  
    * @param saldo Parámetro que define el saldo de la cuenta corriente  
    * @param tasa Parámetro que define la tasa anual de interés de la  
    * cuenta corriente  
    */  
    public CuentaCorriente(float saldo, float tasa) {  
        super(saldo, tasa); // Invoca al constructor de la clase padre  
        sobregiro = 0; // Inicialmente no hay sobregiro  
    }  
    /**  
    * Método que recibe una cantidad de dinero a retirar y actualiza el  
    * saldo de la cuenta  
    * @param cantidad Parámetro que define la cantidad de dinero a  
    * retirar de la cuenta corriente  
    */  
    public void retirar(float cantidad) {  
        float resultado = saldo - cantidad; // Se calcula un saldo temporal  
        /* Si el valor a retirar supera el saldo de la cuenta, el valor  
        excedente se convierte en sobregiro y el saldo es cero */  
        if (resultado < 0) {
```

```

sobregiro = sobregiro - resultado;

saldo = 0;

} else {

super.retirar(cantidad); /* Si no hay sobregiro, se realiza un
retiro normal */

}

}

/**
 * Método que recibe una cantidad de dinero a consignar y actualiza
 * el saldo de la cuenta
 * @param cantidad Parámetro que define la cantidad de dinero a
 * consignar en la cuenta corriente
 */
public void consignar(float cantidad) {
float residuo = sobregiro - cantidad;
if (sobregiro > 0) {
    if ( residuo > 0) { /* Si el residuo es mayor que cero, se libera
    el sobregiro */
        sobregiro = 0;
        saldo = residuo;
    } else { /* Si el residuo es menor que cero, el saldo es cero y
    surge un sobregiro */
        sobregiro = -residuo;
        saldo = 0;
    }
} else {
    super.consignar(cantidad); /* Si no hay sobregiro, se realiza
    una consignación normal */
}
}

```

```

    }

    /**
     * Método que genera el extracto mensual de la cuenta
     */
    public void extractoMensual() {
        super.extractoMensual(); // Invoca al método de la clase padre
    }

    /**
     * Método que muestra en pantalla los datos de una cuenta corriente
     */
    public void imprimir() {
        System.out.println("Saldo = $ " + saldo);
        System.out.println("Cargo mensual = $ " + comisiónMensual);
        System.out.println("Número de transacciones = " +(númeroConsignaciones +
númeroRetiros));

        System.out.println("Valor de sogregiro = $" + (númeroConsignaciones + númeroRetiros));
        System.out.println();
    }
}

```

### **Clase Prueba Cuenta:**

```

package ejeercicio1;

import java.util.*;

public class PruebaCuenta {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Cuenta de ahorros");
        System.out.println("Ingrese saldo inicial= $");
    }
}

```

```
float saldoInicialAhorros = input.nextFloat();  
System.out.print("Ingrese tasa de interés= ");  
float tasaAhorros = input.nextFloat();  
CuentaAhorros cuenta1 = new  
CuentaAhorros(saldoInicialAhorros, tasaAhorros);  
System.out.print("Ingresar cantidad a consignar: $");  
float cantidadDepositar = input.nextFloat();  
cuenta1.consignar(cantidadDepositar);  
System.out.print("Ingresar cantidad a retirar: $");  
float cantidadRetirar = input.nextFloat();  
cuenta1.retirar(cantidadRetirar);  
cuenta1.extractoMensual();  
cuenta1.imprimir();
```

```
System.out.println("Cuenta corriente");  
System.out.println("Ingrese saldo inicial= $");  
float saldoInicialCorriente = input.nextFloat();  
System.out.print("Ingrese tasa de interés= ");  
float tasacorriente = input.nextFloat();  
CuentaCorriente cuenta2 = new  
CuentaCorriente(saldoInicialCorriente, tasacorriente);  
System.out.print("Ingresar cantidad a consignar: $");  
float cantidadDepositar1 = input.nextFloat();  
cuenta2.consignar(cantidadDepositar1);  
System.out.print("Ingrese cantidad a retirar= ");  
float retiroCorriente= input.nextFloat();  
  
cuenta2.retirar(retiroCorriente);  
cuenta2.extractoMensual();
```



```
cuenta2.imprimir();
```

```
input.close();
```

```
}
```

```
}
```

## Punto 4.2 (ejercicio 2):

### Clase Apartaestudio

```
package Inmuebles;
```

```
public class Apartaestudio extends Apartamento {
```

```
    // Atributo que identifica el valor por área de un apartaestudio
```

```
    protected double valorArea = 1500000;
```

```
    /**
```

```
    * Constructor de la clase Apartaestudio
```

```
    * @param identificadorInmobiliario Parámetro que define el
```

```
    * identificador inmobiliario de un apartaestudio
```

```
    * @param área Parámetro que define el área de un apartaestudio
```

```
    * @param dirección Parámetro que define la dirección donde se
```

```
    * encuentra localizado un apartaestudio
```

```
    * @param númeroHabitaciones Parámetro que define el número de
```

```
    * habitaciones que tiene un apartaestudio
```

```
    * @param númeroBaños Parámetro que define el número de baños
```

```
    * que tiene un apartaestudio
```

```
    */
```

```
    public Apartaestudio(int identificadorInmobiliario, int área, String
```

```

dirección,
int númeroHabitaciones, int númeroBaños) {
// Invoca al constructor de la clase padre
// Los apartaestudios tienen una sola habitación y un solo baño
super(identificadorInmobiliario, área, dirección, 1, 1);
}
/**
 * Método que muestra en pantalla los datos de un apartaestudio
 */
void imprimir() {
super.imprimir(); // Invoca al método imprimir de la clase padre
System.out.println();
}
}

```

## Clase Apartamento

```

package Inmuebles;

public class Apartamento extends InmuebleVivienda {
/**
 * Constructor de la clase Apartamento
 * @param identificadorInmobiliario Parámetro que define el
 * identificador inmobiliario de un apartamento
 * @param área Parámetro que define el área de un apartamento
 * @param dirección Parámetro que define la dirección donde se
 * encuentra localizado un apartamento
 * @param númeroHabitaciones Parámetro que define el número de
 * habitaciones que tiene un apartamento
 * @param númeroBaños Parámetro que define el número de baños

```

```

    * que tiene un apartamento
    */

    public Apartamento(int identificadorInmobiliario, int área, String
    dirección, int númeroHabitaciones, int númeroBaños) {
        // Invoca al constructor de la clase padre
        super(identificadorInmobiliario, área, dirección, númeroHabitaciones, númeroBaños);
    }

    /**
    * Método que muestra en pantalla los datos de un apartamento
    */
    void imprimir() {
        super.imprimir(); // Invoca al método imprimir de la clase padre
    }
}

```

## Clase Apartamento Familiar

```

package Inmuebles;

public class ApartamentoFamiliar extends Apartamento {

    protected double valorArea = 2000000;

    /* Atributo que identifica el valor de la administración de un
    apartamento familiar */
    protected int valorAdministración;

    /**
    * Constructor de la clase ApartamentoFamiliar
    * @param identificadorInmobiliario Parámetro que define el
    * identificador inmobiliario de un apartamento familiar
    * @param área Parámetro que define el área de un apartamento familiar
    * @param dirección Parámetro que define la dirección donde se
    */
}

```

```

* encuentra localizado un apartamento familiar

* @param númeroHabitaciones Parámetro que define el número de

* habitaciones que tiene un apartamento familiar

* @param númeroBaños Parámetro que define el número de baños

* que tiene un apartamento familiar

* @param valorAdministración Parámetro que define el valor de la

* administración de un apartamento familiar

*/

public ApartamentoFamiliar(int identificadorInmobiliario, int área,
String dirección, int númeroHabitaciones, int númeroBaños, int
valorAdministración) {
// Invoca al constructor de la clase padre
super(identificadorInmobiliario, área, dirección,
númeroHabitaciones, númeroBaños);
this.valorAdministración = valorAdministración;
}

/**
* Método que muestra en pantalla los datos de un apartamento familiar
*/

void imprimir() {
super.imprimir(); // Invoca al método imprimir de la clase padre
System.out.println("Valor de la administración = $" +
valorAdministración);
System.out.println();
}
}

```

## Clase Casa

```
package Inmuebles;
```

```

public class Casa extends InmuebleVivienda{

    protected int númeroPisos; /* Atributo que identifica el número de
    pisos que tiene una casa */

    /**
    * Constructor de la clase Casa
    * @param identificadorInmobiliario Parámetro que define el
    * identificador inmobiliario de una casa
    * @param área Parámetro que define el área de una casa
    * @param dirección Parámetro que define la dirección donde se
    * encuentra localizada una casa
    * @param númeroHabitaciones Parámetro que define el número de
    * habitaciones que tiene una casa
    * @param númeroBaños Parámetro que define el número de baños
    * que tiene una casa
    * @param númeroPisos Parámetro que define el número de pisos
    * que tiene una casa
    */

    public Casa(int identificadorInmobiliario, int área, String dirección,
    int númeroHabitaciones, int númeroBaños, int númeroPisos) {
    // Invoca al constructor de la clase padre
    super(identificadorInmobiliario, área, dirección,
    númeroHabitaciones, númeroBaños);
    this.númeroPisos = númeroPisos;
    }

    /**
    * Método que muestra en pantalla los datos de una casa
    */

    void imprimir() {
    super.imprimir(); // Invoca al método imprimir de la clase padre

```

```
System.out.println("Número de pisos = " + númeroPisos);  
}  
}
```

## Clase CasaConjuntoCerrado

```
package Inmuebles;
```

```
public class CasaConjuntoCerrado extends CasaUrbana {  
    // Atributo que define el valor por área de una casa en conjunto cerrado  
    protected static double valorArea = 2500000;  
    /* Atributo que define el valor de administración de una casa en  
    conjunto cerrado */  
    protected int valorAdministración;  
    // Atributo que define si una casa en conjunto cerrado tiene piscina  
    protected boolean tienePiscina;  
    /* Atributo que define si una casa en conjunto cerrado tiene campos  
    deportivos */  
    protected boolean tieneCamposDeportivos;  
    /**  
    * Constructor de la clase CasaConjuntoCerrado  
    * @param identificadorInmobiliario Parámetro que define el  
    * identificador inmobiliario de una casa en conjunto cerrado  
    * @param área Parámetro que define el área de una casa en conjunto  
    * cerrado  
    * @param dirección Parámetro que define la dirección donde se  
    * encuentra localizada una casa en conjunto cerrado  
    * @param númeroHabitaciones Parámetro que define el número de  
    * habitaciones que tiene una casa en conjunto cerrado  
    * @param númeroBaños Parámetro que define el número de baños  
    * que tiene una casa en conjunto cerrado
```

- \* @param númeroPisos Parámetro que define el número de pisos
- \* que tiene una casa en conjunto cerrado
- \* @param valorAdministración Parámetro que define el valor de
- \* administración para una casa en conjunto cerrado
- \* @param tienePiscina Parámetro que define si una casa en conjunto
- \* cerrado tiene o no piscina
- \* @param tieneCamposDeportivos Parámetro que define si una casa
- \* en conjunto cerrado tiene o no campos deportivos
- \*/

```

public CasaConjuntoCerrado(int identificadorInmobiliario, int área,
String dirección, int númeroHabitaciones, int númeroBaños,
int númeroPisos, int valorAdministración, boolean tienePiscina,
boolean tieneCamposDeportivos) {
// Invoca al constructor de la clase padre
super(identificadorInmobiliario, área, dirección,
númeroHabitaciones, númeroBaños, númeroPisos);
this.valorAdministración = valorAdministración;
this.tienePiscina = tienePiscina;
this.tieneCamposDeportivos = tieneCamposDeportivos;
}

```

/\*\*

- \* Método que muestra en pantalla los datos de una casa en conjunto
- \* cerrado
- \*/

```

void imprimir() {
    super.imprimir(); // Invoca al método imprimir de la clase padre
    System.out.println("Valor de la administración = " + valorAdministración);
    System.out.println("Tiene piscina? = " + tienePiscina);
    System.out.println("Tiene campos deportivos? = " + tieneCamposDeportivos);
}

```

```
        System.out.println();
    }
}
```

## Clase Casa Rural

```
package Inmuebles;
```

```
public class CasaRural extends Casa {

    // Atributo que identifica el valor por área para una casa rural
    protected static double valorArea = 1500000;

    /* Atributo que identifica la distancia a la que se encuentra la casa
    rural de la cabecera municipal */
    protected int distanciaCabera;

    // Atributo que identifica la altitud a la que se encuentra una casa rural
    protected int altitud;

    /**
     * Constructor de la clase CasaRural
     * @param identificadorInmobiliario Parámetro que define el
     * identificador inmobiliario de una casa rural
     * @param área Parámetro que define el área de una casa rural
     * @param dirección Parámetro que define la dirección donde se
     * encuentra localizada una casa rural
     * @param númeroHabitaciones Parámetro que define el número de
     * habitaciones que tiene una casa rural
     * @param númeroBaños Parámetro que define el número de baños
     * que tiene una casa rural
     * @param númeroPisos Parámetro que define el número de pisos
     * que tiene una casa rural
     * @param distanciaCabera Parámetro que define la distancia de la
```



```

* casa rural a la cabecera municipal

* @param altitud Parámetro que define la altitud sobre el nivel del
* mar en que se encuentra una casa rural
*/

public CasaRural(int identificadorInmobiliario, int área, String dirección, int
númeroHabitaciones, int númeroBaños, int númeroPisos, int distanciaCabera, int altitud) {

    // Invoca al constructor de la clase padre

    super(identificadorInmobiliario, área, dirección,númeroHabitaciones, númeroBaños,
númeroPisos);

    this.distanciaCabera = distanciaCabera;

    this.altitud = altitud;

}

/**
* Método que muestra en pantalla los datos de una casa rural
*/

void imprimir() {
    super.imprimir(); // Invoca al método imprimir de la clase padre
    System.out.println("Distancia la cabecera municipal =" +númeroHabitaciones + " km.");
    System.out.println("Altitud sobre el nivel del mar = " + altitud +" metros.");
    System.out.println();
}
}

```

## Clase Casa Urbana

```

package Inmuebles;

public class CasaUrbana extends Casa{

    /**
    * Constructor de la clase CasaUrbana
    * @param identificadorInmobiliario Parámetro que define el

```

- \* identificador inmobiliario de una casa urbana
- \* @param área Parámetro que define el área de una casa urbana
- \* @param dirección Parámetro que define la dirección donde se
- \* encuentra localizada una casa urbana
- \* @param númeroHabitaciones Parámetro que define el número de
- \* habitaciones que tiene una casa urbana
- \* @param númeroBaños Parámetro que define el número de baños
- \* que tiene una casa urbana
- \* @param númeroPisos Parámetro que define el número de pisos
- \* que tiene una casa urbana

\*/

```
public CasaUrbana(int identificadorInmobiliario, int área, String
dirección, int númeroHabitaciones, int númeroBaños, int
númeroPisos) {
```

```
// Invoca al constructor de la clase padre
```

```
super(identificadorInmobiliario, área, dirección,
númeroHabitaciones, númeroBaños, númeroPisos);
```

```
}
```

```
/**
```

- \* Método que muestra en pantalla los datos de una casa urbana

\*/

```
void imprimir() {
```

```
super.imprimir(); // Invoca al método imprimir de la clase padre
```

```
}
```

```
}
```

## Clase Inmueble

```
package Inmuebles;
```

```
public class Inmueble {
```

```
    protected int identificadorInmobiliario;
```

```

protected int área; // Atributo que identifica el área de un inmueble
protected String dirección; /* Atributo que identifica la dirección de un inmueble */
protected double precioVenta; /* Atributo que identifica el precio de venta de un inmueble */
/**
 * Constructor de la clase Inmueble
 * @param identificadorInmobiliario Parámetro que define el
 * identificador de un inmueble
 * @param área Parámetro que define el área de un inmueble
 * @param dirección Parámetro que define la dirección donde se
 * encuentra localizado un inmueble
 */
Inmueble(int identificadorInmobiliario, int área, String dirección) {
    this.identificadorInmobiliario = identificadorInmobiliario;
    this.área = área;
    this.dirección = dirección;
}
/**
 * Método que a partir del valor del área de un inmueble, calcula su
 * precio de venta
 * @param valorArea El valor unitario por área de un determinado
 * inmueble
 * @return Precio de venta del inmueble
 */
double calcularPrecioVenta(double valorArea) {
    precioVenta = área * valorArea;
    return precioVenta;
}
/**
 * Método que muestra en pantalla los datos de un inmueble

```

```

*/
void imprimir() {
    System.out.println("Identificador inmobiliario = " + identificadorInmobiliario);
    System.out.println("Area = " + área);
    System.out.println("Dirección = " + dirección);
    System.out.println("Precio de venta = $" + precioVenta);
}
}

```

## Clase InmuebleVivienda

```

package Inmuebles;

public class InmuebleVivienda extends Inmueble {
    protected int númeroHabitaciones;

    /* Atributo que identifica el número de baños de un inmueble para
    vivienda */
    protected int númeroBaños;

    /**
     * Constructor de la clase InmuebleVivienda
     * @param identificadorInmobiliario Parámetro que define el
     * identificador inmobiliario de un inmueble para la vivienda
     * @param área Parámetro que define el área de un inmueble para la
     * vivienda
     * @param dirección Parámetro que define la dirección donde se
     * encuentra localizado un inmueble para la vivienda
     * @param númeroHabitaciones Parámetro que define el número de
     * habitaciones que tiene un inmueble para la vivienda
     * @param númeroBaños Parámetro que define el número de baños
     * que tiene un inmueble para la vivienda
     */
}

```

```

public InmuebleVivienda(int identificadorInmobiliario, int área, String dirección, int
númeroHabitaciones, int númeroBaños) {

super(identificadorInmobiliario, área, dirección); /* Invoca al
constructor de la clase padre */

this.númeroHabitaciones = númeroHabitaciones;

this.númeroBaños = númeroBaños;

}

/**
 * Método que muestra en pantalla los datos de un inmueble para la
 * vivienda
 */
void imprimir() {
super.imprimir(); // Invoca al método imprimir de la clase padre
System.out.println("Número de habitaciones = " + númeroHabitaciones);
System.out.println("Número de baños = " + númeroBaños);
}
}

```

## Clase Local

```

package Inmuebles;

public class Local extends Inmueble {

    enum tipo {INTERNO,CALLE}; /* Tipo de inmueble especificado
como un valor enumerado */

protected tipo tipoLocal; /* Atributo que identifica el tipo de
inmueble */

/**
 * Constructor de la clase Local
 * @param identificadorInmobiliario Parámetro que define el
 * identificador inmobiliario de un local

```

```

* @param área Parámetro que define el área de un local
* @param dirección Parámetro que define la dirección donde se
* encuentra localizado un local
* @param tipoLocal Parámetro que define el tipo de local (interno o
* que da a la calle)
*/
public Local(int identificadorInmobiliario, int área, String dirección,
tipo tipoLocal) {
// Invoca al constructor de la clase padre
super(identificadorInmobiliario, área, dirección);
this.tipoLocal = tipoLocal;
}
/**
* Método que muestra en pantalla los datos de un local
*/
void imprimir() {
super.imprimir(); // Invoca al método imprimir de la clase padre
System.out.println("tipo de local = " + tipoLocal);
}
}

```

## Clase Local Comercial

```

package Inmuebles;

public class LocalComercial extends Local{

    protected static double valorArea = 3000000;
    /* Atributo que identifica el centro comercial donde está ubicado el
    local comercial */
    protected String centroComercial;

```

```

/**
 * Constructor de la clase LocalComercial
 *
 * @param identificadorInmobiliario Parámetro que define el
 * identificador inmobiliario de un local comercial
 *
 * @param área Parámetro que define el área de un local comercial
 *
 * @param dirección Parámetro que define la dirección donde se
 * encuentra localizado un local comercial
 *
 * @param tipoLocal Parámetro que define el tipo de local comercial
 * (interno o que da a la calle)
 *
 * @param centroComercial Parámetro que define el nombre del
 * centro comercial donde está ubicado el local comercial
 */
public LocalComercial(int identificadorInmobiliario, int área, String
dirección, tipo tipoLocal, String centroComercial) {
    // Invoca al constructor de la clase padre
    super(identificadorInmobiliario, área, dirección, tipoLocal);
    this.centroComercial = centroComercial;
}

/**
 * Método que muestra en pantalla los datos de un local comercial
 */
void imprimir() {
    super.imprimir(); // Invoca al método imprimir de la clase padre
    System.out.println("Centro comercial = " + centroComercial);
    System.out.println();
}
}

```

## Clase Oficina

```
package Inmuebles;
```

```

public class Oficina extends Local {
    protected static double valorArea = 3500000;
    // Atributo que identifica si una oficina pertenece o no al gobierno
    protected boolean esGobierno;
    /**
     * Constructor de la clase Oficina
     * @param identificadorInmobiliario Parámetro que define el
     * identificador inmobiliario de una oficina
     * @param área Parámetro que define el área de una oficina
     * @param dirección Parámetro que define la dirección donde se
     * encuentra localizada una oficina
     * @param tipoLocal Parámetro que define el tipo de una oficina
     * (interna o que da a la calle)
     * @param esGobierno Parámetro que define un valor booleano para
     * determinar si la oficina es del gobierno o no
     */
    public Oficina(int identificadorInmobiliario, int área, String
    dirección, tipo tipoLocal, boolean esGobierno) {
        // Invoca al constructor de la clase padre
        super(identificadorInmobiliario, área, dirección, tipoLocal);
        this.esGobierno = esGobierno;
    }
    /**
     * Método que muestra en pantalla los datos de una oficina
     */
    void imprimir() {
        super.imprimir(); // Invoca al método imprimir de la clase padre
        System.out.println("Es oficina gubernamental = " + esGobierno);
    }
}

```



```
System.out.println();  
}  
}
```

## Clase Prueba

```
package Inmuebles;
```

```
import Inmuebles.Local.tipo;
```

```
public class Prueba {  
    public static void main(String args[]) {  
        ApartamentoFamiliar apto1 = new  
        ApartamentoFamiliar(103067,120,"Avenida Santander 45-45",3,2,200000);  
        System.out.println("Datos apartamento");  
        apto1.calcularPrecioVenta(aptos1.valorArea);  
        apto1.imprimir();  
        System.out.println("Datos apartamento");  
        Apartaestudio aptestudio1 = new  
        Apartaestudio(12354,50,"Avenida Caracas 30-15",1,1);  
        aptestudio1.calcularPrecioVenta(aptestudio1.valorArea);  
        aptestudio1.imprimir();  
  
        CasaConjuntoCerrado casacerr1= new CasaConjuntoCerrado(20,114,"Manchester cra 49 #44-  
47",  
        4,3,2,300000,true,false);  
        System.out.println("Datos Casa conjunto");  
        casacerr1.calcularPrecioVenta(casacerr1.valorArea);  
        casacerr1.imprimir();  
  
        CasaIndependiente casaind1= new CasaIndependiente(2012,90,"la milagrosa cra 19 #94-57",
```

```

4,3,2);

System.out.println("Datos Casa independiente");

casaind1.calcularPrecioVenta(casaind1.valorArea);

casaind1.imprimir();


CasaRural casaRural1= new CasaRural(2012,90,"la milagrosa cra 19 #94-57",
4,3,2,1000,2100);

System.out.println("Datos Casa Rural");

casaRural1.calcularPrecioVenta(casaRural1.valorArea);

casaRural1.imprimir();


tipo t1 =tipo.CALLE;

LocalComercial localComercial1= new LocalComercial(9999,50,"cra 10 # 54-44",t1,"Ciudad
Fabricato");

System.out.println("Datos Local Comercial");

localComercial1.calcularPrecioVenta(localComercial1.valorArea);

localComercial1.imprimir();


tipo t2 = tipo.INTERNO;

Oficina oficina1= new Oficina(5555,15,"calle 56 # 14-85",t2,true);

System.out.println("Datos Local Comercial");

oficina1.calcularPrecioVenta(oficina1.valorArea);

oficina1.imprimir();
}
}

```

## Punto 4.7 (ejercicio 3)

### Clase Animal

```
package Animales;
```

```
public abstract class Animal {  
  
    protected String sonido;  
    protected String alimentos;  
    protected String habitat;  
    protected String nombreCientifico;  
  
    public abstract String getNombreCientifico();  
  
    public abstract String getSonido();  
  
    public abstract String getAlimentos();  
  
    public abstract String getHabitat();  
  
}
```

### **Clase Canido**

```
package Animales;  
  
public abstract class Canido extends Animal{  
  
}
```

### **Clase Felino**

```
package Animales;  
  
public abstract class Felino extends Animal {
```

```
}
```

## **Clase Gato**

```
package Animales;
```

```
public class Gato extends Felino {
```

```
    public String getSonido() {
```

```
        return "Mauullido";
```

```
    }
```

```
    public String getAlimentos() {
```

```
        return "Ratones";
```

```
    }
```

```
    public String getHabitat() {
```

```
        return "Doméstico";
```

```
    }
```

```
    public String getNombreCientifico() {
```

```
        return "Felis silvestris catus";
```

```
    }
```

```
}
```

## **Clase Leon**

```
package Animales;
```

```
public class Leon extends Felino {
```

```
    public String getSonido() {
```

```
        return "Rugido";
```

```
    }
```

```
public String getAlimentos() {  
    return "Carnívoro";  
}  
  
public String getHabitat() {  
    return "Praderas";  
}  
  
public String getNombreCientifico() {  
    return "Panthera leo";  
}  
}
```

## **Clase Lobo**

```
package Animales;  
  
public class Lobo extends Canido {  
  
    public String getNombreCientifico() {  
        return "Canis Lupus";  
    }  
  
    public String getSonido() {  
        return "Aullido";  
    }  
  
    public String getAlimentos() {  
        return "Carnivoro";  
    }  
}
```

```
}

    public String getHabitat() {
        return "Bosque";
    }
}
```

## **Clase Perro**

```
package Animales;

public class Perro extends Canido {

    public String getNombreCientifico() {
        return "Canis lupus familiaris";
    }

    public String getSonido() {
        return "Ladrido";
    }

    public String getAlimentos() {
        return "Carnivoro";
    }

    public String getHabitat() {
        return "Domestico";
    }
}
```

## Clase Test

```
package Animales;

public class Test {

    public static void main(String[] args) {

        Animal[] animales = new Animal[4];

        animales[0] = new Perro();
        animales[1] = new Lobo();
        animales[2] = new Leon();
        animales[3] = new Gato();

        for (int i = 0; i < 4; i++) {

            System.out.println("Nombre cientifico:" + animales[i].getNombreCientifico());

            System.out.println("Sonido: " + animales[i].getSonido());

            System.out.println("Alimento: " + animales[i].getAlimentos());

            System.out.println("Habitat: " + animales[i].getHabitat() + "\n");

        }

    }

}
```

## Punto 4.8(ejercicio 4)

### Clase Ciclista

```
package CarreraCiclistica;

public abstract class Ciclista {

    private int identificador;
```

```
private String nombre;
```

```
private int tiempoAcumulado = 0;
```

```
private int posiciónGeneral;
```

```
public Ciclista(int identificador, String nombre) {
```

```
    this.identificador = identificador;
```

```
    this.nombre = nombre;
```

```
}
```

```
abstract String imprimirTipo();
```

```
protected int getIdentificador() {
```

```
    return identificador;
```

```
}
```

```
protected void setIdentificador(int identificador) {
```

```
    this.identificador = identificador;
```

```
}
```

```
protected String getNombre() {
```

```
    return nombre;
```

```
}
```

```
protected void setNombre(String nombre) {
```

```
    this.nombre = nombre;
```

```
}
```

```
protected int getPosiciónGeneral(int posiciónGeneral) {
```

```
    return posiciónGeneral;
```



```

    }

    protected void setPosiciónGeneral(int posiciónGeneral) {
        this.posiciónGeneral = posiciónGeneral;
    }

    protected int getTiempoAcumulado() {
        return tiempoAcumulado;
    }

    protected void setTiempoAcumulado(int tiempoAcumulado) {
        this.tiempoAcumulado = tiempoAcumulado;
    }

    protected void imprimir() {
        System.out.println("Identificador = " + identificador);
        System.out.println("Nombre = " + nombre);
        System.out.println("Tiempo Acumulado = " + tiempoAcumulado);
    }
}

```

## Clase Contrarrelojista

```

package CarreraCiclistica;

public class Contrarrelojista extends Ciclista {
    private double velocidadMaxima;

    public Contrarrelojista(int identificador, String nombre, double velocidadMaxima) {
        super(identificador, nombre);
        this.velocidadMaxima = velocidadMaxima;
    }
}

```

```
}
```

```
protected double getVelocidadMaxima() {  
    return velocidadMaxima;  
}
```

```
protected void setVelocidadMaxima(double velocidadMaxima) {  
    this.velocidadMaxima = velocidadMaxima;  
}
```

```
protected void imprimir() {  
    super.imprimir();  
    System.out.println("Aceleración promedio = " +  
        velocidadMaxima);  
}
```

```
protected String imprimirTipo() {  
    return "Es un constrarrelojista";  
}  
}
```

## **Clase Equipo**

```
package CarreraCiclistica;
```

```
import java.util.*;
```

```
public class Equipo {  
    private String nombre;  
    private static double totalTiempo;  
    private String pais;
```

```
ArrayList<Ciclista> listaCiclistas;
```

```
public Equipo(String nombre, String país) {  
    this.nombre = nombre;  
    this.país = país;  
    totalTiempo = 0;  
    listaCiclistas = new ArrayList<Ciclista>();  
}
```

```
public String getNombre() {  
    return nombre;  
}
```

```
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}
```

```
private String getPais() {  
    return país;  
}
```

```
private void setPais(String país) {  
    this.país = país;  
}
```

```
void añadirCiclista(Ciclista ciclista) {  
    listaCiclistas.add(ciclista);  
}
```

```
void listarEquipo() {  
    for (int i = 0; i < listaCiclistas.size(); i++) {  
        Ciclista c = (Ciclista) listaCiclistas.get(i);  
        System.out.println(c.getNombre());  
    }  
}
```

```
void buscarCiclista() {  
    Scanner sc = new Scanner(System.in);  
    String nombreCiclista = sc.next();  
    for (int i = 0; i < listaCiclistas.size(); i++) {  
        Ciclista c = (Ciclista) listaCiclistas.get(i);  
        if (c.getNombre().equals(nombreCiclista)) {  
            System.out.println(c.getNombre());  
        }  
    }  
}
```

```
void calcularTotalTiempo() {  
    for (int i = 0; i < listaCiclistas.size(); i++) {  
        Ciclista c = (Ciclista) listaCiclistas.get(i);  
        totalTiempo = totalTiempo + c.getTiempoAcumulado();  
    }  
}
```

```
void imprimir() {  
    System.out.println("Nombre del equipo = " + nombre);  
    System.out.println("País = " + pais);  
}
```

```
        System.out.println("Total tiempo del equipo = " + totalTiempo);
    }
}
```

## **Clase Escalador**

```
package CarreraCiclistica;
```

```
public class Escalador extends Ciclista {

    private double aceleracionPromedio;

    private double gradoRampa;

    public Escalador(int identificador, String nombre, double aceleracionPromedio, double
gradoRampa) {
        super(identificador, nombre);
        this.aceleracionPromedio = aceleracionPromedio;
        this.gradoRampa = gradoRampa;
    }

    protected double getAceleraciónPromedio() {
        return aceleracionPromedio;
    }

    protected void setAceleraciónPromedio(double aceleracionPromedio) {
        this.aceleracionPromedio = aceleracionPromedio;
    }

    protected double getGradoRampa() {
        return gradoRampa;
    }
}
```

```

protected void setGradoRampa(double gradoRampa) {
    this.gradoRampa = gradoRampa;
}

protected void imprimir() {
    super.imprimir();
    System.out.println("Aceleración promedio = " +
        aceleracionPromedio);
    System.out.println("Grado de rampa = " + gradoRampa);
}

protected String imprimirTipo() {
    return "Es un escalador";
}
}

```

## Clase Test

```

package CarreraCiclistica;

public class Test {
    public static void main(String args[]) {
        Equipo equipo1 = new Equipo("Sky", "Estados Unidos");
        Velocista velocista1 = new Velocista(123979, "Geraint Thomas",
            320, 25);
        Escalador escalador1 = new Escalador(123980, "Egan Bernal",
            25, 10);
        Contrarrelojista contrarrelojista1 = new Contrarrelojista(123981,
            "Jonathan Castroviejo", 120);
        equipo1.añadirCiclista(velocista1);
        equipo1.añadirCiclista(escalador1);
    }
}

```

```

        equipo1.añadirCiclista(contrarrelojista1);
        velocista1.setTiempoAcumulado(365);
        escalador1.setTiempoAcumulado(385);
        contrarrelojista1.setTiempoAcumulado(370);
        equipo1.calcularTotalTiempo();
        equipo1.imprimir();
        equipo1.listarEquipo();

    }
}

```

## Clase Velocista

```

package CarreraCiclistica;

public class Velocista extends Ciclista {
    private double potenciaPromedio;

    private double velocidadPromedio;

    public Velocista(int identificador, String nombre, double potenciaPromedio, double
    velocidadPromedio) {
        super(identificador, nombre);
        this.potenciaPromedio = potenciaPromedio;
        this.velocidadPromedio = velocidadPromedio;
    }

    protected double getPotenciaPromedio() {
        return potenciaPromedio;
    }
}

```

```
protected void setPotenciaPromedio(double potenciaPromedio) {  
    this.potenciaPromedio = potenciaPromedio;  
}
```

```
protected double getvelocidadPromedio() {  
    return velocidadPromedio;  
}
```

```
protected void setVelocidadPromedio(double velocidadPromedio) {  
    this.velocidadPromedio = velocidadPromedio;  
}
```

```
protected void imprimir() {  
    super.imprimir();  
    System.out.println("Potencia promedio = " + potenciaPromedio);  
    System.out.println("Velocidad promedio = " +  
        velocidadPromedio);  
}
```

## Punto 8.1(ejercicio 5)

### Clase VentanaPrincipal

```
import java.awt.*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import javax.swing.*;  
  
public class VentanaPrincipal extends JFrame implements  
    ActionListener {  
    private ListaPersonas lista; // El objeto ListaPersonas de la aplicación
```



```

private Container contenedor; /*
    * Un contenedor de elementos
    * gráficos
    */

// Etiquetas estáticas para los nombres de los atributos
private JLabel nombre, apellidos, teléfono, dirección;

// Campos de ingreso de texto
private JTextField campoNombre, campoApellidos, campoTeléfono,
    campoDirección;

private JButton añadir, eliminar, borrarLista; // Botones
private JList listaNombres; // Lista de personas
private DefaultListModel modelo; // Objeto que modela la lista
private JScrollPane scrollLista; // Barra de desplazamiento vertical

/**
 * Constructor de la clase VentanaPrincipal
 */
public VentanaPrincipal() {
    lista = new ListaPersonas(); // Crea la lista de personas
    inicio();

    setTitle("Personas"); // Establece el título de la ventana
    setSize(270, 350); // Establece el tamaño de la ventana
    setLocationRelativeTo(null); /*
        * La ventana se posiciona en el
        * centro de la pantalla
        */

    // Establece que el botón de cerrar permitirá salir de la aplicación
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    setResizable(false); /*

```

```

        * Establece que el tamaño de la ventana no
        * se puede cambiar
        */
    }

    /**
     * Método que crea la ventana con sus diferentes componentes
     * gráficos
     */
    private void inicio() {
        contenedor = getContentPane(); /*
            * Obtiene el panel de
            * contenidos de la ventana
            */
        contenedor.setLayout(null); /*
            * Establece que el panel no tiene
            * asociado ningún layout
            */

        // Establece la etiqueta y el campo nombre
        nombre = new JLabel();
        nombre.setText("Nombre:");
        nombre.setBounds(20, 20, 135, 23); /*
            * Establece la posición de
            * la etiqueta nombre
            */

        campoNombre = new JTextField();

        // Establece la posición del campo de texto nombre
        campoNombre.setBounds(105, 20, 135, 23);

        // Establece la etiqueta y el campo apellidos

```

```

apellidos = new JLabel();
apellidos.setText("Apellidos:"); /*
    * Establece la posición de la
    * etiqueta apellidos
    */
apellidos.setBounds(20, 50, 135, 23);
campoApellidos = new JTextField();
// Establece la posición del campo de texto apellidos
campoApellidos.setBounds(105, 50, 135, 23);
// Establece la etiqueta y el campo teléfono
teléfono = new JLabel();
teléfono.setText("Teléfono:");
teléfono.setBounds(20, 80, 135, 23); /*
    * Establece la posición de
    * la etiqueta teléfono
    */
campoTeléfono = new JTextField();
// Establece la posición del campo de texto teléfono
campoTeléfono.setBounds(105, 80, 135, 23);
// Establece la etiqueta y el campo dirección
dirección = new JLabel();
dirección.setText("Dirección:");
dirección.setBounds(20, 110, 135, 23); /*
    * Establece la posición
    * de la etiqueta dirección
    */
campoDirección = new JTextField();
// Establece la posición del campo de texto dirección
campoDirección.setBounds(105, 110, 135, 23);

```

```

// Establece el botón Añadir persona
añadir = new JButton();
añadir.setText("Añadir");
añadir.setBounds(105, 150, 80, 23); /*
                                * Establece la posición del
                                * botón Añadir persona
                                */

/*
* Agrega al botón un ActionListener para que gestione eventos
* del botón
*/
añadir.addActionListener(this);

// Establece el botón Eliminar persona
eliminar = new JButton();
eliminar.setText("Eliminar");
eliminar.setBounds(20, 280, 80, 23); /*
                                * Establece la posición del
                                * botón Eliminar persona
                                */

/*
* Agrega al botón un ActionListener para que gestione eventos
* del botón
*/
eliminar.addActionListener(this);

// Establece el botón Borrar lista
borrarLista = new JButton();
borrarLista.setText("Borrar Lista");
borrarLista.setBounds(120, 280, 120, 23); /*
                                * Establece la

```

```

        * posición del botón Borrar lista
    */

/*
    * Agrega al botón un ActionListener para que gestione eventos
    * del botón
    */
borrarLista.addActionListener(this);

// Establece la lista gráfica de personas
listaNombres = new JList();

/*
    * Establece que se pueda seleccionar solamente un elemento de
    * la lista
    */
listaNombres.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
modelo = new DefaultListModel();

// Establece una barra de desplazamiento vertical
scrollLista = new JScrollPane();

// Establece la posición de la barra de desplazamiento vertical
scrollLista.setBounds(20, 190, 220, 80);

// Asocia la barra de desplazamiento vertical a la lista de personas
scrollLista.setViewportView(listaNombres);

// Se añade cada componente gráfico al contenedor de la ventana
contenedor.add(nombre);
contenedor.add(campoNombre);
contenedor.add(apellidos);
contenedor.add(campoApellidos);
contenedor.add(teléfono);
contenedor.add(campoTeléfono);
contenedor.add(dirección);

```

```

        contenedor.add(campoDirección);

        contenedor.add(añadir);

        contenedor.add(eliminar);

        contenedor.add(borrarLista);

        contenedor.add(scrollLista);

    }

    /**
     * Método que gestiona los eventos generados en la ventana principal
     */
    @Override
    public void actionPerformed(ActionEvent evento) {

        if (evento.getSource() == añadir) { // Si se pulsa el botón añadir
            añadirPersona(); // Se invoca añadir persona
        }

        if (evento.getSource() == eliminar) { /*
            * Si se pulsa el botón
            * eliminar
            */

            /**
             * Se invoca el método eliminarNombre que elimina el
             * elemento seleccionado
             */

            eliminarNombre(listaNombres.getSelectedIndex());

            if (evento.getSource() == borrarLista) { /*
                * Si se pulsa el botón
                * borrar lista
                */

```

```

        borrarLista(); // Se invoca borrar lista
    }
}

/**
 * Método que agrega una persona al vector de personas y a la lista
 * gráfica de personas
 */
private void añadirPersona() {
    /*
     * Se obtienen los campos de texto ingresados y se crea una
     * persona
     */
    Persona p = new Persona(campoNombre.getText(),
        campoApellidos.getText(),
        campoTeléfono.getText(), campoDirección.getText());
    lista.añadirPersona(p); /*
        * Se añade una persona al vector de
        * personas
        */
    String elemento = campoNombre.getText() + "-" +
        campoApellidos.getText() +
        "-" + campoTeléfono.getText() + "-" + campoDirección.getText();
    modelo.addElement(elemento); /*
        * Se agrega el texto con los
        * datos de la persona al JList
        */
    listaNombres.setModel(modelo);
}

```

```

// Se colocan todos los campos de texto nulos
campoNombre.setText("");
campoApellidos.setText("");
campoTeléfono.setText("");
campoDirección.setText("");
}

/**
 * Método que elimina una persona del vector de personas y de la
 * lista gráfica de personas en la ventana
 *
 * @param indice Parámetro que define la posición de la persona a
 *             eliminar
 */
private void eliminarNombre(int indice) {
    if (indice >= 0) { // Si la posición existe
        modelo.removeElementAt(indice); /*
            * Se
            * lista.eliminarPersona(indice); /* Se elimina la persona
            * seleccionada del vector de personas
            */
    } else { /*
        * Si no se seleccionó ninguna persona, se genera un
        * mensaje de error
        */
        JOptionPane.showMessageDialog(null, "Debe seleccionar un elemento", "Error",
            JOptionPane.ERROR_MESSAGE);
    }
}
}

```



```

/**
 * Método que elimina todas las personas del vector de personas
 */
private void borrarLista() {
    lista.borrarLista(); // Se eliminan todas las personas del vector
    modelo.clear(); // Limpia el JList, la lista gráfica de personas
}
}

```

## Clase Persona

```

public class Persona {
    String nombre;
    String apellidos;
    String teléfono;
    String dirección;

    public Persona(String nombre, String apellidos, String teléfono,
        String dirección) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.teléfono = teléfono;
        this.dirección = dirección;
    }
}

```

## Clase Lista Personas

```

import java.util.*;

public class ListaPersonas {

```

```
Vector listaPersonas;
```

```
public ListaPersonas() {  
    listaPersonas = new Vector();  
}
```

```
public void añadirPersona(Persona p) {  
    listaPersonas.add(p);  
}
```

```
public void eliminarPersona(int i) {  
    listaPersonas.removeElementAt(i);  
}
```

```
public void borrarLista() {  
    listaPersonas.removeAllElements();  
}  
}
```

## **Clase Main**

```
public class Main {  
    public static void main(String[] args) {  
        VentanaPrincipal miVentanaPrincipal;  
        miVentanaPrincipal = new VentanaPrincipal();  
        miVentanaPrincipal.setVisible(true);  
    }  
}
```

## **Enlaces Videos de YouTube códigos funcionando**

PUNTO 1: <https://youtu.be/bdqUhZgZ4Lo>

PUNTO 2: <https://youtu.be/Seidsckmf8c>

PUNTO 3: <https://youtu.be/s5PhLQ1lea8>

PUNTO 4: <https://youtu.be/7NlfujV6wWc>

PUNTO 5: <https://youtu.be/6GjvHoUdtVA>