



Algoritmos y Estructura de Datos - TI

ÍNDICE

Presentación	5
Red de contenidos	6
Sesiones de aprendizaje	

UNIDAD DE APRENDIZAJE 1

SEMANA 1	: Encapsulamiento: conceptos básicos.	7
SEMANA 2	: Encapsulamiento: control de acceso.	13
SEMANA 3	: Modificador static y referencia this	19

UNIDAD DE APRENDIZAJE 2

SEMANA 4	: Manejo de arreglos: Arreglos unidimensionales	37
SEMANA 5	: Manejo de arreglos: Arreglos biidimensionales	49
SEMANA 6	: Manejo de arreglos: Arreglo de objetos	63

SEMANA 7 : SEMANA DE EXÁMENES PARCIALES

UNIDAD DE APRENDIZAJE 3

SEMANA 8	: Clase ArrayList: métodos y operaciones variadas	75
SEMANA 9	: Clase ArrayList : métodos y operaciones variadas	75
SEMANA 10	: Clase ArrayList y Archivos de texto	89

UNIDAD DE APRENDIZAJE 4

SEMANA 11	: Herencia y polimorfismo: Modificador protected, relación es-un.	103
SEMANA 12	: Herencia y polimorfismo: Sobre-escritura, clases y métodos abstractos.	113
SEMANA 13	: Herencia y polimorfismo: relación tiene-un.	113
SEMANA 14	: Herencia y polimorfismo: polimorfismo y enlace dinámico	125

UNIDAD DE APRENDIZAJE 5

SEMANA 15	: Interfaces	137
SEMANA 16	: Interfaces	137
SEMANA 17	: SEMANA DE EXÁMENES FINALES	
ANEXO	: Caso práctico	149

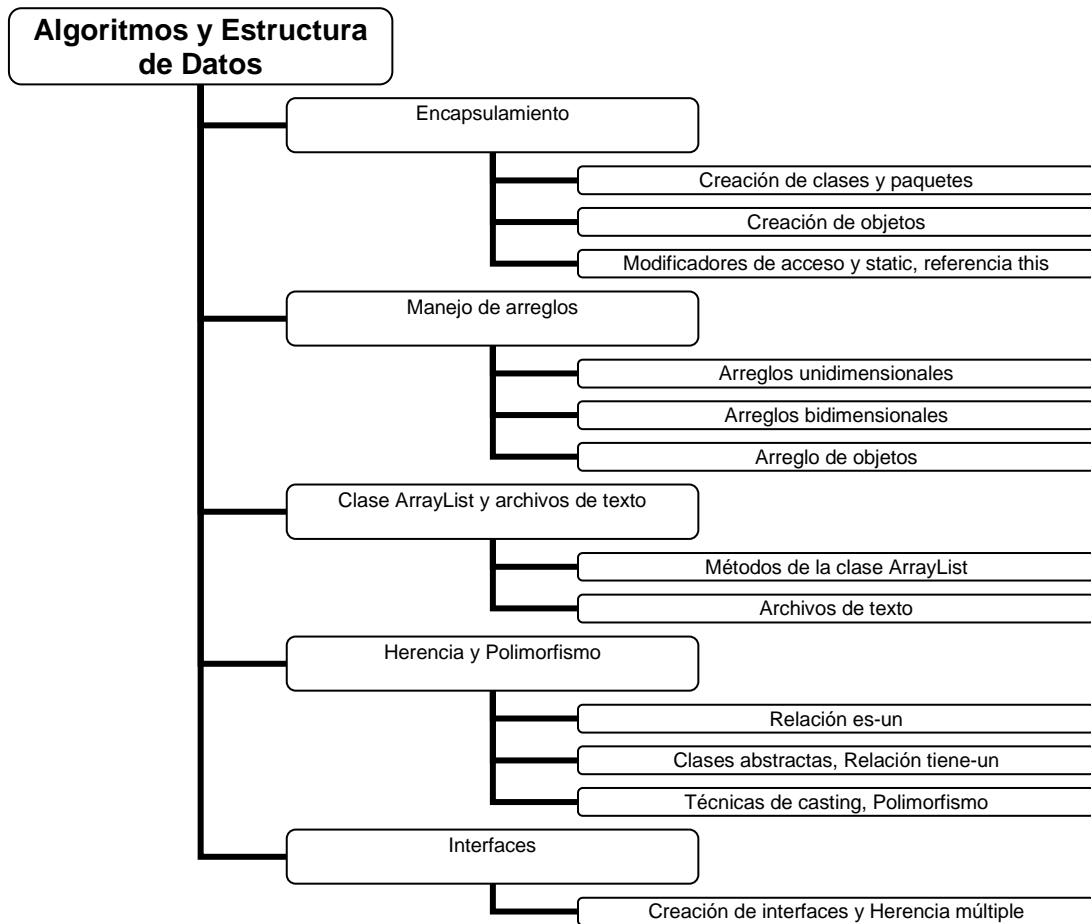
PRESENTACIÓN

Algoritmos y Estructura de Datos pertenece a la línea de programación y desarrollo de aplicaciones y se dicta en las carreras de Computación e Informática, Redes y Comunicaciones, Administración y Sistemas, y Electrónica. Brinda un conjunto de técnicas de programación que permite a los alumnos diseñar algoritmos apropiados y adquirir buenos hábitos de programación.

El manual para el curso ha sido diseñado bajo la modalidad de unidades de aprendizaje, las que se desarrollan durante semanas determinadas. En cada una de ellas, hallará los logros que debe alcanzar al final de la unidad; el tema tratado, el cual será ampliamente desarrollado; y los contenidos, que debe desarrollar, es decir, los subtemas. Por último, encontrará las actividades que deberá desarrollar en cada sesión, que le permitirán reforzar lo aprendido en la clase.

El curso es teórico práctico. Está basado en el paradigma de la programación orientada a objetos. En primer lugar, se inicia con la creación de clases y objetos. Continúa con el manejo de arreglos. Se utiliza la clase ArrayList así como el manejo de archivos de texto. Luego, se desarrollan aplicaciones donde se plasma el concepto de herencia y polimorfismo empleando clases abstractas. Se concluye con la implementación de Interfaces.

RED DE CONTENIDOS



**UNIDAD DE
APRENDIZAJE****1****SEMANA****1-2**

ENCAPSULAMIENTO

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos crean clases agrupadas en paquetes, crean objetos, emplean los modificadores de acceso: public y private, así como el modificador static y la referencia this en Java. Finalmente, aplican el concepto de encapsulamiento.

TEMARIO

- Creación de clases y paquetes
- Creación de objetos
- Modificadores

ACTIVIDADES PROPUESTAS

- Los alumnos crean clases agrupadas en paquetes.
- Los alumnos crean objetos de diversas clases.
- Los alumnos emplean y diferencian los modificadores de acceso: public y private.
- Los alumnos realizan diversas operaciones con los objetos creados.

1. CONCEPTOS BASICOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

1.1 PAQUETE

Un paquete es un conjunto de clases agrupadas que guardan una relación entre sí. Los paquetes se declaran utilizando la palabra **package** seguida del nombre del paquete. Esta instrucción debe colocarse al inicio del archivo y debe ser la primera sentencia que debe ejecutarse. Ejemplo:

```
package semana1;  
public class Alumno{  
    ...  
}
```

Al crearse un paquete se crea una carpeta con el mismo nombre del paquete. En nuestro ejemplo se creará la carpeta semana1. Si coloca la misma instrucción package al inicio de otras clases logrará agrupar varias clases en el mismo paquete.

Uso de la sentencia import

La sentencia import se utiliza para incluir una lista de paquetes en donde se buscará una determinada clase. Su aplicación se aprecia cuando se hace referencia a una clase desde otra que se encuentra fuera del paquete.

La sintaxis es:

```
import nombre_paquete.nombre_clase;
```

Ejemplo:

```
import semana1.Alumno;
```

De esta forma, se tiene acceso a todos los miembros (variables y métodos) de la clase Alumno que se encuentra en el paquete semana1 desde una clase que se encuentra fuera de dicho paquete. En la sentencia import se puede emplear el caracter Asterisco “ * ” con lo cual se estaría indicando que se importe todas las clases del paquete.

Ejemplo:

```
import semana1.*;
```

1.2 CLASE

Una clase es una plantilla que especifica los atributos y el comportamiento de un determinado tipo de objeto. Los atributos se especifican mediante variables (*variables miembro*) y el comportamiento mediante métodos (*métodos miembro*).

En la Programación Orientada a Objetos (POO) una clase define un nuevo tipo de dato. Una vez definido, este nuevo tipo se puede utilizar para crear objetos.

1.2.1 Declaración y definición de una clase

En su forma más simple una clase se declara y define de la siguiente manera:

```
package nomnre_paquete;  
public class NombreClase {  
    // Declaración de variables miembro  
    public tipo1 variable1;  
    public tipo2 variable2;  
    .  
    .  
    .  
  
    // Definición del constructor  
    public NombreClase (parámetros) {  
    }  
  
    // Definición de métodos miembro  
    public tipo1 metodo1(parámetros) {  
    }  
  
    public tipo2 metodo2(parámetros) {  
    }  
    .  
    .  
    .  
}
```

1.2.2 Constructor

Un constructor es un método especial que se utiliza para inicializar un objeto inmediatamente después de su creación.

El constructor de una clase se caracteriza por tener el mismo nombre de su clase y no tener tipo de retorno.

Si una clase no define un constructor, Java crea un constructor por defecto que no tiene parámetros y que no realiza ninguna inicialización especial.

1.3 OBJETO

- Un objeto es una instancia (o ejemplar) de una clase.
- La JVM carga a la memoria el código de bytes de una clase en el primer momento en que la clase sea mencionada durante la ejecución del programa. En adelante, la clase cargada, será utilizada como una fábrica de objetos.
- Cada vez que se crea un objeto se crea una copia de cada una de las variables miembro declaradas por su clase. Estas variables, propias de cada objeto, se denominan: **variables de instancia**.
- En cambio, los métodos, son compartidos por todos los objetos de la misma clase desde el código de la clase previamente cargada.

1.3.1 Creación de objetos

Se siguen los siguientes pasos:

- Se declara una variable referencia que almacenará la dirección de memoria del objeto a crear.

```
nombreClase nombreReferencia;
```

- Se crea el objeto y se asigna su dirección de memoria a la variable referencia. Para esto se usa el operador **new**. El operador **new** crea el objeto dinámicamente (en tiempo de ejecución) y devuelve la dirección de memoria del objeto creado.

```
nombreReferencia = new nombreClase( lista de argumentos );
```

El nombre de la clase seguido por los paréntesis invoca al **constructor** de la clase.

- Los dos pasos anteriores pueden efectuarse en un solo paso de la siguiente manera:

```
nombreClase nombreReferencia = new nombreClase( lista de argumentos );
```

2. ACCESO A LOS MIEMBROS DE UNA CLASE

Para acceder a un miembro de una clase se escribe el nombre de la variable referencia, un punto y el nombre del miembro (*variable miembro* o *método miembro*) al que se desea acceder.

Ejemplo 1:

- Declare la **clase Alumno** dentro del **paquete semana1** que permita registrar un nombre y dos notas. Además, debe crear un método promedio que retorne la nota promedio.

```

package semana1;

public class Alumno {
    // Variables miembro
    public String nombre;
    public int nota1, nota2;

    // Operaciones
    public double promedio() {
        return (nota1 + nota2) / 2.0;
    }
}

```

- En la **clase principal** (clase donde se ha creado la GUI) **crea un objeto** de la clase *Alumno*, y luego de ingresar sus datos visualice la data ingresada, además del promedio obtenido.

```

void procesar() {
    semana1.Alumno a = new semana1.Alumno();
    a.nombre = "Juan";
    a.nota1 = 13;
    a.nota2 = 15;
    listado(a);
}

void listado(semana1.Alumno x) {
    imprimir("nombre : " + x.nombre);
    imprimir("nota 1 : " + x.nota1);
    imprimir("nota 2 : " + x.nota2);
    imprimir("promedio : " + x.promedio());
}

```

➔

nombre	: Juan
nota 1	: 13
nota 2	: 15
promedio	: 14.0

Importante:

Como la clase *Alumno* se encuentra dentro del paquete *semana1* y la clase principal esta fuera de dicho paquete se le debe anteponer al nombre de la clase el nombre del paquete tal como se aprecia en el ejemplo 1: **semana1.Alumno**. Se habrá dado cuenta que realizar esta acción se vuelve tedioso. Por lo tanto, lo más simple sería importar el paquete *semana1* para que el compilador de Java ubique a la clase *Alumno* y de esta forma evitar la referencia al paquete cada vez que se hace referencia a la clase. En este ejemplo, se debe incluir en las sentencias de importación la siguiente línea:

```
import semana1.Alumno;
```

Ahora sí, el código de la clase principal quedaría de este forma:

```

void procesar() {
    Alumno a = new Alumno();
    a.nombre = "Juan";
    a.nota1 = 13;
    a.nota2 = 15;
    listado(a);
}

void listado(Alumno x) {
    imprimir("nombre : " + x.nombre);
    imprimir("nota 1 : " + x.nota1);
    imprimir("nota 2 : " + x.nota2);
    imprimir("promedio : " + x.promedio());
}

```

```

➔  nombre      : Juan
    nota 1     : 13
    nota 2     : 15
    promedio   : 14.0

```

En los siguientes ejemplos, se asume que se ha importado el paquete correspondiente.

Ejemplo 2:

- Añada un constructor a la clase *Alumno* para inicializar sus *variables miembro* al momento de crear el objeto.

```

package semanal;
public class Alumno {
    // Variables miembro
    public String nombre;
    public int nota1, nota2;

    // Constructor
    public Alumno(String nom, int n1, int n2) {
        nombre = nom;
        nota1 = n1;
        nota2 = n2;
    }

    // Operaciones
    public double promedio() {
        return (nota1 + nota2) / 2.0;
    }
}

```

- A la pulsación de un botón cree el objeto y, a la vez, inicialice sus *variables miembro*.

```

void procesar() {
    Alumno a = new Alumno("Juan", 13, 15);
    listado(a);
}

```

```
void listado(Alumno x){
    imprimir("Objeto:"+x);
    imprimir("nombre:"+x.nombre);
    imprimir("nota 1:"+x.nota1);
    imprimir("nota 2:"+x.nota2);
    imprimir("promedio:"+x.promedio());
}
```

➔ Objeto : semanal.Alumno@92e78c
 nombre : Juan
 nota 1 : 13
 nota 2 : 15
 promedio : 14.0

3. CONTROL DE ACCESO Y ENCAPSULAMIENTO

Una clase puede controlar qué partes de un programa pueden acceder a los miembros de su clase: *variables miembro* y *métodos miembro*. Una clase bien diseñada impide el acceso directo a sus *variables miembro* proporcionando a cambio un conjunto de **métodos de acceso** que sirvan como intermediarios entre las *variables miembro* y el mundo exterior. Esto permite controlar el uso correcto de las *variables miembro*, pues los *métodos de acceso* pueden actuar como filtros que prueben los datos que se pretenden ingresar a las *variables miembro*. Por otra parte, algunos métodos de la clase pueden ser necesarios sólo desde el interior de la clase por lo que deben quedar restringidos sólo para uso interno.

Para controlar el acceso a los miembros de una clase, se usan *especificadores o modificadores de acceso* que se anteponen a las declaraciones de los miembros a controlar. Los especificadores de acceso son **public**, **private** y **protected**. Se ve la aplicación del especificador de acceso *protected* cuando se trabaja con herencia, por lo que lo veremos más adelante. En la tabla que sigue, se muestra el uso de los especificadores de acceso.

Especificador	Acceso a los miembros			
	Desde la misma clase	Desde una subclase	Desde una clase del mismo paquete	Desde el exterior de la misma clase
public	X	X	X	X
private	X			
protected	X	X	X	
sin especificador	X		X	

La primera columna indica si la propia clase tiene acceso al miembro definido por el especificador de acceso. La segunda columna indica si las subclases de la clase (sin importar dentro de qué paquete se encuentren estas) tienen acceso a los miembros. La tercera columna indica si las clases del mismo paquete que la clase (sin importar su parentesco) tienen acceso a los miembros. La cuarta columna indica si todas las clases tienen acceso a los miembros.

Encapsulamiento es la capacidad de ocultar los detalles internos del comportamiento de una clase y exponer o dar a conocer sólo los detalles que sean necesarios para el resto de clases. Este ocultamiento nos permite **restringir** y **controlar** el uso de la

clase. Restringir, porque habrán ciertos atributos y métodos privados o protegidos y controlar, porque habrán ciertos mecanismos para modificar el estado de nuestra clase como, por ejemplo, los métodos de acceso.

Ejemplo 3:

Haga que las *variables miembro* de la clase *Alumno* sean de uso privado y declare sus respectivos métodos de acceso: get/set.

```
package semana2;
public class Alumno {
    // Variables miembro
    private String nombre;
    private int nota1, nota2;

    // Constructor
    public Alumno(String nom, int n1, int n2) {
        nombre = nom;
        nota1 = n1;
        nota2 = n2;
    }

    // Métodos de acceso: get

    public String getNombre() {
        return nombre;
    }

    public int getNota1() {
        return nota1;
    }

    public int getNota2() {
        return nota2;
    }

    // Métodos de acceso: set

    public void setNombre(String nom) {
        nombre = nom;
    }

    public void setNota1(int n) {
        nota1 = n;
    }

    public void setNota2(int n) {
        nota2 = n;
    }

    // Operaciones
    public double promedio() {
        return (nota1 + nota2) / 2.0;
    }
}
```

- A la pulsación de un botón, cree el objeto y, a la vez, inicialice sus *variables miembro*. Luego, modifique sus notas.

```
void procesar() {
    Alumno a = new Alumno("Juan",13,15);
    listado(a);
    a.setNota1(19);
    a.setNota2(18);
    listado(a);
}

void listado(Alumno x) {
    imprimir("nombre : " + x.getNombre());
    imprimir("nota 1 : " + x.getNota1());
    imprimir("nota 2 : " + x.getNota2());
    imprimir("promedio : " + x.promedio());
}
```

→ nombre : Juan
 nota 1 : 13
 nota 2 : 15
 promedio : 14.0

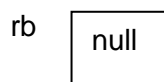
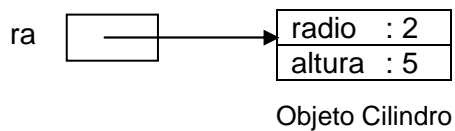
nombre : Juan
 nota 1 : 19
 nota 2 : 18
 promedio : 18.5

4. ASIGNACIÓN ENTRE VARIABLES REFERENCIA

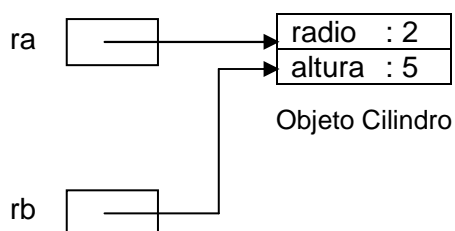
Cuando se asigna la variable referencia **ra** a la variable referencia **rb**, se asigna la dirección de memoria contenida en **ra**. Luego de ello, tanto **ra** como **rb** referencian (controlan) al mismo objeto.

Ejemplo 4:

```
Cilindro ra = new Cilindro(2,5);
Cilindro rb;
```



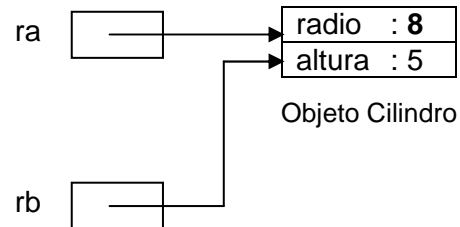
```
rb = ra;
```



Note que a partir de este momento tanto **ra** como **rb** se refieren al mismo objeto.

```
rb.setRadio(8);
```

Cambia por 8 el valor del radio del objeto a través de la referencia **rb**.



```
imprimir("Radio : " + ra.getRadio());
```

Extrae el valor del radio del objeto a través de la referencia **ra** e imprime 8.

Ejemplo 5:

- A la pulsación de un botón, cree el objeto **a** de la clase Alumno inicializando sus *variables miembro*. Luego, a través de la referencia **b**, modifique las notas.

```
void procesar() {
    Alumno a = new Alumno("Juan",13,15);
    listado(a);
    Alumno b = a;
    b.setNota1(19);
    b.setNota2(18);
    listado(a);
}
```

➔ nombre : Juan
 nota 1 : 13
 nota 2 : 15
 promedio : 14.0

nombre : Juan
 nota 1 : 19
 nota 2 : 18
 promedio : 18.5

Ejercicios

- 1) Cree la **clase empleado** dentro del paquete semana2 y declare los siguientes atributos como privados: codigo (int), nombre (String), horas (int), tarifa (double) e implemente los siguientes métodos:

- a) Un constructor que inicialice a los atributos
- b) Métodos de acceso: set/get para cada atributo
- c) Un método que retorne el sueldo bruto (horas*tarifa)
- d) Un método que retorne el descuento (11% del sueldo bruto)
- e) Un método que retorne el sueldo neto (sueldo bruto - descuento)

En la **clase principal** (donde esta la GUI), implemente lo siguiente:

- a) Cree un objeto de tipo empleado con datos fijos.
- b) Cree un método listar que imprima todos los atributos del empleado y el sueldo neto.
- c) Incremente la tarifa del empleado en 10% e imprima su nueva tarifa.

- 2) Cree la **clase producto** dentro del paquete semana2 y declare los siguientes atributos como privados: codigo (int), descripción (String), cantidad (int), precio unitario en soles (double) e implemente los siguientes métodos:

- a) Un constructor que inicialice a los atributos
- b) Métodos de acceso: set/get para cada atributo
- c) Un método que retorne el importe a pagar soles (cantidad * precio unit)
- d) Un método que retorne el importe a pagar dólares. El método recibirá como parámetro el tipo de cambio.

En la **clase principal** (donde está la GUI), implemente lo siguiente:

- a) Cree un objeto de tipo producto cada vez que se pulse el botón procesar. Los datos serán capturados de la GUI. Asuma la existencia de los métodos para la captura de datos.
- b) Cree un método listar que imprima todos los atributos del producto y el importe a pagar en dólares.
- c) Disminuya el precio unitario del producto en 7% e imprima su nuevo precio unitario.



MODIFICADOR STATIC Y REFERENCIA THIS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos crean clases clasificadas en paquetes, crean objetos, emplean los modificadores de acceso: public y private así como el modificador static y la referencia this en Java. Finalmente, aplican el concepto de encapsulamiento.

TEMARIO

- Modificador static
- Referencia this
- Sobrecarga de constructores
- Inicializadores estáticos

ACTIVIDADES PROPUESTAS

- Los alumnos emplean el modificador static y la referencia this.
- Los alumnos entienden el concepto de sobrecarga.
- Los alumnos reconocen una librería empleando métodos estáticos
- Los alumnos emplean inicializadores estáticos.

1. MODIFICADOR: STATIC

Una *variable de instancia* es una variable de la que se crea una copia independiente para cada objeto de la clase. En cambio, una *variable de clase* es una variable de la que se crea una sola copia para todos los objetos de la clase. Para que una variable sea variable de clase hay que preceder su declaración con el modificador **static**.

Las *variables de clase* se crean en el momento en que la clase se carga a la memoria, lo que ocurre antes de crear cualquier objeto de la clase. Esto significa que las *variables de clase* pueden existir aún cuando no exista objeto de la clase. Se puede acceder a una *variable de clase* pública a través de una referencia a un objeto de la clase o mediante el nombre de la clase.

Ejemplo 1

Haga que la clase *Alumno* cuente la cantidad de objetos creados.

```
package semana3;
public class Alumno {
    // Variables miembro
    private String nombre;
    private int nota1, nota2;
    public static int cantidad = 0;

    // Constructor
    public Alumno(String nom, int n1, int n2) {
        nombre = nom;
        nota1 = n1;
        nota2 = n2;
        cantidad++;
    }

    // Métodos de acceso: get/set
    public String getNombre() {
        return nombre;
    }

    public int getNota1() {
        return nota1;
    }

    public int getNota2() {
        return nota2;
    }

    public void setNota1(int n) {
        nota1 = n;
    }

    public void setNota2(int n) {
        nota2 = n;
    }
}
```

```

        // Operaciones
        public double promedio() {
            return (nota1 + nota2) / 2.0;
        }
    }

```

- A la pulsación de un botón, cree tres objetos y visualice sus contenidos. Muestre, además, la cantidad de objetos.

```

void procesar() {
    Alumno a1 = new Alumno("Juan",13,15),
           a2 = new Alumno("Pedro",16,17),
           a3 = new Alumno("María",14,20);
    listado(a1);
    listado(a2);
    listado(a3);
    imprimir("# objetos creados : " + Alumno.cantidad);
}

```

```

➔ nombre : Juan
   nota 1 : 13
   nota 2 : 15
   promedio : 14.0

   nombre : Pedro
   nota 1 : 16
   nota 2 : 17
   promedio : 16.5

   nombre : María
   nota 1 : 14
   nota 2 : 20
   promedio : 17.0

# objetos creados : 3

```

2. REFERENCIA: THIS

Dentro de un método de la clase, la referencia **this** contiene la dirección de memoria del objeto que invocó al método. La referencia **this** es un parámetro oculto añadido por el compilador.

Una de sus aplicaciones más comunes es cuando el nombre de un parámetro coincide con el nombre de una variable miembro. En este caso, su objetivo es diferenciar la variable miembro del parámetro en sí.

Ejemplo 2

Haga que los parámetros del constructor y de los métodos set tengan el mismo nombre que las *variables miembro*. Adicionalmente, agregue un método que calcule el mayor promedio entre 2 objetos de tipo Alumno.

```

package semana3;
public class Alumno {
    // Variables miembro
    private String nombre;
    private int notal, nota2;
    public static int cantidad = 0;

    // Constructor
    public Alumno(String nombre, int notal, int nota2) {
        this.nombre = nombre;
        this.notal = notal;
        this.nota2 = nota2;
        cantidad++;
    }

    // Métodos de acceso: get/set
    public String getNombre() {
        return nombre;
    }

    public int getNotal() {
        return notal;
    }

    public int getNota2() {
        return nota2;
    }

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

    public void setNotal(int notal) {
        this.notal = notal;
    }

    public void setNota2(int nota2) {
        this.nota2 = nota2;
    }

    // Operaciones
    public double promedio() {
        return (notal + nota2) / 2.0;
    }

    public Alumno mayorPromedio(Alumno obj){
        /* this contiene la referencia del objeto que invocó
        al método */
        if(this.promedio()>obj.promedio())
            return this;
        else
            return obj;
    }
}

```

- A la pulsación de un botón, cree dos objetos y visualice sus contenidos. Muestre, además, la cantidad de objetos y el nombre del alumno con mayor promedio.

```
void procesar(){
    Alumno a1 = new Alumno("Juan",15,14),
    a2 = new Alumno("Pedro",12,13);
    listado(a1);
    listado(a2);
    imprimir("# objetos creados : " + Alumno.cantidad);
    Alumno objMaxProm=a1.mayorPromedio(a2);
    imprimir("El alumno con mayor promedio es:"+
    objMaxProm.getNombre());
}

void listado(Alumno x){
    imprimir("Objeto:"+x);
    imprimir("Nombre:"+x.getNombre());
    imprimir("Nota1:"+x.getNota1());
    imprimir("Nota2:"+x.getNota2());
    imprimir("Promedio:"+x.promedio());
    imprimir("");
}
```

3. SOBRECARGA DE MÉTODOS

La sobrecarga de métodos consiste en crear varios métodos con el mismo nombre. Para diferenciar un método de otro, se utiliza el número y tipo de parámetros que tiene el método y no su tipo de retorno. Los constructores también pueden ser sobrecargados.

Ejemplo 3

Haga que las *variables miembro* de la clase *Alumno* se inicialicen por defecto y, luego, considere tres constructores.

```
package semana3;
public class Alumno {
    // Variables miembro
    private String nombre = "ninguno";
    private int nota1 = 88, nota2 = 88;
    public static int cantidad = 0;

    // Constructores
    public Alumno(String nombre, int nota1, int nota2) {
        this.nombre = nombre;
        this.nota1 = nota1;
        this.nota2 = nota2;
        cantidad++;
    }
    public Alumno(String nombre) {
        this.nombre = nombre;
        cantidad++;
    }
}
```

```

public Alumno() {
    /* En este caso el this se utiliza para llamar al
    segundo constructor */
    this("Monica");
}

// Métodos de acceso: get/set
public String getNombre() {
    return nombre;
}

public int getNota1() {
    return nota1;
}

public int getNota2() {
    return nota2;
}

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

public void setNota1(int nota1) {
    this.nota1 = nota1;
}

public void setNota2(int nota2) {
    this.nota2 = nota2;
}

// Operaciones
public double promedio() {
    return (nota1 + nota2) / 2.0;
}
}

```

- A la pulsación de un botón, cree tres objetos y visualice sus contenidos.

```

void procesar() {
    Alumno a1 = new Alumno("Juan",13,15),
    a2 = new Alumno("Pedro"),
    a3 = new Alumno();
    listado(a1);
    listado(a2);
    listado(a3);
    imprimir("# objetos creados : " + Alumno.cantidad);
}

```

```

➔ nombre : Juan
  nota 1 : 13
  nota 2 : 15
 promedio : 14.0

nombre : Pedro

```



```
nota 1 : 88
nota 2 : 88
promedio : 88.0

nombre : Monica
nota 1 : 88
nota 2 : 88
promedio : 88.0

# objetos creados : 3
```

4. MÉTODOS DE CLASE

Así como existen variables de clase, existen métodos de clase. Para declarar un método como método de clase, hay que anteponer a su tipo de retorno la palabra **static**.

Un método de clase puede operar únicamente sobre variables de clase y/o métodos de clase y no puede usar la referencia **this**.

Se puede acceder a un método de clase público a través de una referencia a un objeto de la clase o mediante el nombre de la clase.

Ejemplo 4

Declare la clase Clase que cuente la cantidad de objetos creados. Luego, a la pulsación de un botón, cree tres objetos.

```
package semana3;
public class Clase {
    private static int contador = 0;

    public Clase() {
        contador++;
    }

    public static int getContador() {
        return contador;
    }
}

void procesar() {
    Clase c1 = new Clase(),
    c2 = new Clase(),
    c3 = new Clase();
    imprimir("# objetos creados :" + Clase.getContador());
}
```

5. APLICACIÓN DE METODOS DE CLASE

Una aplicación interesante donde empleamos métodos de clase es cuando creamos una librería, es decir, una clase que contenga métodos a los cuales invocaremos desde cualquier otra clase de un proyecto sin necesidad de crear un objeto.

Ejemplo 5

Diseñe una clase que contenga métodos que puedan ser invocados desde cualquier clase sin necesidad de crear un objeto.

```
package compartido;
public class LibNumeros {

    private LibNumeros(){

    }

    public static double pot(double num, double exp){
        return Math.pow(num,exp);
    }

    public static double raizN(double num, double raiz){
        return Math.pow(num,(1/raiz));
    }

    public static double redondeoN(double num, int cant){
        return Math rint(num*pot(10,cant))/pot(10,cant);
    }

}
```

El constructor de la clase tiene un acceso privado para que no pueda ser invocado y, de esta forma, no podrá crear un objeto de tipo LibNumeros.

Descripción de los métodos de la clase LibNumeros:

- **Método pot:** el método retorna la potencia de un número (parámetro num) elevado a un exponente (parámetro exp).
- **Método raizN:** el método retorna la raíz (parámetro raíz) de un número (parámetro num).
- **Método redondeoN:** el método retorna un número (parámetro num) redondeado a una cantidad de decimales indicada (parámetro cant).

La llamada a los métodos sería de la siguiente manera:

```
void procesar(){
    imprimir("potencia de 5 elevado a 2:"+ LibNumeros.pot(5,2));
    imprimir("raiz cúbica de 27:"+ LibNumeros.raizN(27,3));
    imprimir("redondeo del número 13.44567 a 3 decimales:" +
        LibNumeros.redondeoN(13.44567,3));
}
```

Los resultados son:

```
potencia de 5 elevado a 2:25.0
raiz cúbica de 27:3.0
redondeo del número 13.44567 a 3 decimales:13.446
```

Ejemplo 6

Otro ejemplo de librería, pero esta vez trabajando con cadenas:

```
package compartido;
public class LibCadenas {

    private LibCadenas (){}
    }
    public static String mayus(String cad){
        return cad.toUpperCase();
    }

    public static String minus(String cad){
        return cad.toLowerCase();
    }

    public static String rellena(String cad, int num){
        int tam=cad.length();
        for(int i=tam; i<num; i++)
            cad+=' ';
        return cad;
    }

    public static String extrae(String cad, int pos){
        // pos en el rango de 1 .. longitud de la cadena.
        cad=cad.trim();
        int tam = cad.length();
        if(pos>=1 && pos<=tam)
            return ""+cad.charAt(pos-1);
        else
            return "Verifique los parámetros";
    }

    public static String extrae(String cad, int posIni,
                                int posFin){
        // pos en el rango de 1 .. longitud de la cadena.
        int tam=cad.length();
        if(posIni>=1 && posFin<=tam)
            return cad.substring(posIni-1,posFin);
        else
            return "Verifique los parámetros";
    }
}
```

Descripción de los métodos de la clase LibCadenas:

- **Método mayus:** El método retorna la cadena en mayúsculas.
- **Método minus:** El método retorna la cadena en minúsculas.

- **Método rellena:** El método rellena con espacios una cadena (parámetro cad) de tal forma que la cadena este compuesta por la cantidad de caracteres especificados (parámetro num).
- **Método extrae:** El método extrae es un método sobrecargado. Esto quiere decir que se puede invocar al método mandándole 2 parámetros (cadena y posición) o mandándole 3 parámetros (cadena, posición inicial y posición final). Si utilizamos la sintaxis: `LibCadenas.extrae(cadena,pos);` el método retornará una cadena que contenga el caracter que se encuentre en la posición pos.
Si utilizamos la sintaxis: `LibCadenas.extrae(cadena,posIni,posFin);` el método retornará la cadena comprendida desde la posición inicial hasta la posición final indicada.

La llamada a los métodos sería de la siguiente manera:

```
void procesar(){
    imprimir(LibCadenas.mayus("CibertEc"));
    imprimir(LibCadenas.minus("CIBERTEC"));
    imprimir("|"+LibCadenas.rellena("hola",10)+"|");
    imprimir(LibCadenas.extrae("Cibertec",5));
    imprimir(LibCadenas.extrae("hola como estas",6,9));
}
```

Los resultados son:

```
CIBERTEC
cibertec
|hola    |
r
como
```

6. INICIALIZADORES ESTÁTICOS

Un inicializador static es un bloque de código definido en la clase que se llama automáticamente al cargarse la clase en la memoria. Se diferencia del constructor en que no es invocado para cada objeto, sino una sola vez para toda la clase. La forma de crear un inicializador static es la siguiente:

```
static{
    . . .
    . . .
}
```

Este bloque de código se utiliza para darle valores iniciales a las variables estáticas. También se utilizan para invocar a métodos nativos, es decir, métodos escritos por ejemplo en lenguaje C. En una clase, se pueden definir varios inicializadores estáticos, los cuales se ejecutan en el orden en el que han sido definidos.

Ejemplo 7

Cree la clase **Empleado** en el **paquete semana3** y declare los siguiente **atributos** como **privados**: cod (int), nom (String) y horas (double). La clase debe tener las siguientes **variables estáticas** con acceso **privado**: tarifa, porcentaje de descuento (la tarifa, como el porcentaje de descuento, deben ser variables estáticas porque se va aplicar la misma tarifa y porcentaje de descuento a todos los empleados), contador de empleados y el sueldo neto acumulado de todos los empleados.

Implemente los siguientes métodos:

- Un constructor que inicialice a los atributos
- Métodos de acceso: set/get para cada atributo
- Un método que retorne el sueldo bruto (horas*tarifa)
- Un método que retorne el descuento (un porcentaje del sueldo bruto)
- Un método que retorne el sueldo neto (sueldo bruto - descuento)

La tarifa y el porcentaje de descuento se debe inicializar con S/. 40 y 11%, respectivamente. Para ello, utilice **inicializadores estáticos**. Debe existir la posibilidad de que el usuario pueda modificar la tarifa y el porcentaje de descuento.

En la **clase principal** (donde esta la GUI), implemente lo siguiente:

- Cree un objeto de tipo empleado cada vez que se pulse el **botón Procesar**. Los datos serán capturados de la GUI.
- Cree un método listar que imprima todos los atributos del empleado, la tarifa, el porcentaje de descuento, el sueldo bruto, el descuento en soles y el sueldo neto del empleado.
- Imprima la cantidad de empleados creados.
- Imprima el sueldo neto acumulado de todos los empleados.
- Cree un método modificar que permita ingresar la nueva tarifa y el nuevo porcentaje de descuento para los empleados. Este método debe ser llamado al pulsar el **botón Modificar**.

Solución del problema:

Cabe resaltar que, en esta solución, estamos aplicando casi todo lo que hemos visto en la primera unidad del curso.

Código de la clase Empleado:

```
package semana3;

import javax.swing.*.*;

public class Empleado {
    // Atributos privados
    private int cod;
    private String nom;
    private double horas;
```

```
// Variables de clase o estáticas
private static double tarifa;
private static double pordscto;
private static int cont;
private static double snetoAcum;

// Inicializadores static
static{
    tarifa=40;
    pordscto=11;
    mostrar("llamada al inicializador 1");
}

static{
    cont=0;
    snetoAcum=0;
    mostrar("llamada al inicializador 2");
}

// Constructor
public Empleado(int cod, String nom, double horas) {
    this.cod=cod;
    this.nom=nom;
    this.horas=horas;
    cont++;
    snetoAcum+=sneto();
    mostrar("llamada al constructor");
}

// Métodos de acceso: set
public void setCodigo(int cod){
    this.cod=cod;
}

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

public void setHoras(double horas){
    this.horas=horas;
}

// Métodos de acceso: get
public int getCodigo(){
    return cod;
}

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

```
public double getHoras(){
    return horas;
}

// Métodos de cálculo
public double sbruto(){
    return horas*tarifa;
}

public double descuento(){
    return sbruto()*(pordscto/100);
}

public double sneto(){
    return sbruto()-descuento();
}

// Métodos de clase o estáticos tipo void
public static void setTarifa(double t){
    tarifa=t;
}

public static void setPordscto(double p){
    pordscto=p;
}

public static void mostrar(String cad){
    JOptionPane.showMessageDialog(null,cad);
}

// Métodos de clase o estáticos con valor de retorno
public static double getTarifa(){
    return tarifa;
}

public static double getPordscto(){
    return pordscto;
}

public static int getContador(){
    return cont;
}

public static double getSnetoAcum(){
    return snetoAcum;
}

}
```

Código de la clase Principal:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import semana3.Empleado;

public class Principal extends JApplet implements
ActionListener{
    // Declaración de variables globales
    JLabel lblCodigo, lblNombre, lblHoras;
    JTextField txtCodigo, txtNombre, txtHoras;
    JButton btnProcesar, btnBorrar, btnModificar;
    JTextArea txtS;
    JScrollPane scpScroll;

    public void init() {
        getContentPane().setLayout(null);
        getContentPane().setBackground(new
            Color (214,211,206));

        // lblCodigo
        lblCodigo=new JLabel("Codigo:");
        lblCodigo.setBounds(10,10,100,20);
        getContentPane().add(lblCodigo);

        // lblNombre
        lblNombre=new JLabel("Nombre:");
        lblNombre.setBounds(10,30,100,20);
        getContentPane().add(lblNombre);

        // lblHoras
        lblHoras=new JLabel("Horas:");
        lblHoras.setBounds(10,50,100,20);
        getContentPane().add(lblHoras);

        // txtCodigo
        txtCodigo=new JTextField();
        txtCodigo.setBounds(120,10,100,20);
        getContentPane().add(txtCodigo);

        // txtNombre
        txtNombre=new JTextField();
        txtNombre.setBounds(120,30,100,20);
        getContentPane().add(txtNombre);

        // txtHoras
        txtHoras=new JTextField();
        txtHoras.setBounds(120,50,100,20);
        getContentPane().add(txtHoras);
```



```
// btnProcesar
btnProcesar = new JButton("Procesar");
btnProcesar.setBounds(250, 10, 100, 20);
btnProcesar.addActionListener(this);
getContentPane().add(btnProcesar);

// btnBorrar
btnBorrar = new JButton("Borrar");
btnBorrar.setBounds(250, 30, 100, 20);
btnBorrar.addActionListener(this);
getContentPane().add(btnBorrar);

// btnModificar
btnModificar = new JButton("Modificar");
btnModificar.setBounds(250, 50, 100, 20);
btnModificar.addActionListener(this);
getContentPane().add(btnModificar);

// txtS
txtS = new JTextArea();
txtS.setFont(new Font("monospaced", 0, 12));

// scpScroll
scpScroll=new JScrollPane(txtS);
scpScroll.setBounds(10, 90, 350, 240);
getContentPane().add(scpScroll);
}

// Procesa eventos de tipo ActionEvent
public void actionPerformed( ActionEvent e ){
    if( e.getSource() == btnProcesar )
        procesar();
    if( e.getSource() == btnModificar )
        modificar();
    if( e.getSource() == btnBorrar )
        borrar();
}

void procesar(){
    Empleado emp = new
    Empleado(getCodigo(),getNombre(),getHoras());
    listar(emp);
    imprimir("Cantidad de empleados creados:" +
Empleado.getContador());
    imprimir("Sueldo neto acumulado:" +
Empleado.getSnetoAcum());
    imprimir("");
}
```

```
void listar(Empleado x){
    imprimir("Objeto:"+x);
    imprimir("Código:"+x.getCodigo());
    imprimir("Nombre:"+x.getNombre());
    imprimir("Horas:"+x.getHoras());
    imprimir("Tarifa:"+x.getTarifa());
    imprimir("% de descuento:"+x.getPordscto());
    imprimir("Sueldo bruto:"+x.sbruto());
    imprimir("Descuento S/.:"+x.descuento());
    imprimir("Sueldo neto:"+x.sneto());
    imprimir("");
}

void modificar(){
    try{
        String valor1 = JOptionPane.showInputDialog(
            this,"Ingrese la nueva tarifa:",
            Empleado.getTarifa());
        double tarifa = Double.valueOf(valor1);
        String valor2 = JOptionPane.showInputDialog(
            this,"Ingrese el nuevo % descuento:",
            Empleado.getPordscto());
        double pordscto = Double.valueOf(valor2);
        //Modificando las variables estáticas
        Empleado.setTarifa(tarifa);
        Empleado.setPordscto(pordscto);
    }
    catch(Exception x){
        JOptionPane.showMessageDialog(this,"Verifique
            los datos ingresados");
    }
}

void borrar(){
    txtCodigo.setText("");
    txtNombre.setText("");
    txtHoras.setText("");
    txtS.setText("");
    txtCodigo.requestFocus();
}

void imprimir( String s ){
    txtS.append(s + "\n");
}

int getCodigo(){
    return Integer.parseInt(txtCodigo.getText());
}
```

```
String getNombre(){  
    return txtNombre.getText();  
}  
  
double getHoras(){  
    return Double.parseDouble(txtHoras.getText());  
}  
}
```

Importante:

Cabe resaltar que, una vez que la clase se carga en la memoria (esto ocurre al crearse el primer objeto o al utilizar un variable o método **static**), primero se ejecutan los inicializadores static y, luego, el constructor de la clase.

Ejercicios

Cree la clase **Vendedor** en el **paquete semana3** y declare los siguiente **atributos** como **privados**: cod (int), nom (String) y monto vendido (double). La clase debe tener las siguientes **variables estáticas** con acceso **privado**: porcentaje de comisión (debe ser variable estática, porque es el mismo para todos los vendedores), contador de vendedores y el sueldo bruto acumulado de todos los vendedores.

Implemente los siguientes métodos:

- a) Un constructor que inicialice a los atributos
- b) Métodos de acceso: set/get para cada atributo
- c) Un método que retorne el sueldo basico (sueldo fijo de S/ 550.00)
- d) Un método que retorne la comisión en soles (monto vendido * % de comisión)
- e) Un método que retorne el sueldo bruto (sueldo basico + comisión)

El porcentaje de comisión se debe inicializar con 5% . Para ello, utilice **inicializadores estáticos**. Debe existir la posibilidad de que el usuario pueda modificar el porcentaje de comisión.

En la **clase principal** (donde esta la GUI) implemente lo siguiente:

- a) Cree un objeto de tipo vendedor cada vez que se pulse el **botón Procesar**. Los datos serán capturados de la GUI.
- b) Cree un método listar que imprima todos los atributos del vendedor, el porcentaje de comisión, el sueldo basico, la comisión en soles y el sueldo bruto del vendedor.
- c) Imprima la cantidad de vendedores creados.
- d) Imprima el sueldo bruto acumulado de todos los vendedores.
- e) Cree un método modificar que permita ingresar el nuevo porcentaje de comisión para los vendedores. Este método debe ser llamado al pulsar el **botón Modificar**.

**UNIDAD DE
APRENDIZAJE****2****SEMANA****4**

MANEJO DE ARREGLOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos manipulan arreglos unidimensionales y bidimensionales con tipos primitivos, así como arreglo de objetos en diversas aplicaciones.

TEMARIO

- Arreglos unidimensionales
- Operaciones variadas

ACTIVIDADES PROPUESTAS

- Los alumnos reconocen un arreglo unidimensional.
- Los alumnos emplean arreglos unidimensionales en diversas aplicaciones.

1. ARREGLO UNIDIMENSIONAL

Un arreglo es un conjunto de elementos dispuestos uno a continuación de otro, donde cada elemento conserva su propio espacio (tamaño en bytes).

El espacio ocupado por cada elemento es igual para todos y, en él, se pueden almacenar diferentes valores, pero del mismo tipo de dato.

El tipo más simple de un arreglo es el unidimensional.

a) Declaración:

Forma 1:

```
tipo_dato nombre_arreglo[];
```

ejemplos:

```
int n[];
double sueldos[];
String apellidos[];
```

Forma 2:

```
tipo_dato[] nombre_arreglo;
```

ejemplos:

```
int[] n;
double[] sueldos;
String[] apellidos;
```

Con la declaración sólo se crea la variable que hace referencia al arreglo y su contenido es null.

b) Creación:

```
nombre_arreglo = new tipo[tamaño];
```

ejemplos:

```
n = new int [10];
sueldos = new double[8];
apellidos = new String[12];
```

Con la creación le indicamos a Java la cantidad de elementos que va a tener el arreglo (tamaño del bloque de elementos) y la variable que hace referencia al arreglo almacena la dirección del primer elemento del arreglo.

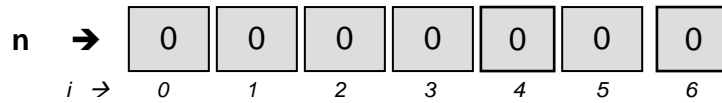
c) Declaración y creación en la misma línea:

```
tipo_dato nombre_arreglo[] = new tipo_dato[tamaño];
```

ejemplo:

```
int n[] = new int[7];
```

gráficamente:



Consideraciones:

- ✓ Java, por defecto, inicializa un arreglo de tipo `int` con 0, un arreglo de tipo `double` con 0.0, un arreglo de tipo `String` con `null` y un arreglo de tipo `boolean` con `false`.
- ✓ Java cuenta las posiciones a partir de 0
- ✓ Esta posición se denomina índice: `i`
- ✓ Los siete elementos del arreglo `n` son: `n[0]`, `n[1]`, `n[2]`, `n[3]`, `n[4]`, `n[5]`, `n[6]`
- ✓ `n[7]` no está definido
- ✓ `n.length` devuelve el tamaño del arreglo (en este caso, 7)
- ✓ El índice `i` se encuentra en el rango: $0 \leq i < n.length$

d) Inicialización: Se puede inicializar un arreglo con números aleatorios de la siguiente forma:

```
void generar() {
    for(int i=0; i< n.length ; i++)
        n[i]=aleatorio(10,99);
}

int aleatorio (int min, int max) {
    return (int)((max - min + 1) * Math.random()+ min);
}
```

e) Declaración, creación e inicialización: Se puede declarar, crear e inicializar un arreglo en una misma línea de la siguiente forma:

```
tipo_de_dato nombre_del_arreglo[]={valor1, valor2, ...};
```

Ejemplos:

```
int n[]={1,12,0,-5,25};
String nombres[]= {"Juan","Pedro","Ana"};
```

2. OPERACIONES VARIADAS

Ejemplo 1

Este ejemplo esta compuesto de 2 clases: la **clase ArregloUnidimensional** (clase que maneja el arreglo) y la **clase Principal** (clase que interactúa con el usuario a través de la GUI).

Código de la clase ArregloUnidimensional

```
package semana4;

public class ArregloUnidimensional {

    //Atributos privados
    private int edades[];
    private int indice;

    //Constructor
    public ArregloUnidimensional() {
        edades = new int[10];
        indice=0;
    }

    //Métodos públicos
    public void adicionar(int edad) {
        edades[indice]=edad;
        indice++;
    }

    public int getIndice(){
        return indice;
    }

    public int longTotal(){
        return edades.length;
    }

    public int obtener(int pos){
        return edades[pos];
    }

    public void reinicializar(){
        edades = new int[10];
        indice=0;
    }

    public void generar(){
        for(int i=0; i<longTotal(); i++)
            edades[i]=aleatorio(10,99);
        indice=longTotal();
    }
}
```



```

    public int buscar(int edad){
        for(int i=0; i<indice; i++)
            if(edad==edades[i])
                return i;

        return -1;
    }

    public int sumaEdades(){
        int suma=0;
        for(int i=0; i<indice; i++)
            suma+=edades[i];
        return suma;
    }

    public int edadMenor(){
        int min=edades[0];
        for(int i=1; i<indice; i++)
            if(edades[i]<min)
                min=edades[i];
        return min;
    }

    public int edadMayor(){
        int max=edades[0];
        for(int i=1; i<indice; i++)
            if(edades[i]>max)
                max=edades[i];
        return max;
    }

    public double edadPromedio(){
        return sumaEdades()*1.0/indice;
    }

    //Método privado
    private int aleatorio (int min, int max) {
        return (int)((max - min + 1) * Math.random()+ min);
    }
}

```

Podemos apreciar que la clase ArregloUnidimensional tiene **2 atributos privados**:

- Un **arreglo de edades** de tipo int
- La variable **indice** de tipo int, que almacena la cantidad de edades ingresadas y la posición del arreglo donde se almacenará una nueva edad.

Descripción de métodos

- El método **constructor** se encarga de crear el arreglo con 10 elementos e inicializa la variable indice en cero ya que el arreglo esta vacío.

- El método **adicionar** se encarga de almacenar la edad en el arreglo e incrementa la variable índice en uno.
- El método **getIndice** es un método de acceso para el atributo índice, ya que éste es privado. Devuelve la cantidad de edades ingresadas.
- El método **longTotal** devuelve la longitud total de todo el arreglo
- El método **obtener** devuelve la edad que se encuentra en el arreglo en la posición pos, ya que desde afuera de la clase no se tiene acceso al arreglo por ser privado.
- El método **reinicializar** crea un nuevo bloque de 10 elementos de tipo int y se almacena en la variable edades la referencia o dirección de éste nuevo bloque de elementos perdiéndose la dirección del bloque anterior. El garbage collector se encargará de destruir el bloque anterior.
- El método **generar** llena el arreglo de números aleatorios de 2 cifras.
- El método **buscar** localiza una edad y devuelve la posición donde lo encontró. Si la edad no existe, devuelve -1, que es un indicador de no encontrado.
- Los métodos sumaEdades, edadMenor, edadMayor y edadPromedio son métodos de cálculo.

Código de la clase Principal

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import semana4.ArregloUnidimensional;

public class Principal extends JApplet implements
ActionListener {

    // Declaración de variable globales
    JLabel      lblEdad, lblArreglo;
    JTextField  txtEdad;
    JButton      btnIngresar, btnListar, btnReportar,
                btnReinicializar, btnGenerar,
                btnBuscar;
    JTextArea   txtS;
    JScrollPane scpScroll;

    ArregloUnidimensional a=new ArregloUnidimensional();

    // Creación de la interfaz gráfica de usuario: GUI

    public void init() {
        getContentPane().setLayout(null);
        getContentPane().setSize(460,400);
        getContentPane().setBackground(Color.lightGray);

        lblEdad = new JLabel("Edad :");
        lblEdad.setBounds(40,20,100,25);
        getContentPane().add(lblEdad);

        txtEdad = new JTextField();
```

```
txtEdad.setBounds(120,20,80,25);
getContentPane().add(txtEdad);

lblArreglo = new JLabel("Arreglo
Lineal",JLabel.CENTER);
lblArreglo.setBackground(new
Color(226,223,234));
lblArreglo.setOpaque(true);
lblArreglo.setForeground(Color.black);
lblArreglo.setFont(new Font("dialog",3,19));
lblArreglo.setBounds(250,20,199,40);
getContentPane().add(lblArreglo);

btnIngresar = new JButton("Ingresar");
btnIngresar.setBounds(320,80,150,23);
btnIngresar.addActionListener(this);
getContentPane().add(btnIngresar);

btnListar = new JButton("listar");
btnListar.setBounds(320,105,150,23);
btnListar.addActionListener(this);
getContentPane().add(btnListar);

btnReportar = new JButton("reportar");
btnReportar.setBounds(320,130,150,23);
btnReportar.addActionListener(this);
getContentPane().add(btnReportar);

btnReinicializar = new JButton("reinicializar
arreglo");
btnReinicializar.setBounds(320,155,150,23);
btnReinicializar.addActionListener(this);
getContentPane().add(btnReinicializar);

btnGenerar = new JButton("generar");
btnGenerar.setBounds(320,180,150,23);
btnGenerar.addActionListener(this);
getContentPane().add(btnGenerar);

btnBuscar = new JButton("Buscar");
btnBuscar.setBounds(320,205,150,23);
btnBuscar.addActionListener(this);
getContentPane().add(btnBuscar);

txtS = new JTextArea();

scpScroll=new JScrollPane(txtS);
scpScroll.setBounds(10,80,300,298);
getContentPane().add(scpScroll);

}
```

```

// Ejecución de los eventos tipo ActionEvent
public void actionPerformed (ActionEvent e) {
    if (e.getSource() == btnIngresar)
        ingresar();
    if (e.getSource() == btnListar)
        listar();
    if (e.getSource() == btnReportar)
        reportar();
    if (e.getSource() == btnReinicializar)
        reinicializarArreglo();
    if (e.getSource() == btnGenerar)
        generar();
    if (e.getSource() == btnBuscar)
        buscar();
    limpiar();
}

void ingresar() {
    try{
        if(a.getIndice()<a.longTotal()){
            int edad=getEdad();
            a.adicionar(edad);
            listar();
        }
        else
            mensaje("Arreglo lleno");
    }
    catch(Exception x){
        mensaje("Edad incorrecta");
    }
}

void listar() {
    txtS.setText("");
    if(a.getIndice()>0){
        for(int i=0; i<a.getIndice(); i++){
            imprimir("edad["+i+"] =" +
                a.obtener(i));
        }
    }
    else
        mensaje("Arreglo vacío...");
}

void reportar() {
    txtS.setText("");
    if(a.getIndice()>0){
        imprimir("Suma de edades:"+a.sumaEdades());
        imprimir("Edad menor:"+a.edadMenor());
        imprimir("Edad mayor:"+a.edadMayor());
        imprimir("Promedio de edades:" +
            a.edadPromedio());
    }
}

```

```
        else
            mensaje("Arreglo vacío...");
    }

    void reinicializarArreglo() {
        if(a.getIndice()>0){
            txtS.setText("");
            a.reinicializar();
            mensaje("El arreglo se reinicializó");
        }
        else
            mensaje("Arreglo vacío...");
    }

    void generar() {
        a.generar();
        listar();
    }

    void buscar(){
        try{
            int edad=getEdad();
            int pos=a.buscar(edad);
            if(pos== -1)
                mensaje("La edad no existe");
            else
                mensaje("La edad se encuentra en la
                    posición: "+pos);
        }
        catch(Exception x){
            mensaje("Edad incorrecta");
        }
    }

    void limpiar() {
        txtEdad.setText("");
        txtEdad.requestFocus();
    }

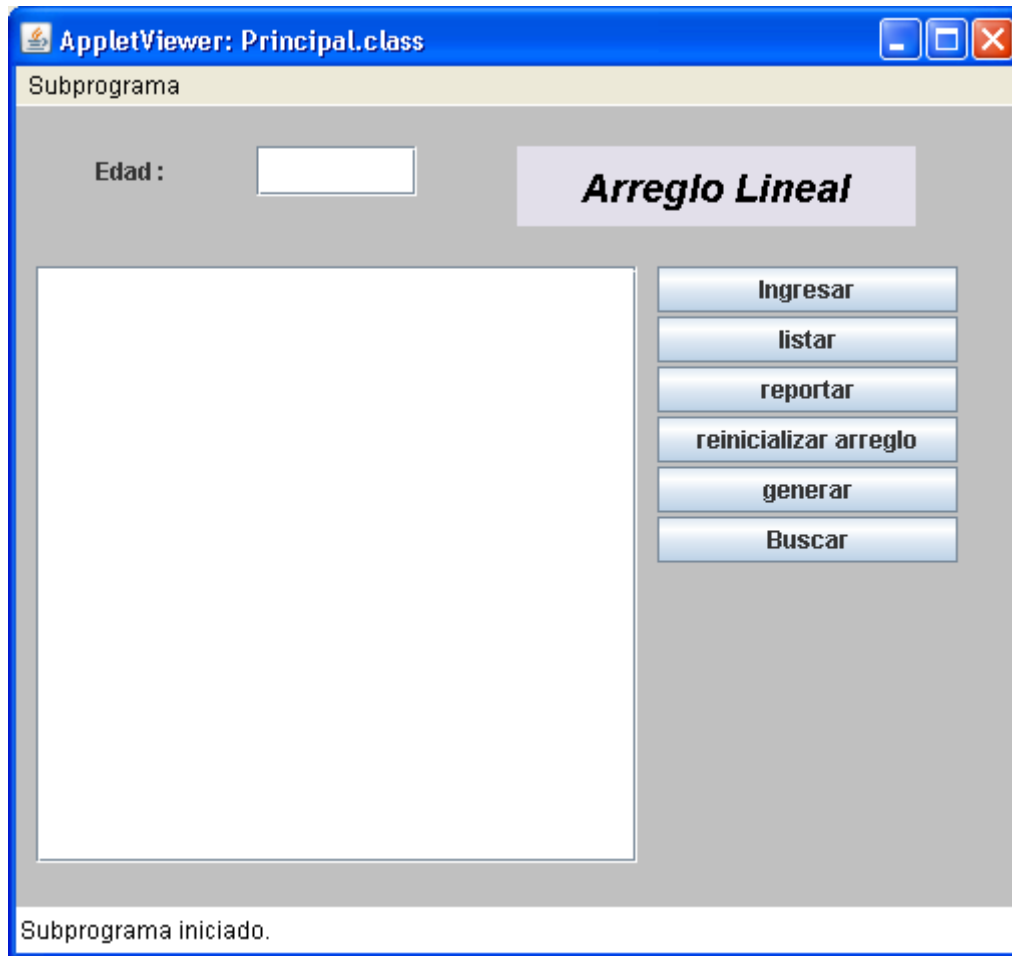
    void imprimir (String cadena) {
        txtS.append(cadena + "\n");
    }

    int getEdad() {
        return Integer.parseInt(txtEdad.getText());
    }

    void mensaje(String cad){
        JOptionPane.showMessageDialog(this,cad);
    }

}
```

GUI del Ejemplo 1:



Ejemplo 2

Cree un método que permita adicionar edades en forma ilimitada, es decir, que si el arreglo edades se llena, éste se amplie.

En la clase **ArregloUnidimensional**:

```
public void adicionar(int edad) {
    if(indice==edades.length)
        ampliarArreglo();

    edades[indice]=edad;
    indice++;
}

private void ampliarArreglo(){
    int aux[]=edades;
    edades=new int[indice+10];
    for(int i=0; i<indice; i++)
        edades[i]=aux[i];
}
```

En la clase **Principal**:

```
void ingresar() {
    try{
        int edad=getEdad();
        a.adicionar(edad);
        listar();
    }
    catch(Exception x){
        mensaje("Edad incorrecta");
    }
}
```

Ejemplo 3

Cree un método que permita ordenar en forma ascendente el arreglo de edades.

En la clase **ArregloUnidimensional**:

```
public void ordenarAscendente(){
    int aux;
    for(int i=0; i<indice-1; i++)
        for(int j=i+1; j<indice; j++)
            if(edades[i]>edades[j]){
                //intercambio
                aux=edades[i];
                edades[i]=edades[j];
                edades[j]=aux;
            }
}
```

En la clase **Principal**:

```
void ordenar(){
    a.ordenarAscendente();
    listar();
}
```

Ejercicios

- 1) Cree la clase **ArregloSueldos** en el paquete semana4 y declare los siguientes atributos como privados:

- Un arreglo de sueldos de tipo double
- Una variable índice de tipo int que almacene la cantidad de sueldos

Implemente los siguientes métodos:

- Un constructor que cree el arreglo
- Un método que genere 10 sueldos en el rango: 500 .. 3000
- Un método que calcule el sueldo promedio.
- Un método que retorne la cantidad de sueldos mayores a 2000

En la clase **Principal**, debe programar los botones generar, sueldo promedio y cantidad de sueldos de tal forma que, al pulsar estos botones, se llamen a los métodos creados en la clase ArregloSueldos.

- 2) Cree la clase **ArregloNumeros** en el paquete semana4 y declare los siguientes atributos como privados:

- Un arreglo de números de tipo int
- Una variable índice de tipo int que almacene la cantidad de números

Implemente los siguientes métodos:

- Un constructor que cree el arreglo
- Un método que genere 15 numeros en el rango: 10 .. 99
- Un método que calcule el número mayor
- Un método que retorne la cantidad de números impares
- Un método que retorne la cantidad de números capicuas

En la clase **Principal**, debe programar los botones:

- Generar, que al pulsarlo, se genere los 15 números en el arreglo
- Reportar, que al pulsarlo, imprime el número mayor, la cantidad de números impares y la cantidad de números capicuas.

**UNIDAD DE
APRENDIZAJE****2****SEMANA****5**

ARREGLO BIDIMENSIONAL

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos manipulan arreglos unidimensionales y bidimensionales con tipos primitivos, así como arreglo de objetos en diversas aplicaciones.

TEMARIO

- Arreglo Bidimensional
- Operaciones variadas

ACTIVIDADES PROPUESTAS

- Los alumnos reconocen un arreglo bidimensional.
- Los alumnos emplean arreglos bidimensionales en diversas aplicaciones.

1. ARREGLO BIDIMENSIONAL

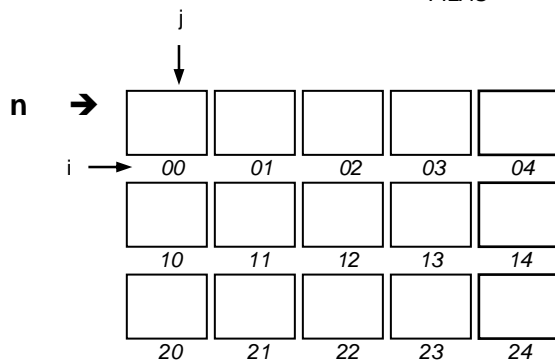
Un arreglo bidimensional es una matriz donde cada elemento es referenciado a través de una fila y una columna. El espacio ocupado por cada elemento es igual para todos y, en él, se puede almacenar diferentes valores, pero del mismo tipo de dato.

Arreglo bidimensional (dos direcciones)

Ej:

```
int n[][] = new int[3][5];
```

↑ *FILAS*
↑ *COLUMNAS*



- Se ha declarado un arreglo bidimensional **n** del tipo de dato entero.
- El índice **i** lo usaremos para acceder a una fila y el índice **j** para acceder a una columna.
- Los 15 elementos del arreglo **n** son:

```
n[0][0], n[0][1], n[0][2], n[0][3], n[0][4],
n[1][0], n[1][1], n[1][2], n[1][3], n[1][4],
n[2][0], n[2][1], n[2][2], n[2][3], n[2][4]
```

- **n[3][5]** no está definido.
- En forma genérica: **n[i][j]**

```
0 <= i < FILAS
0 <= j < COLUMNAS
```

- **n.length** devuelve la cantidad de filas, en este caso, 3
- **n[i].length** devuelve la cantidad de columnas de la fila **i**, en este caso, 5

NÚMERO DE ELEMENTO: $e = (i * \text{columnas} + j) + 1$

Así:

$i \quad j \qquad \qquad i \quad j \qquad e$

- $n[0][0]$ se refiere al $(0 * 5 + 0) + 1 = 1$ er. elemento

- $n[1][0]$ se refiere al $(1 * 5 + 0) + 1 = 6$ to. elemento

- $n[2][0]$ se refiere al $(2 * 5 + 0) + 1 = 11$ vo. elemento

- $n[2][4]$ se refiere al $(2 * 5 + 4) + 1 = 15$ vo. y último elemento

INGRESO:

Elabore un método que genere números de dos cifras.

```
void generarNumeros() {
    for (int i=0; i<FILAS; i++)
        for (int j=0; j<COLUMNAS; j++)
            n[i][j] = aleatorio(10,99);
}
```

j
↓

n →

	60	57	14	78	75
i →	00	01	02	03	04
	80	65	61	41	84
	10	11	12	13	14
	14	72	73	66	57
	20	21	22	23	24

LISTADO:

Visualice los números generados.

```
void listar() {
    for (int i=0; i<FILAS; i++) {
        for (int j=0; j<COLUMNAS; j++)
            txtS.append("n[" + i + "][" + j + "] : " +
                        n[i][j] + "\t");
        txtS.append("\n");
    }
}
```

→ $n[0][0]:60$ $n[0][1]:57$ $n[0][2]:14$ $n[0][3]:78$ $n[0][4]:75$
 $n[1][0]:80$ $n[1][1]:65$ $n[1][2]:61$ $n[1][3]:41$ $n[1][4]:84$
 $n[2][0]:14$ $n[2][1]:72$ $n[2][2]:73$ $n[2][3]:66$ $n[2][4]:57$

2. OPERACIONES VARIADAS

Ejemplo 1

Diseñe un programa que genere, aleatoriamente, un arreglo bidimensional de números en función de la cantidad de filas y columnas que el usuario ingrese. Los números deben estar en el rango de 10 a 99. Luego, al pulsar el botón Procesar, que calcule lo siguiente:

- a) La cantidad de números
- b) La suma de los números
- c) El número mayor
- d) El número menor
- e) El promedio de los números
- f) La suma de los números de la fila 2
- g) La suma de los números de la columna 3

El arreglo Bidimensional se debe crear en la **clase ArregloBidimensional** dentro del **paquete semana5** y la GUI se debe controlar desde la **clase Principal**.

Código de la clase ArregloBidimensional

```
package semana5;
public class ArregloBidimensional {

    //Atributos privados
    private int fil, col;
    private int n[][];

    //Constructor
    public ArregloBidimensional(int fil, int col){
        n = new int[fil][col];
        this.fil=fil;
        this.col=col;
        generar();
    }

    //Métodos públicos

    public int getFilas(){
        return fil;
    }

    public int getColumnas(){
        return col;
    }

    public int getLongitud(){
        return fil*col;
    }

    public int obtener(int posFil, int posCol){
        return n[posFil][posCol];
    }
}
```

```
public int suma(){
    int sum=0;
    for(int i=0; i<fil; i++)
        for(int j=0; j<col; j++)
            sum+=n[i][j];
    return sum;
}

public int mayor(){
    int max=n[0][0];
    for(int i=0; i<fil; i++)
        for(int j=0; j<col; j++)
            if(n[i][j]>max)
                max=n[i][j];
    return max;
}

public int menor(){
    int min=n[0][0];
    for(int i=0; i<fil; i++)
        for(int j=0; j<col; j++)
            if(n[i][j]<min)
                min=n[i][j];
    return min;
}

public double promedio(){
    return suma()*1.0/getLongitud();
}

public int sumaFila(int f){
    int sum=0;
    for(int i=0; i<fil; i++)
        if(i==f)
            for(int j=0; j<col; j++)
                sum+=n[i][j];
    return sum;
}

public int sumaColumna(int c){
    int sum=0;
    for(int j=0; j<col; j++)
        if(j==c)
            for(int i=0; i<fil; i++)
                sum+=n[i][j];
    return sum;
}

//Métodos privados

private void generar(){
    for(int i=0; i<fil; i++)
        for(int j=0; j<col; j++)
            n[i][j]=aleatorio(10,99);
}
```

```
    }

    private int aleatorio(int min, int max){
        return (int)((max-min+1)*Math.random()+min);
    }
}
```

Código de la clase Principal

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import semana5.ArregloBidimensional;

public class Principal extends JApplet implements
ActionListener{

    JLabel lblFilas, lblColumnas;
    JTextField txtFilas, txtColumnas;
    JButton btnGenerar, btnProcesar;
    JTextArea txtS;
    JScrollPane scpScroll;

    ArregloBidimensional a;

    // Crea la interfaz gráfica de usuario GUI
    public void init() {
        getContentPane().setLayout(null);
        getContentPane().setBackground(Color.lightGray);

        lblFilas = new JLabel("Filas");
        lblFilas.setBounds(30,20,100,20);
        getContentPane().add(lblFilas);

        txtFilas = new JTextField();
        txtFilas.setBounds(150,20,60,20);
        getContentPane().add(txtFilas);

        lblColumnas = new JLabel("Columnas");
        lblColumnas.setBounds(30,50,100,20);
        getContentPane().add(lblColumnas);

        txtColumnas = new JTextField();
        txtColumnas.setBounds(150,50,60,20);
        getContentPane().add(txtColumnas);

        btnGenerar=new JButton("Generar");
        btnGenerar.setBounds(300,20,100,20);
        btnGenerar.addActionListener(this);
        getContentPane().add(btnGenerar);

        btnProcesar=new JButton("Procesar");
        btnProcesar.setBounds(300,50,100,20);
        btnProcesar.addActionListener(this);
        getContentPane().add(btnProcesar);
    }
}
```

```
txtS=new JTextArea();

scpScroll=new JScrollPane(txtS);
scpScroll.setBounds(30,100,400,220);
getContentPane().add(scpScroll);

}

// Procesa eventos de tipo(ActionEvent
public void actionPerformed( ActionEvent e ){
    if(e.getSource()==btnGenerar)
        generar();
    if(e.getSource()==btnProcesar)
        procesar();
}

void generar(){
    int filas=getFilas();
    int col=getColumnas();

    if(filas>0 && col>0){
        a = new ArregloBidimensional(filas,col);
        listar();
    }
    else
        mensaje("datos incorrectos");
}

void listar(){
    txtS.setText("");
    for(int i=0; i<a.getFilas(); i++){
        for(int j=0; j<a.getColumnas(); j++){
            int x = a.obtener(i,j);
            txtS.append("n["+i+"]["+j+"]="+x+"\t");
        }
        txtS.append("\n");
    }
}

void procesar(){
    imprimir("");
    imprimir("a) Cantidad de números:"+a.getLongitud());
    imprimir("b) Suma de números:"+a.suma());
    imprimir("c) Número mayor:"+a.mayor());
    imprimir("d) Número menor:"+a.menor());
    imprimir("e) Promedio de números:"+a.promedio());
    imprimir("f) Suma de números fila 2:"+a.sumaFila(2));
    imprimir("g) Suma de números columna 3:"+
        a.sumaColumna(3));
}

int getFilas() {
    int num;
    try {
        num = Integer.parseInt(txtFilas.getText());
    }
```

```

    }
    catch(Exception e) {
        num = 0;
    }
    return num;
}

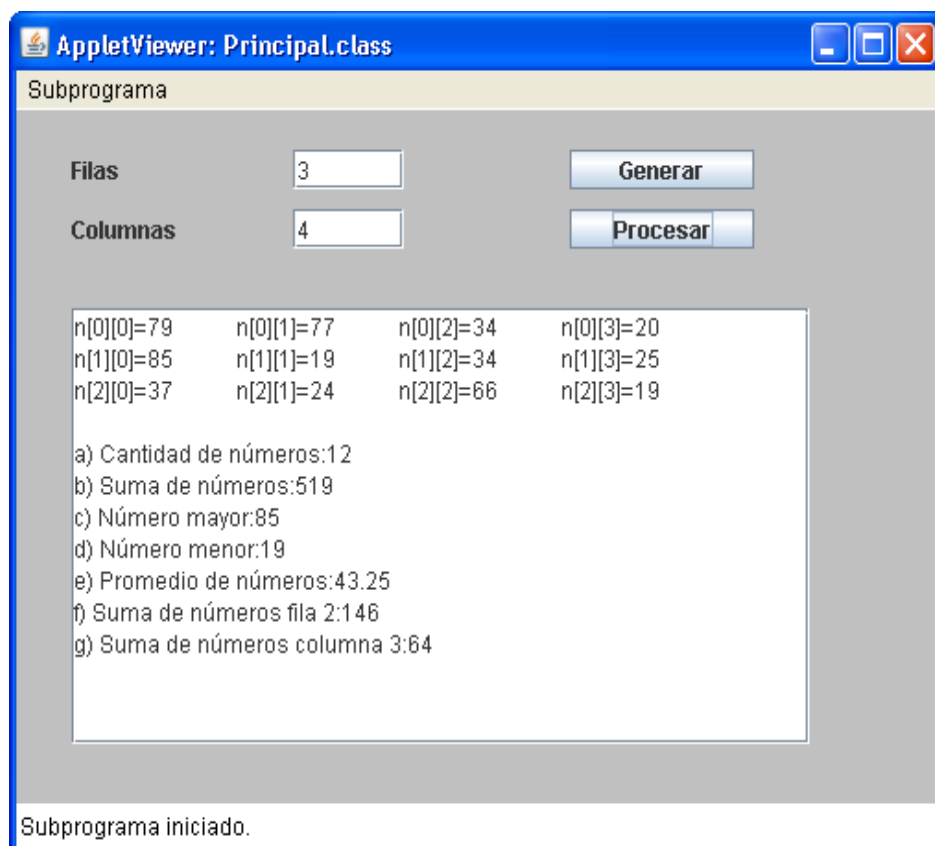
int getColumnas() {
    int num;
    try {
        num = Integer.parseInt(txtColumnas.getText());
    }
    catch(Exception e) {
        num = 0;
    }
    return num;
}

void imprimir(String cad){
    txtS.append(cad + "\n");
}

void mensaje(String cad){
    JOptionPane.showMessageDialog(this,cad);
}
}

```

GUI :



Ejemplo 2

Diseñe un programa para encuestar a un grupo de personas de diferentes edades acerca de sus preferencias en bebidas gaseosas entre Pepsi Cola, Coca Cola, Fanta y Crush. La empresa encuestadora desea saber lo siguiente:

- 1) Cuántas personas en total, prefieren cada tipo de gaseosa.
- 2) Cuántas personas fueron encuestadas en cada rango de edades

El arreglo Bidimensional se debe crear en la **clase ArregloEncuestas** dentro del **paquete semana5** y la GUI se debe controlar desde la **clase Principal**.

Código de la clase ArregloEncuestas

```
package semana5;

public class ArregloEncuestas {
    //Atributos privados
    private final int filas =5,col= 4;
    private int encuestas[][];

    //Constructor
    public ArregloEncuestas() {
        encuestas=new int[filas][col];
    }

    //Métodos públicos
    public int getFilas(){
        return filas;
    }

    public int getColumnas(){
        return col;
    }

    public void registrar(int per, int gas){
        encuestas[per][gas]++;
    }

    public int obtener(int posFila, int posCol){
        return encuestas[posFila][posCol];
    }
}
```

Código de la clase Principal

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import semana5.ArregloEncuestas;
```

```
public class Principal extends JApplet implements
ActionListener{

    JLabel lblPersonas,lblGaseosas;
    JComboBox cboPersonas,cboGaseosas;
    JButton btnEncuestar,btnCalcular;
    JTextArea txtS;
    JScrollPane scpScroll;

    ArregloEncuestas a = new ArregloEncuestas();

    // Crea la interfaz gráfica de usuario GUI
    public void init() {
        getContentPane().setLayout(null);
        getContentPane().setBackground(Color.lightGray);

        lblPersonas = new JLabel("Personas");
        lblPersonas.setBounds(30,20,100,20);
        getContentPane().add(lblPersonas);

        cboPersonas = new JComboBox();
        cboPersonas.setBounds(150, 20, 110, 20);
        cboPersonas.addItem("12 - 15 años");
        cboPersonas.addItem("16 - 20 años");
        cboPersonas.addItem("21 - 30 años");
        cboPersonas.addItem("31 - 50 años");
        cboPersonas.addItem("más 50 años");
        getContentPane().add(cboPersonas);

        lblGaseosas = new JLabel("Gaseosas");
        lblGaseosas.setBounds(30,50,100,20);
        getContentPane().add(lblGaseosas);

        cboGaseosas = new JComboBox();
        cboGaseosas.setBounds(150, 50, 110, 20);
        cboGaseosas.addItem("Pepsi Cola");
        cboGaseosas.addItem("Coca Cola");
        cboGaseosas.addItem("Fanta");
        cboGaseosas.addItem("Crush");
        getContentPane().add(cboGaseosas);

        btnEncuestar=new JButton("Encuestar");
        btnEncuestar.setBounds(350,20,100,20);
        btnEncuestar.addActionListener(this);
        getContentPane().add(btnEncuestar);

        btnCalcular=new JButton("Calcular");
        btnCalcular.setBounds(350,50,100,20);
        btnCalcular.addActionListener(this);
        getContentPane().add(btnCalcular);

        txtS=new JTextArea();

        scpScroll=new JScrollPane(txtS);
        scpScroll.setBounds(30,100,500,340);
        getContentPane().add(scpScroll);
```

```
        listar();
    }

    // Procesa eventos de tipo(ActionEvent
    public void actionPerformed( ActionEvent e ){
        if(e.getSource()==btnEncuestar)
            encuestar();

        if(e.getSource()==btnCalcular)
            calcular();
    }

    void encuestar(){
        int per,gas;

        per=cboPersonas.getSelectedIndex();
        gas=cboGaseosas.getSelectedIndex();

        a.registrar(per,gas);
        listar();
    }

    void calcular(){
        pregunta1();
        pregunta2();
    }

    void pregunta1(){
        int tot;
        for(int j=0; j<a.getColumnas(); j++){
            tot=0;
            for(int i=0; i<a.getFilas(); i++){
                tot+=a.obtener(i,j);
                imprimir("Total gaseosa "+
                    (String)cboGaseosas.getItemAt(j)+":"+tot);
            }
        }
    }

    void pregunta2(){
        int tot;
        for(int i=0; i<a.getFilas(); i++){
            tot=0;
            for(int j=0; j<a.getColumnas(); j++){
                tot+=a.obtener(i,j);
                imprimir("Total personas "+
                    (String)cboPersonas.getItemAt(i)+":"+tot);
            }
        }
    }

    void imprimir(String cad){
        txtS.append(cad + "\n");
    }

    void listar(){
        txtS.setText("");
    }
}
```

```

        imprimir("\t\t Preferencia de Gaseosas \n");
        imprimir("Rango de Edades \t Pepsi Cola \t Coca Cola
\t Fanta \t Crush");
        for(int i=0; i<a.getFilas(); i++){
            txtS.append((String)cboPersonas.getItemAt(i)+"\t\t");
            for(int j=0; j<a.getColumnas(); j++){
                txtS.append(a.obtener(i,j)+"\t");
            }
            imprimir("");
        }
    }
}

```

GUI :

Subprograma

Personas: 12 - 15 años

Gaseosas: Pepsi Cola

Encuestar

Calcular

Preferencia de Gaseosas				
Rango de Edades	Pepsi Cola	Coca Cola	Fanta	Crush
12 - 15 años	2	1	3	1
16 - 20 años	1	0	0	0
21 - 30 años	1	3	0	0
31 - 50 años	2	4	0	0
más 50 años	1	2	1	0

Total gaseosa Pepsi Cola:7
 Total gaseosa Coca Cola:10
 Total gaseosa Fanta:4
 Total gaseosa Crush:1
 Total personas 12 - 15 años:7
 Total personas 16 - 20 años:1
 Total personas 21 - 30 años:4
 Total personas 31 - 50 años:6
 Total personas más 50 años:4

Subprograma iniciado.

Ejercicios

- 1) Diseñe un programa que genere un arreglo bidimensional de edades en función de la cantidad de filas y columnas que el usuario ingrese. Las edades deben estar en el rango de 18 a 90 años. Luego, calcule lo siguiente:
 - a) La cantidad de edades generadas
 - b) La suma de todas las edades
 - c) La edad menor
 - d) La edad mayor
 - e) El promedio de edades
 - f) La cantidad de personas que superan los 40 años

El arreglo Bidimensional se debe crear en la clase ArregloEdades dentro del paquete semana5 y la GUI se debe controlar desde la clase Principal.

- 2) Diseñe un programa que genere una matriz cuadrada ($n \times n$) de números enteros e implemente los siguientes métodos:
 - a) Un método que imprima los elementos y la suma que conforma su diagonal izquierda (que inicia en la esquina superior izquierda y termina en la esquina inferior derecha).
 - b) Un método que imprima los elementos y la suma que conforma su diagonal derecha (que inicia en la esquina superior derecha y termina en la esquina inferior izquierda).

El arreglo Bidimensional se debe crear en la clase MatrizCuadrada dentro del paquete semana5 y la GUI se debe controlar desde la clase Principal.

**UNIDAD DE
APRENDIZAJE****2****SEMANA****6**

ARREGLO DE OBJETOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos manipulan arreglos unidimensionales y bidimensionales con tipos primitivos, así como arreglo de objetos en diversas aplicaciones.

TEMARIO

- Arreglo de objetos
- Operaciones variadas

ACTIVIDADES PROPUESTAS

- Los alumnos reconocen un arreglo de objetos.
- Los alumnos emplean arreglo de objetos en diversas aplicaciones.

1. ARREGLO DE OBJETOS

Un arreglo de objetos es un conjunto de variables de referencia que controlan objetos del mismo tipo.

a) Creación de un arreglo de objetos:

```
Libro alib[] = new Libro[3];
```

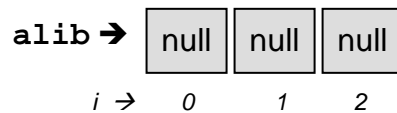
Donde:

Libro es el nombre de la clase.

alib[] es el nombre del arreglo.

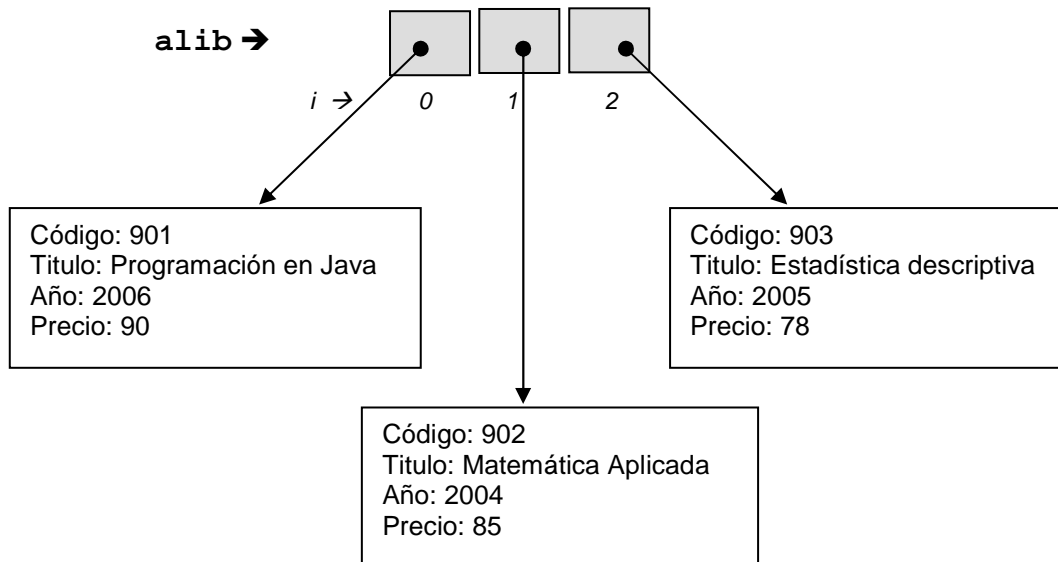
new es el operador para reservar memoria. En este ejemplo se esta reservando 3 espacios o variables de referencia que por defecto tienen el valor de null hasta que se almacene la dirección o referencia de un objeto.

b) Representación gráfica:



c) Creación y almacenamiento de objetos

```
alib[0]=new Libro(901,"Programación en Java", 2006, 90);  
alib[1]=new Libro(902,"Matemática Aplicada", 2004, 85);  
alib[2]=new Libro(903,"Estadística descriptiva", 2005, 78);
```



2. OPERACIONES VARIADAS

El siguiente ejemplo esta compuesto de 3 clases: la **clase Libro** (clase que maneja los atributos de los objetos de tipo libro y métodos de acceso set y get), la **clase ArregloLibros** (clase que maneja el arreglo de objetos) y la clase **Principal** (clase que interactúa con el usuario a través de la GUI).

La aplicación consiste en un mantenimiento de libros, es decir, se consideran las opciones de ingresar, consultar, modificar, eliminar y listar libros. Adicionalmente, se realiza un aumento de 15% al precio de todos los libros cuyo año de publicación sea mayor al 2003 y se muestra la siguiente información estadística:

- El título del libro más caro
- El título del libro más barato

Código de la clase Libro

```
package semana6;
```

```
public class Libro {

    // Atributos privados
    private int codigo, año;
    private String titulo;
    private double precio;

    // Constructor
    public Libro( int codigo, String titulo, int año, double
    precio){
        this.codigo = codigo;
        this.titulo = titulo;
        this.año = año;
        this.precio = precio;
    }

    //Métodos de acceso
    public void setCodigo( int codigo ){
        this.codigo = codigo;
    }

    public void setTitulo( String titulo ){
        this.titulo = titulo;
    }

    public void setAño( int año ){
        this.año = año;
    }

    public void setPrecio( double precio ){
        this.precio = precio;
    }

    public int getCodigo(){
        return codigo;
    }
}
```

```
    public String getTitulo(){
        return titulo;
    }

    public int getAño(){
        return año;
    }

    public double getPrecio(){
        return precio;
    }
}
```

Código de la clase ArregloLibros

```
package semana6;

public class ArregloLibros {

    //Atributos privados
    private Libro alib[];
    private int indice;

    //Constructor
    public ArregloLibros() {
        alib = new Libro[5];
        indice=0;
    }

    //Métodos públicos
    public void adicionar(Libro x){
        if(indice==alib.length)
            ampliarArreglo();

        alib[indice]=x;
        indice++;
    }

    public int tamaño(){
        return indice;
    }

    public Libro obtener(int pos){
        return alib[pos];
    }

    public int buscar(int cod){
        for(int i=0; i<indice; i++)
            if(cod==alib[i].getCodigo())
                return i;

        return -1;
    }
}
```

```
public void eliminar(int pos){
    for(int i=pos; i<indice-1; i++){
        alib[i]=alib[i+1];

        indice--;
    }

    public void aumento(){
        for(int i=0; i<indice; i++){
            Libro x=alib[i];
            if(x.getAño(>2003)
                x.setPrecio(x.getPrecio()*1.15);
        }
    }

    public String masCaro(){
        double max=0;
        String tit="";

        for(int i=0; i<indice; i++){
            Libro x=alib[i];
            if(x.getPrecio(>max){
                max=x.getPrecio();
                tit=x.getTitulo();
            }
        }
        return tit;
    }

    public String masBarato(){
        double min=Double.MAX_VALUE;
        String tit="";

        for(int i=0; i<indice; i++){
            Libro x=alib[i];
            if(x.getPrecio(<min){
                min=x.getPrecio();
                tit=x.getTitulo();
            }
        }
        return tit;
    }

    //Método privado
    private void ampliarArreglo(){
        Libro aux[]=alib;
        alib=new Libro[indice+5];
        for(int i=0; i<indice; i++){
            alib[i]=aux[i];
        }
    }
}
```

Código de la clase Principal

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import semana6.*;

public class Principal extends JApplet implements
ActionListener, ItemListener{
    // Declaración de variables
    JButton btnBorrar, btnProcesar, btnAumento, btnEstadisticas;
    JLabel lblOpcion, lblCodigo, lblTitulo, lblAno, lblPrecio;
    JTextField txtCodigo, txtTitulo, txtAno, txtPrecio;
    JComboBox cboOpcion;
    JTextArea txtS;
    JScrollPane scpScroll;

    ArregloLibros a = new ArregloLibros();

    // Crea la interfaz gráfica de usuario GUI
    public void init() {
        getContentPane().setLayout(null);
        getContentPane().setBackground(new
            Color(214,211,206));

        lblOpcion = new JLabel("Opción");
        lblOpcion.setBounds(15, 15, 90, 23);
        add(lblOpcion);

        lblCodigo = new JLabel("Codigo");
        lblCodigo.setBounds(15, 39, 90, 23);
        add(lblCodigo);

        lblTitulo = new JLabel("Titulo");
        lblTitulo.setBounds(15, 63, 90, 23);
        add(lblTitulo);

        lblAno = new JLabel("Año");
        lblAno.setBounds(15, 87, 90, 23);
        add(lblAno);

        lblPrecio = new JLabel("Precio");
        lblPrecio.setBounds(15, 111, 90, 23);
        add(lblPrecio);

        cboOpcion = new JComboBox();
        cboOpcion.setBounds(105, 15, 150, 23);
        cboOpcion.addItem("Ingresar");
        cboOpcion.addItem("Modificar");
        cboOpcion.addItem("Consultar");
        cboOpcion.addItem("Eliminar");
        cboOpcion.addItem("Listar");
        cboOpcion.addItemListener(this);
        add(cboOpcion);
```

```
txtCodigo = new JTextField();
txtCodigo.setBounds(105, 39, 150, 23);
add(txtCodigo);

txtTitulo = new JTextField();
txtTitulo.setBounds(105, 63, 150, 23);
add(txtTitulo);

txtAño = new JTextField();
txtAño.setBounds(105, 87, 150, 23);
add(txtAño);

txtPrecio = new JTextField();
txtPrecio.setBounds(105, 111, 150, 23);
add(txtPrecio);

btnProcesar = new JButton("Procesar");
btnProcesar.setBounds(365, 15, 110, 23);
btnProcesar.addActionListener(this);
add(btnProcesar);

btnBorrar = new JButton("Borrar");
btnBorrar.setBounds(365, 39, 110, 23);
btnBorrar.addActionListener(this);
add(btnBorrar);

btnAumento = new JButton("Aumento");
btnAumento.setBounds(365, 63, 110, 23);
btnAumento.addActionListener(this);
add(btnAumento);

btnEstadisticas = new JButton("Estadísticas");
btnEstadisticas.setBounds(365, 87, 110, 23);
btnEstadisticas.addActionListener(this);
add(btnEstadisticas);

txtS = new JTextArea();
txtS.setFont(new Font("monospaced", Font.PLAIN, 12));
txtS.setEditable(false);

scpScroll = new JScrollPane(txtS);
scpScroll.setBounds(15, 140, 450, 227);
add(scpScroll);
}

//-----
public void actionPerformed( ActionEvent e ){
    if( e.getSource() == btnProcesar )
        procesar();

    if( e.getSource() == btnBorrar )
        borrar();

    if( e.getSource() == btnAumento )
        aumento();
}
```

```

        if( e.getSource() == btnEstadisticas )
            estadisticas();
    }
    //-----
    public void itemStateChanged(ItemEvent e){
        if( e.getSource() == cboOpcion )
            seleccionar();
    }
    //-----
    void seleccionar(){
        switch(cboOpcion.getSelectedIndex()){
            case 0:
            case 1:
                lblCodigo.setVisible(true);
                txtCodigo.setVisible(true);
                lblTitulo.setVisible(true);
                txtTitulo.setVisible(true);
                lblAno.setVisible(true);
                txtAno.setVisible(true);
                lblPrecio.setVisible(true);
                txtPrecio.setVisible(true);
                break;
            case 2:
            case 3:
                lblCodigo.setVisible(true);
                txtCodigo.setVisible(true);
                lblTitulo.setVisible(false);
                txtTitulo.setVisible(false);
                lblAno.setVisible(false);
                txtAno.setVisible(false);
                lblPrecio.setVisible(false);
                txtPrecio.setVisible(false);
                break;
            default:
                lblCodigo.setVisible(false);
                txtCodigo.setVisible(false);
                lblTitulo.setVisible(false);
                txtTitulo.setVisible(false);
                lblAno.setVisible(false);
                txtAno.setVisible(false);
                lblPrecio.setVisible(false);
                txtPrecio.setVisible(false);
        }
    }
    //-----
    void borrar(){
        txtCodigo.setText("");
        txtTitulo.setText("");
        txtAno.setText("");
        txtPrecio.setText("");
        txtCodigo.requestFocus();
    }
    //-----
    void procesar(){
        switch(cboOpcion.getSelectedIndex()){

```

```

        case 0 :
            ingresar();
            break;

        case 1:
            modificar();
            break;

        case 2:
            consultar();
            break;

        case 3:
            eliminar();
            break;

        default:
            listar();

    }

}

//-----
int getCodigo(){
    return Integer.parseInt(txtCodigo.getText());
}

//-----
String getTitulo(){
    return txtTitulo.getText();
}

//-----
int getAño(){
    return Integer.parseInt(txtAño.getText());
}

//-----
double getPrecio(){
    return Double.parseDouble(txtPrecio.getText());
}

//-----
void imprimir(){
    txtS.setText("");
}

//-----
void imprimir( String s ){
    txtS.append( s + "\n");
}

//-----
void mensaje( String s){
    JOptionPane.showMessageDialog(this,s);
}

//-----
void listar(){
    imprimir();
    if(a.tamaño()>0){
        imprimir("Código \t Título \t Año \t Precio");
        for(int i=0; i<a.tamaño(); i++){
            Libro x =a.obtener(i);

```

```

        imprimir(x.getCodigo()+"\t"+x.getTitulo()+"\t"+
                x.getAño()+"\t"+x.getPrecio());
    }
}
else
    mensaje("No hay Libros");
}
//-----
void ingresar() {
    int pos = a.buscar(getCodigo());
    if(pos == -1){
        Libro x = new Libro(getCodigo() ,getTitulo(),
            getAño(),getPrecio());
        a.adicionar(x);
        listar();
        mensaje("Libro agregado");
    }
    else
        mensaje("Código ya existe");
}
//-----
void modificar() {
    int pos = a.buscar(getCodigo());
    if(pos != -1){
        Libro x = a.obtener(pos);
        x.setTitulo(getTitulo());
        x.setAño(getAño());
        x.setPrecio(getPrecio());
        listar();
        mensaje("Libro modificado");
    }
    else
        mensaje("Código no existe");
}
//-----
void consultar() {
    int pos = a.buscar(getCodigo());
    if(pos != -1){
        Libro x = a.obtener(pos);
        imprimir();
        imprimir("Código:" + x.getCodigo());
        imprimir("Título:" + x.getTitulo());
        imprimir("Año:" + x.getAño());
        imprimir("Precio:" + x.getPrecio());
    }
    else
        mensaje("Código no existe");
}
//-----
void eliminar() {
    int pos = a.buscar(getCodigo());
    if(pos != -1){
        a.eliminar(pos);
    }
}

```



```

        mensaje("Libro eliminado");
        listar();
    }
    else
        mensaje("Código no existe");
}
//-----
void aumento(){
    a.aumento();
    listar();
}
//-----
void estadisticas(){
    imprimir();
    imprimir("El título del libro más caro:" +
a.masCaro());
    imprimir("El título del libro más barato:" +
a.masBarato());
}
}

```

GUI del Problema:

AppletViewer: Principal.class

Subprograma

Opción	Ingresar ▼	Procesar
Codigo	<input type="text"/>	Borrar
Titulo	<input type="text"/>	Aumento
Año	<input type="text"/>	Estadisticas
Precio	<input type="text"/>	

Subprograma iniciado.

Ejercicios

- 1) Asuma la existencia de la clase Empleado que cuenta con los siguientes atributos privados: código (entero), nombre (cadena), horas trabajadas (entero) y tarifa (real). Considere que la clase cuenta con un constructor que inicializa los atributos y los métodos de acceso set y get para cada atributo. Diseñe las opciones de ingresar, modificar y listar del mantenimiento de empleados. Para ello, implemente las clases ArregloEmpleados (clase que maneja el arreglo de objetos) y la clase Principal (clase que controla la GUI).
Adicionalmente, implemente, en la clase ArregloEmpleados, los siguientes métodos:
 - a) Diseñe un método que aumente la tarifa a los empleados cuyo nombre empiece con la letra 'R'.
 - b) Diseñe un método que retorne el nombre del empleado que tiene la tarifa más alta.

- 2) Asuma la existencia de la clase Vendedor que cuenta con los siguientes atributos privados: código (entero), nombre (cadena), y monto vendido (real). Considere que la clase cuenta con un constructor que inicializa los atributos y los métodos de acceso set y get para cada atributo. Diseñe las opciones de ingresar, consultar y eliminar del mantenimiento de vendedores. Para ello, implemente las clases ArregloVendedores (clase que maneja el arreglo de objetos) y la clase Principal (clase que controla la GUI).
Adicionalmente, implemente, en la clase ArregloVendedores, los siguientes métodos:
 - a) Diseñe un método que retorne el monto promedio de aquellos vendedores cuyo nombre empiece con 'A'.
 - b) Diseñe un método que retorne el nombre del vendedor que vendió menos.

**UNIDAD DE
APRENDIZAJE****3****SEMANA****8-9**

CLASE ARRAYLIST Y ARCHIVOS DE TEXTO

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos utilizan los métodos de la clase ArrayList para efectuar operaciones con objetos (ingresar, consultar, eliminar, modificar, listar entre otras). Utilizan las clases BufferedReader, FileReader, PrintWriter, FileWriter y StringTokenizer para almacenar la data en archivos de texto.

TEMARIO

- Métodos de la clase ArrayList
- Operaciones con objetos

ACTIVIDADES PROPUESTAS

- Los alumnos emplean los métodos de la clase ArrayList para manipular un arreglo de objetos.
- Los alumnos crean un mantenimiento.

1. CLASE ARRAYLIST

Métodos de la clase ArrayList

Esta Clase dispone de diversos métodos para manipular una colección de objetos dinámicamente. Para crear un ArrayList, se utiliza la siguiente instrucción:

```
ArrayList <nombre_clase> var_referencia = new ArrayList < nombre_clase > ();
```

Ejemplo:

```
ArrayList <Producto> prod = new ArrayList <Producto> ();
```

prod es un **objeto** de tipo **ArrayList** que va a **manipular un arreglo de objetos** de tipo **Producto**.

Es necesario importar el paquete:

```
java.util.ArrayList;
```

MÉTODOS	DESCRIPCIÓN
<code>add(Object)</code>	Agrega un elemento al final. <pre>public void adicionar(Producto x){ prod.add(x); }</pre>
<code>add(int, Object)</code>	Agrega un elemento en la posición especificada en el primer parámetro. <pre>prod.add(0,x);</pre>
<code>clear()</code>	Elimina todos los elementos. <pre>prod.clear();</pre>
<code>get(int)</code>	Devuelve el elemento de la posición especificada. <pre>public Producto obtener(int pos){ return prod.get(pos); }</pre>
<code>indexOf(Object)</code>	Devuelve el índice del elemento especificado, de no encontrarlo devuelve -1. <pre>public int posicion(Producto x){ return prod.indexOf(x); }</pre>

<code>remove(int)</code>	Elimina el elemento de la posición especificada. <pre>public void eliminar(int x){ prod.remove(x); }</pre>
<code>remove(Object)</code>	Elimina el elemento especificado. <pre>public void eliminar(Producto x){ prod.remove(x); }</pre>
<code>set(int, Object)</code>	Reemplaza el elemento de la posición especificada en el primer parámetro por el elemento del segundo parámetro. <pre>public void modificar(int pos, Producto x){ prod.set(pos,x); }</pre>
<code>size()</code>	Devuelve la cantidad de elementos agregados. <pre>public int tamaño(){ return prod.size(); }</pre>

2. Operaciones con objetos

a) Operaciones de mantenimiento:

Empleando la clase `ArrayList` realice un mantenimiento de productos. Para ello, implemente las clases `Producto`, `ArregloProductos` y `Principal`.

Paso 1

Diseñe la clase **Producto** en el paquete `semana8` que tenga como atributos privados los siguientes datos: código, descripción, precio unitario y stock. Implemente métodos de acceso **set/get**

```
package semana8;
public class Producto{
    // Atributos privados
    private int codigo,stock;
    private String descripcion;
    private double pu;

    // Constructor
    public Producto( int codigo, String descripcion, double pu,
    int stock ){
        this.codigo=codigo;
        this.descripcion=descripcion;
        this.pu=pu;
        this.stock=stock;
    }
}
```

```

    // Fija el código
    public void setCodigo( int codigo ){
        this.codigo = codigo;
    }

    // Fija la descripción
    public void setDescription( String descripcion ){
        this.descripcion = descripcion;
    }

    // Fija el precio unitario
    public void setPu( double pu ){
        this.pu = pu;
    }

    // Fija el stock
    public void setStock( int stock ){
        this.stock = stock;
    }

    // Retorna el código
    public int getCodigo(){
        return codigo;
    }

    // Retorna la descripción
    public String getDescripcion(){
        return descripcion;
    }

    // Retorna el precio unitario
    public double getPu(){
        return pu;
    }

    // Retorna el stock
    public int getStock(){
        return stock;
    }
}

```

Paso 2

Diseñe la clase **ArregloProductos** en el paquete **semana8** que tenga como atributo privado un objeto de tipo **ArrayList**; luego, implemente los siguientes métodos:

- Un constructor sin parámetros que cree el objeto **ArrayList**
- Un método **adicionar** que reciba un producto y lo adicione al **ArrayList**
- Un método **eliminar** que reciba un producto y lo elimine del **ArrayList**
- Un método **obtener** que reciba una posición y retorne el producto de esa posición
- Un método **buscar** que reciba un código y retorne el producto con ese código
- Un método **tamaño** que retorne la cantidad de productos ingresados al **ArrayList**
- Un método **mayorprecio** que retorne el producto más caro
- Un método **menorprecio** que retorne el producto más barato

```
package semana8;

import java.util.ArrayList;

public class ArregloProductos{
    private ArrayList <Producto> prod;

    public ArregloProductos(){
        prod=new ArrayList <Producto> ();
    }

    public void adicionar(Producto x){
        prod.add(x);
    }

    public void eliminar(Producto x){
        prod.remove(x);
    }

    public Producto obtener(int pos){
        return prod.get(pos);
    }

    public Producto buscar(int codigo){
        for(int i=0; i<prod.size(); i++)
            if(codigo==prod.get(i).getCodigo())
                return prod.get(i);

        return null;
    }

    public int tamaño(){
        return prod.size();
    }

    public double mayorPrecio(){
        double max=prod.get(0).getPu();
        for(int i=1; i<prod.size(); i++)
            if(prod.get(i).getPu(>max)
                max=prod.get(i).getPu();

        return max;
    }

    public double menorPrecio(){
        double min=prod.get(0).getPu();
        for(int i=1; i<prod.size(); i++)
            if(prod.get(i).getPu(<min)
                min=prod.get(i).getPu();

        return min;
    }
}
```

Paso 3

Una vez desarrollada las clases **Producto** y **ArregloProductos**, implemente los siguientes métodos del programa principal: ingreso, consulta, modificación, eliminación y listado.

En todo momento, el programa mostrará un listado como el siguiente:

Código	Descripción	P_Unitario	Stock
901	Lavadora	600	10
902	Equipo	450	12
903	Refrigerador	1550	8

 Total de Productos : 3
 Producto más caro : 1550
 Producto más barato : 450

GUI:

AppletViewer: Principal.class

Subprograma

Opción: Ingresar

Codigo: 903

Descripcion: Refrigerador

Precio: 1550

Stock: 8

Procesar

Borrar

Codigo	Descripción	P_Unitario	Stock
901	Lavadora	600.0	10
902	Equipo	450.0	12
903	Refrigerador	1550.0	8

Total de Productos:3
 Producto mas caro:1550.0
 Producto mas barato:450.0

Subprograma iniciado.


```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import semana8.*;

public class Principal extends JApplet implements
ActionListener, ItemListener{
    // Declaración de variables
    JButton btnProcesar, btnBorrar;
    JLabel lblOpcion, lblCodigo, lblDescripcion, lblPrecio,
    lblStock;
    JTextField txtCodigo, txtDescripcion, txtPrecio, txtStock;
    JComboBox cboOpcion;
    JTextArea txtS;
    JScrollPane scpScroll;

    //Creo el objeto de tipo ArregloProductos
    ArregloProductos p=new ArregloProductos();

    // Crea la interfaz gráfica de usuario GUI
    public void init() {
        getContentPane().setLayout(null);
        getContentPane().setBackground(new
        Color(214,211,206));

        lblOpcion = new JLabel("Opción");
        lblOpcion.setBounds(15, 15, 90, 23);
        getContentPane().add(lblOpcion);

        lblCodigo = new JLabel("Codigo");
        lblCodigo.setBounds(15, 39, 90, 23);
        getContentPane().add(lblCodigo);

        lblDescripcion = new JLabel("Descripcion");
        lblDescripcion.setBounds(15, 63, 90, 23);
        getContentPane().add(lblDescripcion);

        lblPrecio = new JLabel("Precio");
        lblPrecio.setBounds(15, 87, 90, 23);
        getContentPane().add(lblPrecio);

        lblStock = new JLabel("Stock");
        lblStock.setBounds(15, 111, 90, 23);
        getContentPane().add(lblStock);

        cboOpcion = new JComboBox();
        cboOpcion.setBounds(105, 15, 150, 23);
        cboOpcion.addItem("Ingresar");
        cboOpcion.addItem("Consultar");
        cboOpcion.addItem("Modificar");
        cboOpcion.addItem("Eliminar");
        cboOpcion.addItemListener(this);
        getContentPane().add(cboOpcion);

        txtCodigo = new JTextField();
        txtCodigo.setBounds(105, 39, 150, 23);
```

```

        getContentPane().add(txtCodigo);

        txtDescripcion = new JTextField();
        txtDescripcion.setBounds(105,63, 150, 23);
        getContentPane().add(txtDescripcion);

        txtPrecio = new JTextField();
        txtPrecio.setBounds(105, 87, 150, 23);
        getContentPane().add(txtPrecio);

        txtStock = new JTextField();
        txtStock.setBounds(105, 111, 150, 23);
        getContentPane().add(txtStock);

        btnProcesar = new JButton("Procesar");
        btnProcesar.setBounds(365, 15, 101, 23);
        btnProcesar.addActionListener(this);
        getContentPane().add(btnProcesar);

        btnBorrar = new JButton("Borrar");
        btnBorrar.setBounds(365, 39, 101, 23);
        btnBorrar.addActionListener(this);
        getContentPane().add(btnBorrar);

        txtS = new JTextArea();
        txtS.setFont(new Font("courier", Font.PLAIN, 12));

        scpScroll=new JScrollPane(txtS);
        scpScroll.setBounds(15, 140, 450, 227);
        getContentPane().add(scpScroll);
    }

    //-----
    public void actionPerformed( ActionEvent e ){
        if( e.getSource() == btnProcesar )
            procesar();

        if( e.getSource() == btnBorrar )
            borrar();
    }
    //-----
    public void itemStateChanged(ItemEvent e){
        if( e.getSource() == cboOpcion )
            seleccionar();
    }
    //-----
    void seleccionar(){
        int opcion=cboOpcion.getSelectedIndex();
        if( opcion == 1 || opcion == 3 ){
            lblDescripcion.setVisible(false);
            txtDescripcion.setVisible(false);
            lblPrecio.setVisible(false);
            txtPrecio.setVisible(false);
            lblStock.setVisible(false);
            txtStock.setVisible(false);
        }
    }

```

```

        else{
            lblDescripcion.setVisible(true);
            txtDescripcion.setVisible(true);
            lblPrecio.setVisible(true);
            txtPrecio.setVisible(true);
            lblStock.setVisible(true);
            txtStock.setVisible(true);
        }
    }
//-----
void borrar(){
    txtCodigo.setText("");
    txtDescripcion.setText("");
    txtPrecio.setText("");
    txtStock.setText("");
    txtCodigo.requestFocus();
}
//-----
void procesar(){
    switch(cboOpcion.getSelectedIndex()){
        case 0 :
            ingresar();
            break;
        case 1:
            consultar();
            break;
        case 2:
            modificar();
            break;
        default:
            eliminar();
    }
}
//-----
int getCodigo(){
    return Integer.parseInt(txtCodigo.getText());
}
//-----
String getDescripcion(){
    return txtDescripcion.getText();
}
//-----
double getPrecio(){
    return Double.parseDouble(txtPrecio.getText());
}
//-----
int getStock(){
    return Integer.parseInt(txtStock.getText());
}
//-----
void mensaje( String m, String tipo ){
    JOptionPane.showMessageDialog(this,
        m,tipo,JOptionPane.INFORMATION_MESSAGE);
}
//-----
String rellena(String cad){

```

```

        int longitud=cad.length();
        for(int i=longitud; i<15; i++)
            cad+=" ";
        return cad;
    }
    //-----
    void imprimir(){
        txtS.setText("");
    }
    //-----
    void imprimir(String s){
        txtS.append( s + "\n");
    }
    //-----
    // Ingresa un producto evitando que el código se repita
    void ingresar(){
        Producto prod=p.buscar(getCodigo());
        if(prod==null){
            prod=new Producto(getCodigo(),
                getDescripcion(),getPrecio(),getStock());
            p.adicionar(prod);
            listar();
            mensaje("Producto Ingresado", "Mensaje de
                confirmación");
        }
        else
            mensaje("Codigo ya existe", "Mensaje de error");
    }
    //-----
    // Muestra el listado solicitado
    void listar(){
        imprimir();
        if(p.tamaño(>0){
            imprimir("Codigo \t Descripción \t\t P_Unitario
                \t Stock");

            for(int i=0; i<p.tamaño(); i++){
                Producto prod=p.obtener(i);
                imprimir(prod.getCodigo()+"\t" +
                    rellena(prod.getDescripcion())+"\t\t"+
                    prod.getPu() + "\t\t"+prod.getStock());
            }
            imprimir("");
            imprimir("Total de Productos:"+p.tamaño());
            imprimir("Producto mas caro:"+p.mayorPrecio());
            imprimir("Producto mas barato:"+
                p.menorPrecio());
        }
        else
            imprimir("No hay productos");
    }
    //-----
    // Consulta la descripción, precio y stock de un producto
    dado su código

```

```
void consultar(){
    imprimir();
    Producto prod=p.buscar(getCodigo());
    if(prod!=null){
        imprimir("Código           : "+
        prod.getCodigo());
        imprimir("Descripción       : "+
        prod.getDescripcion());
        imprimir("Precio Unitario   : "+prod.getPu());
        imprimir("Stock            : "+
        prod.getStock());
    }
    else
        mensaje("Producto no existe", "Mensaje de
        error");
}
//-----
// Modifica la descripción, precio y stock de un producto
dado su código
void modificar(){
    Producto prod=p.buscar(getCodigo());
    if(prod!=null){
        prod.setDescripcion(getDescripcion());
        prod.setPu(getPrecio());
        prod.setStock(getStock());
        listar();
        mensaje("Producto Modificado", "Mensaje de
        confirmación");
    }
    else
        mensaje("Producto no existe", "Mensaje de
        error");
}
//-----
// Elimina un producto dado su código
void eliminar(){
    Producto prod=p.buscar(getCodigo());
    if(prod!=null){
        p.eliminar(prod);
        listar();
        mensaje("Producto Eliminado", "Mensaje de
        confirmación");
    }
    else
        mensaje("Producto no existe", "Mensaje de
        error");
}
}
```

En el mantenimiento, hemos visto que para realizar la opción de modificación se han utilizado los métodos de acceso set para modificar los atributos de un producto. A continuación, se muestra otra forma de obtener los mismos resultados:

Agregue los siguientes métodos en la clase **ArregloProductos**

```
public int posicion(Producto x){
    return prod.indexOf(x);
}

public void modificar(int pos, Producto x){
    prod.set(pos,x);
}
```

En la clase **Principal**, sustituya el método modificar por el siguiente:

```
// Modifica la descripción, precio y stock de un producto dado su código
void modificar(){
    Producto prod=p.buscar(getCodigo());
    if(prod!=null){
        Producto x=new Producto(
            getCodigo(),getDescripcion(),getPrecio(),getStock());
        p.modificar(p.posicion(prod),x);
        listar();
        mensaje("Producto Modificado", "Mensaje de confirmación");
    }
    else
        mensaje("Producto no existe", "Mensaje de error");
}
```

Esta segunda forma consiste en reemplazar el nuevo objeto de tipo Producto por el anterior.

b) Operaciones variadas:

En la clase **Principal**, implemente los siguientes métodos:

- 1) Diseñe un método que muestre los productos que empiecen con la letra 'm' cuyo precio unitario se encuentre en el rango de precios ingresados desde la GUI.

```
void mostrar(){
    imprimir("Codigo \t Descripción \t P_Unitario \t Stock");
    for(int i=0; i<p.tamaño(); i++){
        Producto x = p.obtener(i);
        char letra = x.getDescripcion().charAt(0);
        if(letra=='m' && x.getPu()>=getPreIni() &&
            x.getPu()<=getPreFin())
            imprimir(x.getCodigo()+"\t"+x.getDescripcion()+"\t"+
                x.getPu()+"\t"+x.getStock());
    }
}
```

- 2) Diseñe un método que muestre los productos que terminen con la letra 'a' cuyo stock sea cero.

```
void mostrar(){
    imprimir("Codigo \t Descripción \t P_Unitario \t Stock");
    for(int i=0; i<p.tamaño(); i++){
        Producto x = p.obtener(i);
        int pos = x. getDescripcion().length()-1;
        char letra = x. getDescripcion().charAt(pos);
        if(letra=='a' && x. getStock()==0)
            imprimir(x.getCodigo()+"\t"+x.getDescripcion()+"\t"+
                x.getPu()+"\t"+x.getStock());
    }
}
```

- 3) Diseñe un método que disminuya en 10% el precio unitario de aquellos productos cuyo stock sea mayor a 100.

```
void rebaja(){
    for(int i=0; i<p.tamaño(); i++){
        Producto x = p.obtener(i);
        if(x.getStock(>100)
            x.setPu(x.getPu()*0.90);
    }
    listar();
}
```

- 4) Diseñe un método que elimine a los productos cuyo stock sea menor al stock ingresado desde la GUI.

```
void eliminarVarios(){
    for(int i=0; i<p.tamaño(); i++){
        Producto x = p.obtener(i);
        if(x.getStock(<getStock())){
            p.eliminar(x);
            i--;
        }
    }
    listar();
}
```

Ejercicios

- 1) Asuma la existencia de la clase Celular que cuenta con los siguientes atributos privados: código (entero), marca (cadena), modelo (cadena) y precio unitario (real). Considere que la clase cuenta con un constructor que inicializa los atributos y los métodos de acceso set y get para cada atributo. Diseñe las opciones de ingresar, modificar y listar del mantenimiento de celulares. Para ello, implemente las clases ArregloCelulares (clase que maneja un objeto privado de tipo ArrayList) y la clase Principal (clase que controla la GUI).
Adicionalmente, implemente en la clase ArregloCelulares los siguientes métodos:

- a. Diseñe un método que aumente en 8% el precio unitario a los celulares cuya marca termine con la letra 'a'.
- b. Diseñe un método que retorne los modelos de celular de la marca enviada como parámetro.

- 2) Asuma la existencia de la clase Video que cuenta con los siguientes atributos privados: codVideo (entero), nombre de película (cadena), codGenero (0=comedia, 1=suspenso, 2=terror) y precio unitario (real). Considere que la clase cuenta con un constructor que inicializa los atributos y los métodos de acceso set y get para cada atributo. Diseñe las opciones de ingresar, consultar y eliminar del mantenimiento de videos. Para ello, implemente las clases ArregloVideos (clase que maneja un objeto privado de tipo ArrayList) y la clase Principal (clase que controla la GUI).
Adicionalmente, implemente en la clase Principal los siguientes métodos:

- a. Diseñe un método que imprima el precio unitario promedio de aquellos videos del género suspenso.
- b. Diseñe un método que elimine los videos del género ingresado desde la GUI.

**UNIDAD DE
APRENDIZAJE****3****SEMANA****10**

ARCHIVOS DE TEXTO

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos utilizan los métodos de la clase ArrayList para efectuar operaciones con objetos (ingresar, consultar, eliminar, modificar, listar, entre otras). Utilizan las clases BufferedReader, FileReader, PrintWriter, FileWriter y StringTokenizer para almacenar la data en archivos de texto.

TEMARIO

- Configuración del JDK
- Clases y métodos para manipular archivos de texto.
- Operaciones de lectura y escritura en archivos de texto

ACTIVIDADES PROPUESTAS

- Los alumnos emplean las clases BufferedReader, FileReader, PrintWriter, FileWriter y StringTokenizer para almacenar un arreglo de objetos en archivos de texto. Para manipular el arreglo de objetos, utilizan los métodos de la clase ArrayList.

1. CONFIGURACIÓN DEL JDK

Lo primero que debemos hacer para poder manejar una aplicación usando archivos es configurar el JDK; por esta razón, siga los siguientes pasos:

1. Desde el Explorador ejecute `java\jdk1.6.0\bin\policytool`.
2. En la ventana de Error pulse el botón Aceptar.
3. Haga click en Archivo - Abrir.
Buscar en `java\jdk1.6.0\jre\lib\security\java.policy`
4. Seleccione CodeBase <ALL> y pulse "Editar entrada de norma".
5. Pulse "Agregar permiso".
6. En "Permiso", seleccione "FilePermission".
7. En "Nombre de destino", seleccione "<<ALL FILES>>".
8. En "Acciones", seleccione "read, write, delete, execute".
9. Pulse "Aceptar", luego, "Terminar".
10. Haga click en Archivo – Guardar.
11. Al mensaje "Norma escrita satisfactoriamente..." dele Aceptar.
12. Por último cierre la ventana.

Recomendaciones:

- Debemos incluir la librería que permite manejar archivos (input - output).

```
import java.io.*;
```

- Todo código que involucre manejo de archivos debe estar en una estructura try-catch, ya que podría producirse algún error; por ejemplo, si no existe el archivo.

2. CLASES Y MÉTODOS PARA EL MANEJO DE ARCHIVOS:

BufferedReader.- Es la clase que permite leer los datos de un archivo a memoria (input). Lo primero que tenemos que hacer es crear una variable de tipo `BufferedReader` por ejemplo:

```
BufferedReader br;
```

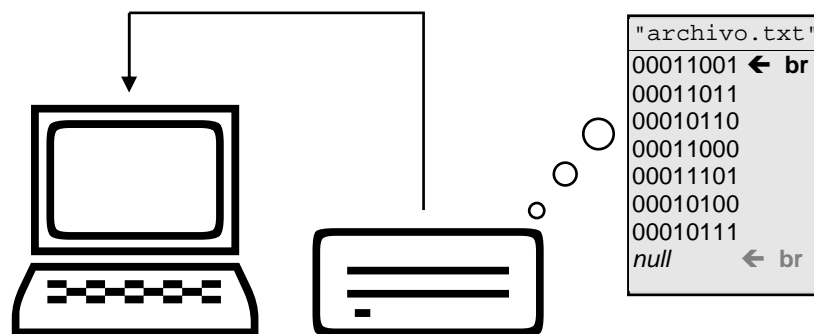
Luego, invocamos al constructor de la clase `BufferedReader` y le mandamos como parámetro un objeto de tipo `FileReader`. De esta forma:

```
br = new BufferedReader(new FileReader(archivo));
```

La clase **FileReader**, abre un archivo como sólo lectura.

- Una vez abierto el archivo, `br` apunta a la primera cadena de bits.
- El método **readLine()** asociado a `br` captura una cadena de bits y salta a la siguiente línea de bits. Dicha cadena es convertida seguidamente al tipo de dato requerido.
- Cuando no existen más cadenas `br` apunta a `null`.
- El método **close()** cierra el acceso al archivo.

```
BufferedReader br = new BufferedReader(new FileReader(archivo));
```



PrintWriter.- Es la clase que permite escribir los datos de la memoria hacia un archivo (output). Lo primero que tenemos que hacer es crear una variable de tipo `PrintWriter` por ejemplo:

```
PrintWriter pw;
```

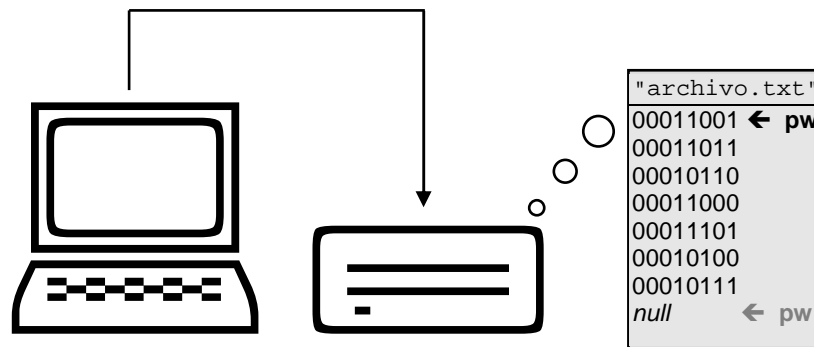
Luego, invocamos al constructor de la clase `PrintWriter` y le mandamos como parámetro un objeto de tipo `FileWriter`. De esta forma:

```
pw = new PrintWriter(new FileWriter(archivo));
```

La clase **FileWriter**, se encarga de abrir el archivo en modo de escritura. Es decir, si el archivo contiene información ésta se pierde. Si el archivo no existe, lo crea.

- Una vez abierto el archivo, pw apunta al inicio.
- El método **println(data)** asociado a pw graba como cadena de bits la data indicada y genera un salto de línea en el archivo.
- El método **close()** cierra el acceso al archivo.

```
PrintWriter pw = new PrintWriter(new FileWriter(archivo));
```



3. OPERACIONES DE LECTURA Y ESCRITURA

Diseñe un mantenimiento de estudiantes que permita ingresar, consultar, modificar, eliminar y listar estudiantes. Para ello, cree las clases Estudiante, ArregloEstudiante (en el paquete semana10) y Principal. Cree los métodos cargar y grabar en la clase ArregloEstudiantes. Al cargar el JApplet, se deberán leer los datos del archivo estudiantes.txt; si el archivo no existe, deberá aparecer un mensaje de error.

Código de la clase Estudiante

```
package semana10;

public class Estudiante{

    // Atributos privados
    private int codigo, ciclo;
    private String nombre;
    private double pension;

    // Constructor
    public Estudiante( int codigo, String nombre, int ciclo,
    double pension){
        this.codigo = codigo;
        this.nombre = nombre;
        this.ciclo = ciclo;
        this.pension = pension;
    }
}
```

```
    }

    // Métodos de acceso: set/get
    public void setCodigo( int codigo ){
        this.codigo = codigo;
    }

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

    public void setCiclo( int ciclo ){
        this.ciclo = ciclo;
    }

    public void setPension( double pension ){
        this.pension = pension;
    }

    public int getCodigo(){
        return codigo;
    }

    public String getNombre(){
        return nombre;
    }

    public int getCiclo(){
        return ciclo;
    }

    public double getPension(){
        return pension;
    }
}
```

Código de la clase ArregloEstudiantes

```
package semanal0;

import java.util.*;
import javax.swing.*;
import java.io.*;

public class ArregloEstudiantes{

    //Atributo privado
    private ArrayList <Estudiante> est;

    //Constructor
    public ArregloEstudiantes() {
        est=new ArrayList <Estudiante> ();
        cargar();
    }

    //Métodos para manipular el arreglo de objetos
```

```

public void adicionar(Estudiante x){
    est.add(x);
}

public void eliminar(Estudiante x){
    est.remove(x);
}

public Estudiante obtener(int pos){
    return est.get(pos);
}

public Estudiante buscar(int codigo){
    for(int i=0; i<est.size(); i++)
        if(codigo==est.get(i).getCodigo())
            return est.get(i);

    return null;
}

public int tamaño(){
    return est.size();
}

//Métodos para manipular el archivo de texto
private void cargar(){
    try{
        File archivo = new File("estudiantes.txt");
        if(archivo.exists()){
            BufferedReader br=new BufferedReader(new
                FileReader("estudiantes.txt"));
            String linea;

            while((linea=br.readLine())!=null){
                StringTokenizer st=new
                    StringTokenizer(linea,",");

                int cod =
                    Integer.parseInt(st.nextToken().trim());
                String nom=st.nextToken().trim();
                int ciclo =
                    Integer.parseInt(st.nextToken().trim());
                double pension =
                    Double.parseDouble(st.nextToken().trim());

                Estudiante x =new Estudiante( cod,
                    nom,ciclo,pension);
                adicionar(x);
            }
            br.close();
        }
        else
            JOptionPane.showMessageDialog(null,"El
                archivo estudiantes.txt no existe");
    }
    catch(Exception x){

```

```

        JOptionPane.showMessageDialog(null,"Se produjo
        un error= "+x);
    }

}

public void grabar(){
    try{
        PrintWriter pw =new PrintWriter(new
        FileWriter("estudiantes.txt"));
        for(int i=0; i<tamaño(); i++)
            pw.println(obtener(i).getCodigo()+", "+
                obtener(i).getNombre()+", "+
                obtener(i).getCiclo()+", "+
                obtener(i).getPension());

        pw.close();
    }
    catch(Exception x){
        JOptionPane.showMessageDialog(null,"Se produjo
        un error= "+x);
    }
}
}

```

Código de la clase Principal

```

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import semana10.*;

public class Principal extends JApplet implements
ActionListener, ItemListener{
    // Declaración de variables
    JButton btnBorrar, btnProcesar;
    JLabel lblOpcion, lblCodigo, lblNombre, lblCiclo,
    lblPension;
    JTextField txtCodigo, txtNombre, txtCiclo, txtPension;
    JComboBox cboOpcion;
    JTextArea txtS;
    JScrollPane scpScroll;

    //Creo el objeto de tipo ArregloEstudiantes
    ArregloEstudiantes a=new ArregloEstudiantes();

    // Crea la interfaz gráfica de usuario GUI
    public void init() {
        getContentPane().setLayout(null);
        getContentPane().setBackground(new
        Color(214,211,206));

        lblOpcion = new JLabel("Opción");
        lblOpcion.setBounds(15, 15, 90, 23);
    }
}

```

```
add(lblOpcion);

lblCodigo = new JLabel("Codigo");
lblCodigo.setBounds(15, 39, 90, 23);
add(lblCodigo);

lblNombre = new JLabel("Nombre");
lblNombre.setBounds(15, 63, 90, 23);
add(lblNombre);

lblCiclo = new JLabel("Ciclo");
lblCiclo.setBounds(15, 87, 90, 23);
add(lblCiclo);

lblPension = new JLabel("Pension");
lblPension.setBounds(15, 111, 90, 23);
add(lblPension);

cboOpcion = new JComboBox();
cboOpcion.setBounds(105, 15, 150, 23);
cboOpcion.addItem("Ingresar");
cboOpcion.addItem("Modificar");
cboOpcion.addItem("Consultar");
cboOpcion.addItem("Eliminar");
cboOpcion.addItem("Listar");
cboOpcion.addItemListener(this);
add(cboOpcion);

txtCodigo = new JTextField();
txtCodigo.setBounds(105, 39, 150, 23);
add(txtCodigo);

txtNombre = new JTextField();
txtNombre.setBounds(105, 63, 150, 23);
add(txtNombre);

txtCiclo = new JTextField();
txtCiclo.setBounds(105, 87, 150, 23);
add(txtCiclo);

txtPension = new JTextField();
txtPension.setBounds(105, 111, 150, 23);
add(txtPension);

btnProcesar = new JButton("Procesar");
btnProcesar.setBounds(365, 15, 110, 23);
btnProcesar.addActionListener(this);
add(btnProcesar);

btnBorrar = new JButton("Borrar");
btnBorrar.setBounds(365, 39, 110, 23);
btnBorrar.addActionListener(this);
add(btnBorrar);

txtS = new JTextArea();
txtS.setFont(new Font("monospaced", Font.PLAIN, 12));
```



```

        txtS.setEditable(false);

        scpScroll = new JScrollPane(txtS);
        scpScroll.setBounds(15, 140, 450, 227);
        add(scpScroll);

        listar();
    }

    //-----
    public void actionPerformed( ActionEvent e ){
        if( e.getSource() == btnProcesar )
            procesar();

        if( e.getSource() == btnBorrar )
            borrar();
    }
    //-----
    public void itemStateChanged(ItemEvent e){
        if( e.getSource() == cboOpcion )
            seleccionar();
    }
    //-----
    void seleccionar(){
        switch(cboOpcion.getSelectedIndex()){
            case 0:
            case 1:
                lblCodigo.setVisible(true);
                txtCodigo.setVisible(true);
                lblNombre.setVisible(true);
                txtNombre.setVisible(true);
                lblCiclo.setVisible(true);
                txtCiclo.setVisible(true);
                lblPension.setVisible(true);
                txtPension.setVisible(true);
                break;
            case 2:
            case 3:
                lblCodigo.setVisible(true);
                txtCodigo.setVisible(true);
                lblNombre.setVisible(false);
                txtNombre.setVisible(false);
                lblCiclo.setVisible(false);
                txtCiclo.setVisible(false);
                lblPension.setVisible(false);
                txtPension.setVisible(false);
                break;
            default:
                lblCodigo.setVisible(false);
                txtCodigo.setVisible(false);
                lblNombre.setVisible(false);
                txtNombre.setVisible(false);
                lblCiclo.setVisible(false);
                txtCiclo.setVisible(false);
                lblPension.setVisible(false);
                txtPension.setVisible(false);
        }
    }

```

```

        txtPension.setVisible(false);
    }
}
//-----
void borrar(){
    txtCodigo.setText("");
    txtNombre.setText("");
    txtCiclo.setText("");
    txtPension.setText("");
    txtCodigo.requestFocus();
}
//-----
void procesar(){
    switch(cboOpcion.getSelectedIndex()){
        case 0 :
            ingresar();
            break;

        case 1:
            modificar();
            break;

        case 2:
            consultar();
            break;

        case 3:
            eliminar();
            break;

        default:
            listar();
    }
}
//-----
int getCodigo(){
    return Integer.parseInt(txtCodigo.getText());
}
//-----
String getNombre(){
    return txtNombre.getText();
}
//-----
int getCiclo(){
    return Integer.parseInt(txtCiclo.getText());
}
//-----
double getPension(){
    return Double.parseDouble(txtPension.getText());
}
//-----
void imprimir(){
    txtS.setText("");
}
//-----

```

```
void imprimir(String s){
    txtS.append( s + "\n");
}
//-----
void mensaje(String cad){
    JOptionPane.showMessageDialog(this,cad);
}
//-----
void listar(){
    imprimir();
    if(a.tamaño()>0){
        imprimir("Codigo \t Nombre \t Ciclo \t
        Pensión");

        for(int i=0; i<a.tamaño(); i++){
            Estudiante e =a.obtener(i);
            imprimir(e.getCodigo()+"\t" +
                    e.getNombre()+"\t"+e.getCiclo()+
                    "\t"+e.getPension());
        }
    }
    else
        imprimir("No hay estudiantes");
}
//-----
void ingresar(){
    Estudiante e =a.buscar(getCodigo());
    if(e == null){
        e = new Estudiante( getCodigo(),getNombre(),
                            getCiclo(),getPension());

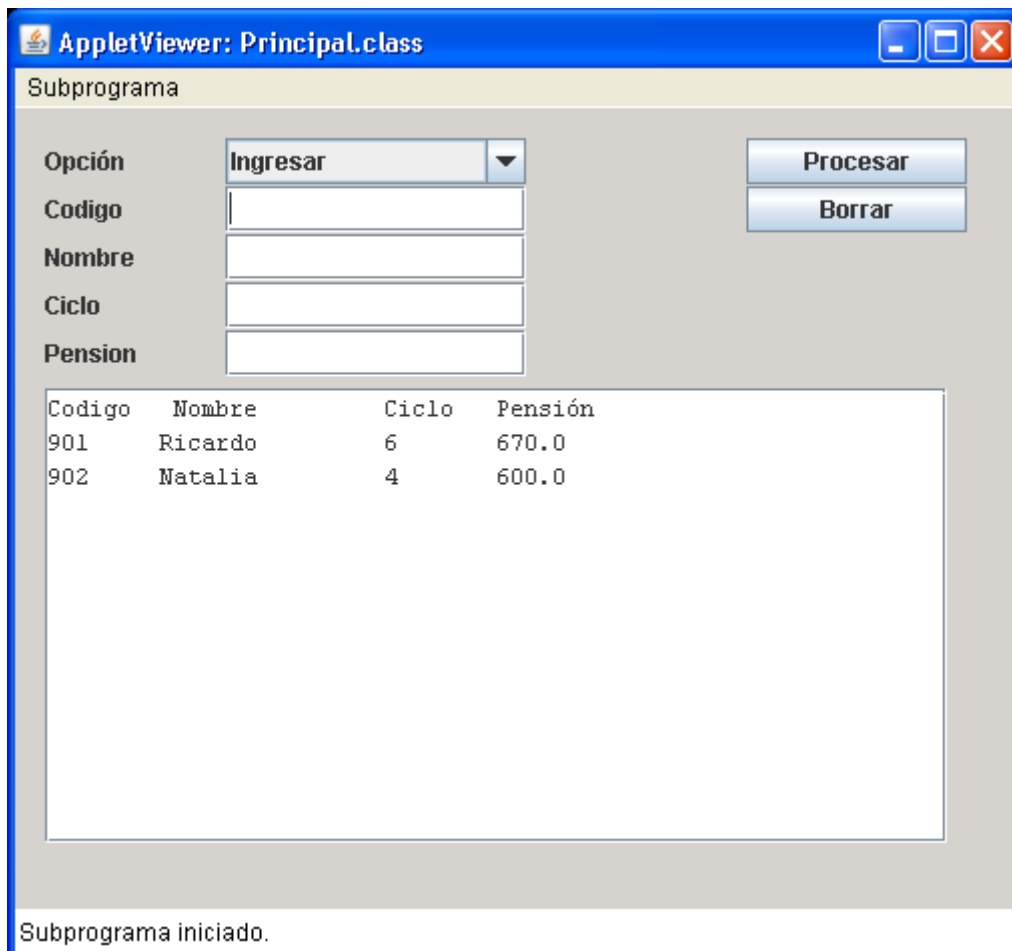
        a.adicionar(e);
        a.grabar();
        listar();
        mensaje("Estudiante Ingresado");
    }
    else
        mensaje("Codigo ya existe");
}
//-----
void modificar(){
    Estudiante e =a.buscar(getCodigo());
    if(e!=null){
        e.setNombre(getNombre());
        e.setCiclo(getCiclo());
        e.setPension(getPension());
        a.grabar();
        listar();
        mensaje("Estudiante Modificado");
    }
    else
        mensaje("Estudiante no existe");
}
//-----
void consultar(){
    imprimir();
}
```

```

        Estudiante e =a.buscar(getCodigo());
        if(e!=null){
            imprimir("Código   : "+e.getCodigo());
            imprimir("Nombre   : "+e.getNombre());
            imprimir("Ciclo    : "+e.getCiclo());
            imprimir("Pensión  : "+e.getPension());
        }
        else
            mensaje("Estudiante no existe");
    }
    //-----
    void eliminar(){
        Estudiante e =a.buscar(getCodigo());
        if(e!=null){
            a.eliminar(e);
            a.grabar();
            listar();
            mensaje("Estudiante Eliminado");
        }
        else
            mensaje("Estudiante no existe");
    }
}

```

GUI:



AppletViewer: Principal.class

Subprograma

Opción: Ingresar

Codigo:

Nombre:

Ciclo:

Pension:

Procesar

Borrar

Codigo	Nombre	Ciclo	Pensión
901	Ricardo	6	670.0
902	Natalia	4	600.0

Subprograma iniciado.

Ejercicios

- 1) Diseñe un mantenimiento de celulares que permita ingresar, consultar, modificar, eliminar y listar estudiantes. Para ello, cree las clases Celular, ArregloCelulares (en el paquete semana10) y Principal. Cree los métodos cargar y grabar en la clase ArregloCelulares. Al cargar el JApplet se deberán leer los datos del archivo celulares.txt; si el archivo no existe, deberá aparecer un mensaje de error.
Asuma la existencia de la clase Celular que cuenta con los siguientes atributos privados: código (entero), marca (cadena), modelo (cadena) y precio unitario (real). Considere que la clase cuenta con un constructor que inicializa los atributos y los métodos de acceso set y get para cada atributo.
- 2) Diseñe un mantenimiento de videos que permita ingresar, consultar, modificar, eliminar y listar videos. Para ello, cree las clases Video, ArregloVideos (en el paquete semana10) y Principal. Cree los métodos cargar y grabar en la clase ArregloVideos. Al cargar el JApplet, se deberán leer los datos del archivo videos.txt; si el archivo no existe, deberá aparecer un mensaje de error.
Asuma la existencia de la clase Video que cuenta con los siguientes atributos privados: codVideo (entero), nombre de película (cadena), codGenero (0=comedia, 1=suspenso, 2=terror) y precio unitario (real). Considere que la clase cuenta con un constructor que inicializa los atributos y los métodos de acceso set y get para cada atributo.

**UNIDAD DE
APRENDIZAJE****4****SEMANA****11**

HERENCIA Y POLIMORFISMO

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos aplican el concepto de herencia y polimorfismo, la relación de generalización/especialización entre clases, se emplean las técnicas de casting y clases abstractas en casos prácticos.

TEMARIO

- Conceptos básicos
- Uso del modificador protected
- Relación es-un

ACTIVIDADES PROPUESTAS

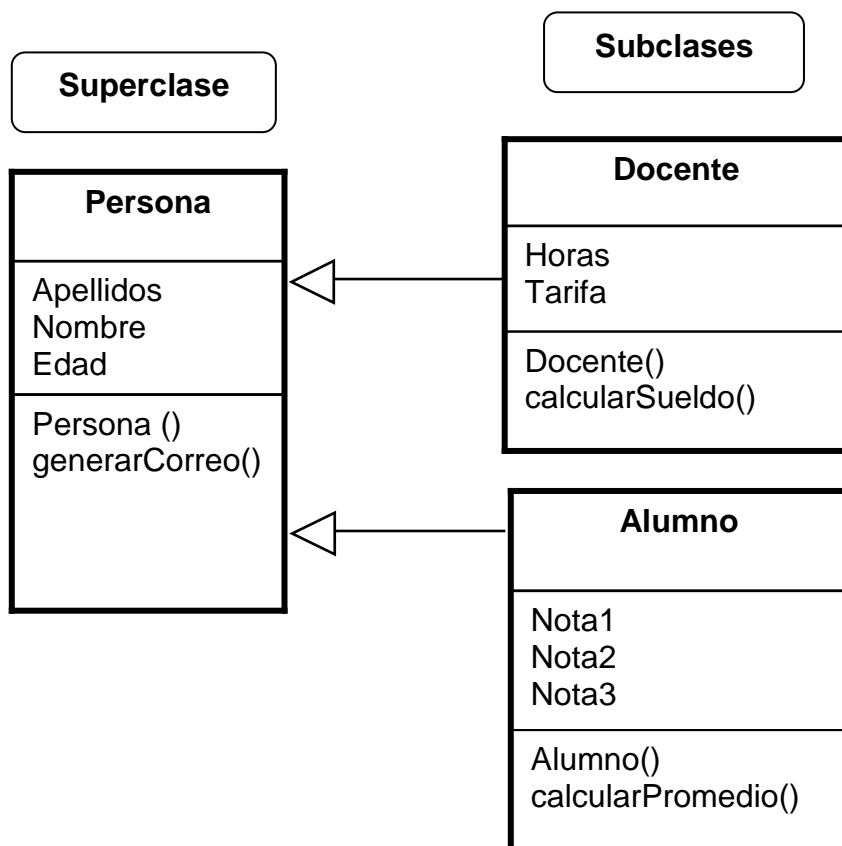
- Los alumnos aplican el concepto de Herencia (relación es-un).
- Los alumnos emplean el modificador protected.

1. RELACIÓN DE GENERALIZACIÓN / ESPECIALIZACIÓN

La relación de generalización / especialización se da cuando dos o más clases tienen muchas de sus partes en común (atributos y métodos) lo que normalmente se abstrae en la creación de una nueva clase que reúne todas sus características comunes.

La relación generalización / especialización es la relación de una clase más general y una clase más específica. La clase más específica se denomina clase hija o subclase y posee información adicional, mientras que la clase más general se denomina clase padre o superclase.

La generalización y la especialización son diferentes perspectivas del mismo concepto, la generalización es una perspectiva ascendente (*bottom-up*), mientras que la especialización es una perspectiva descendente (*top-down*).



Las subclases heredan características de las clases de las que se derivan y añaden características específicas que las diferencian. Las clases se organizan en una estructura jerárquica.

2. HERENCIA

La herencia es el mecanismo mediante el cual se puede definir una clase (subclase) sobre la base de otra clase (superclase) heredando aquellos miembros de la superclase (atributos y métodos) que hayan sido declarados como **public**, **protected** o sin especificador de acceso. Una superclase declara un miembro como **protected** para permitir el acceso al miembro desde el interior de sus subclases y desde una clase que se encuentra en el mismo paquete, a la vez que impide el acceso al miembro desde el exterior de la superclase. El constructor no es un miembro, por lo que no es heredado por las subclases.

La nueva clase puede añadir nuevos miembros e incluso puede redefinir miembros de la superclase. Redefinir un miembro de la superclase implica definir en la subclase un miembro con el mismo nombre que el de la superclase. Esto hace que el miembro de la superclase quede oculto en la subclase. A la redefinición de métodos se denomina también **sobrescritura** de métodos.

La forma general de la declaración de una clase que hereda de otra clase es la siguiente:

```
public class nombreDeSubclase extends nombreDeSuperclase {  
    // Cuerpo de la clase  
}
```

La herencia permite establecer una jerarquía de especialización mediante la relación "es-un" o "es-una".

Ejemplo 1:

Un Mamífero *es un* Animal.
Un Ave *es un* Animal.
Una Vaca *es un* Mamífero.
Un Pato *es un* Ave.

Lo que puede expresarse como:

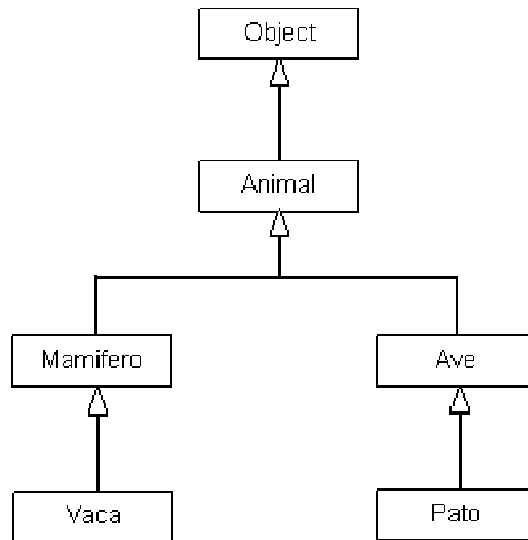
```
class Animal { ... }  
class Mamifero extends Animal { ... }  
class Vaca extends Mamifero { ... }  
class Ave extends Animal { ... }  
class Pato extends Ave { ... }
```

Si una clase no tiene una superclase explícita, implícitamente su superclase es la clase Object.

Así, en el caso de la clase Animal, implícitamente figura como:

```
class Animal extends Object { ... }
```

La clase Object define e implementa un comportamiento requerido por todas las clases dentro del Sistema Java.



Uso de super para invocar al constructor de la superclase

El constructor de la superclase puede invocarse desde la subclase utilizando la palabra **super** de la siguiente forma:

super(lista de argumentos);

Esta instrucción tiene que ser la primera sentencia a ejecutarse dentro del constructor de la subclase.

Ejemplo 2:

Define una superclase

```

package padre;
public class A {
    // Atributos
    public int v1; //se hereda
    private int v2; //no se hereda
    protected int v3; //se hereda

    // Constructor
    public A(int v1, int v2, int v3){ //no se hereda
        this.v1=v1;
        this.v2=v2;
        this.v3=v3;
    }

    // Métodos de acceso
    public int getv2(){ // se hereda
        return v2;
    }
}
  
```

```

    public int getv3(){ // se hereda
        return v3;
    }
}

```

Define una subclase

```

package hijo;

import padre.A;

public class B extends A{

    // Atributo
    public int v4;

    // Constructor
    public B(int v1, int v2, int v3, int v4){
        //super invoca al constructor de la superclase
        super(v1,v2,v3);
        this.v4=v4;
    }

    public int suma(){
        return v1+getv2()+v3+v4;
    }
}

```

Ejemplo de uso

```

import hijo.B; //hay que importar la clase B del paquete
               //hijo

// procesar es un método de la clase principal que esta
//fuera de los paquetes padre y hijo
void procesar(){
    B r = new B(1,3,5,7);
    imprimir("Objeto r de tipo B:"+r);
    imprimir("V1:"+r.v1);
    imprimir("V2:"+r.getv2());
    // utilizamos el método getv3() heredado ya que no se
    //puede emplear v3 por tener acceso protegido
    imprimir("V3:"+r.getv3());
    imprimir("V4:"+r.v4);
    imprimir("Suma:"+r.suma());
}

```

Ejemplo 3:

Implemente la **clase Persona** en el paquete padre con los siguientes miembros:

- **Atributos protegidos:** apellidos, nombres y edad
- Constructor que inicializa los atributos de la clase
- Un método generarCorreo() que retorna el correo formado por el primer caracter del nombre, el primer caracter del apellido, la edad y al final "@cibertec.edu.pe".

Luego, implemente dos **subclases de Persona: Docente y Alumno** en el paquete hijo.

Docente presenta los siguientes miembros:

- Atributos privados: horas que dicta por semana y tarifa
- Constructor con parámetros para inicializar los atributos: apellidos, nombres, edad horas y tarifa
- Método calcularSueldo() que retorna horasXtarifa

Alumno presenta los siguientes miembros:

- Atributos privados: tres notas de tipo double
- Constructor con parámetros para inicializar los atributos: apellidos, nombres, edad y las tres notas
- Método calcularPromedio() que retorna el promedio simple de las tres notas

Por último, implemente el método Procesar de la clase **Principal** que contiene el actionPerformed() para crear los objetos de Docente y Alumno e invocar a sus métodos y a los de su superclase.

Clase Persona:

```
package padre;
public class Persona {
    protected String apellido,nombre;
    protected int edad;

    public Persona(String ape, String nom, int ed) {
        apellido=ape;
        nombre=nom;
        edad=ed;
    }

    public String generarCorreo(){
        return ""+nombre.charAt(0)+apellido.charAt(0)+
            edad+"@cibertec.edu.pe";
    }
}
```

Subclase Docente:

```
package hijo;
import padre.Persona;

public class Docente extends Persona{
    private double horas,tarifa;
```

```
public Docente(String ape, String nom, int ed, double h,
               double t) {
    super(ape,nom,ed);
    horas=h;
    tarifa=t;
}

public double calcularSueldo(){
    return horas*tarifa;
}

public String mostrarDatos(){
    return "Apellido: "+apellido+"\n"+
           "Nombre: "+nombre+"\n"+
           "Edad: "+edad+"\n"+
           "Horas: "+horas+"\n"+
           "Tarifa: "+tarifa+"\n";
}
}
```

Subclase Alumno:

```
package hijo;
import padre.Persona;

public class Alumno extends Persona{

    private double nota1,nota2,nota3;

    public Alumno(String ape, String nom, int ed, double n1,
                  double n2, double n3) {
        super(ape,nom,ed);
        nota1=n1;
        nota2=n2;
        nota3=n3;
    }

    public double calcularPromedio(){
        return (nota1+nota2+nota3)/3;
    }

    public String mostrarDatos(){
        return "Apellido: "+apellido+"\n"+
               "Nombre: "+nombre+"\n"+
               "Edad: "+edad+"\n"+
               "Nota1: "+nota1+"\n"+
               "Nota2: "+nota2+"\n"+
               "Nota3: "+nota3+"\n";
    }
}
```

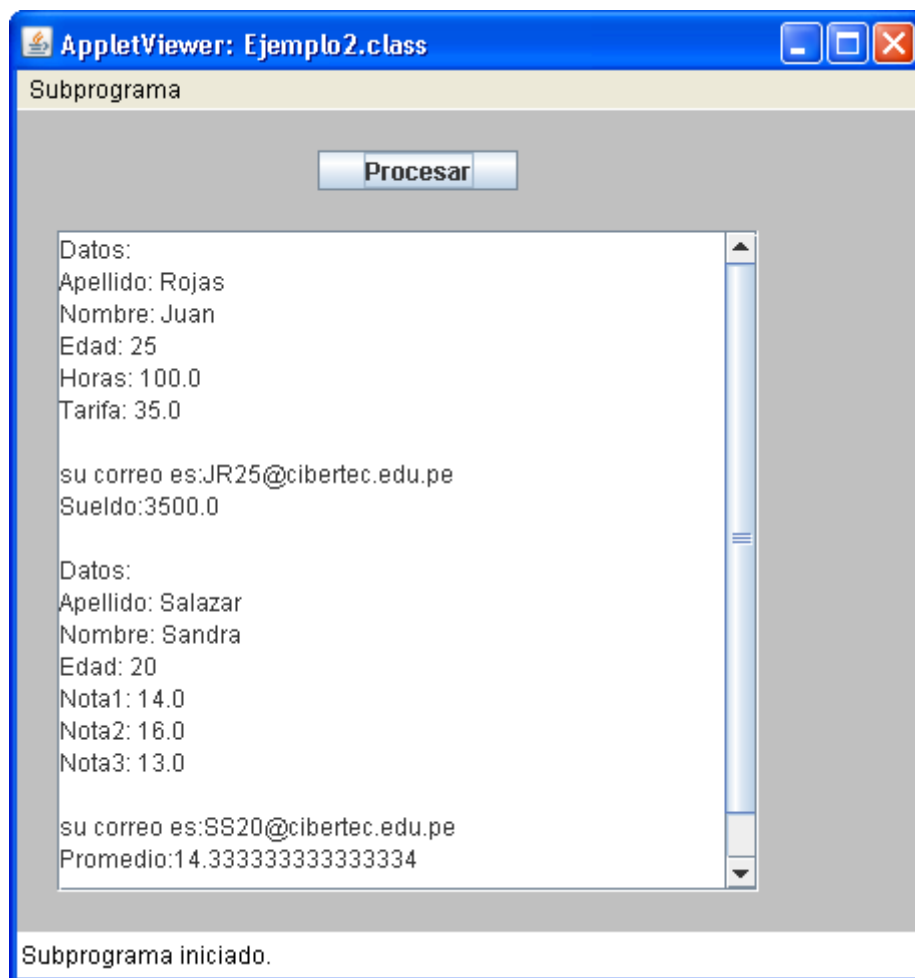
En la clase Principal:

```
import hijo.*;
...
...

void procesar(){
    Docente obj1 = new Docente("Rojas","Juan",25,100,35);
    imprimir("Datos: \n"+obj1.mostrarDatos());
    imprimir("su correo es:"+obj1.generarCorreo());
    imprimir("Sueldo:"+obj1.calcularSueldo()+"\n");

    Alumno obj2 = new Alumno("Salazar","Sandra",20,14,
                             16,13);
    imprimir("Datos: \n"+obj2.mostrarDatos());
    imprimir("su correo es:"+obj2.generarCorreo());
    imprimir("Promedio:"+obj2.calcularPromedio()+"\n");
}
```

GUI :



Ejercicios

- 1) Implemente la **clase Figura** en el paquete padre con los siguientes miembros:
 - Atributos privados: x, y que representa la ubicación de la figura geométrica
 - Constructor que inicializa los atributos de la clase.
 - Un método ubicacion() que retorna la ubicación de la figura geométrica según sus posiciones x e y.

Luego, implemente dos **subclases de Figura: Cuadrado y Círculo** en el paquete hijo.

Cuadrado presenta los siguientes miembros:

- Atributo privado: lado
- Constructor con parámetros para inicializar los atributos: x, y, lado.
- Método area() que retorna el area del cuadrado (lado*lado)

Círculo presenta los siguientes miembros:

- Atributo privado: radio
- Constructor con parámetros para inicializar los atributos: x, y, radio
- Método area() que retorna el área del círculo ($\pi \cdot \text{radio} \cdot \text{radio}$)

Por último, implemente el método Procesar de la clase **Principal** que contiene el `actionPerformed()` y cree 2 objetos: uno de tipo Cuadrado y el otro de tipo Circulo e imprima su ubicación y área de cada objeto.

- 2) Diseñe la **clase Trabajador** en el paquete padre con los siguientes miembros:
 - **Atributos protegidos:** nombre, apellido, telefono de tipo String
 - Constructor que inicializa los atributos de la clase.
 - Un método generarCodigo() que retorna el código formado por el primer carácter del nombre, el último carácter del apellido y el teléfono del trabajador.

Luego, implemente dos **subclases de Trabajador: Empleado y Consultor** en el paquete hijo.

Empleado presenta los siguientes miembros:

- Atributos privados: sueldo básico y porcentaje de bonificación
- Constructor con parámetros para inicializar los atributos: nombre, apellido, teléfono, sueldo básico y porcentaje de bonificación
- Un método boniSoles() que retorna la bonificación en soles ($\text{sbas} \cdot \% \text{ bonificación}$)
- Un método sbruto() que retorna el sueldo bruto del empleado ($\text{sbas} + \text{bonificación en soles}$)
- Un método mostrarDatos() que retorne una cadena conteniendo: nombre, apellido, telefono, bonificación en soles y el sueldo bruto

Consultor presenta los siguientes miembros:

- Atributos privados: horas trabajadas y tarifa horaria
- Constructor con parámetros para inicializar los atributos: nombre, apellido, teléfono, horas y tarifa
- Un método sbruto() que retorna el sueldo bruto del consultor (horas*tarifa)
- Un método mostrarDatos() que retorne una cadena conteniendo: nombre, apellido, telefono y el sueldo bruto

Por último, implemente el método Procesar de la clase **Principal** que contiene el `actionPerformed()` y cree 2 objetos: uno de tipo Empleado y el otro de tipo Consultor e imprima sus datos invocando al método `mostrarDatos()` y su código generado.

**UNIDAD DE
APRENDIZAJE****4****SEMANA****12-13**

HERENCIA Y POLIMORFISMO

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos aplican el concepto de herencia y polimorfismo, la relación de generalización/especialización entre clases, se emplean las técnicas de casting y clases abstractas en casos prácticos.

TEMARIO

- Sobreescritura de métodos
- Clases y métodos abstractos
- Relación tiene-un

ACTIVIDADES PROPUESTAS

- Los alumnos emplean la sobreescritura de métodos
- Los alumnos emplean clases y métodos abstractos
- Los alumnos emplean la Herencia (relación tiene-un)

1) Sobrescritura de métodos

En una jerarquía de clases, cuando un método de una subclase tiene el mismo nombre, los mismos parámetros y el mismo tipo de retorno que un método de su superclase, se dice que el método de la subclase **sobrescribe** al método de la superclase. Cuando se llama a un método sobrescrito dentro de una subclase, siempre se refiere a la versión del método definida en la subclase. La versión del método definida por la superclase queda oculta.

Uso de super para acceder a una variable oculta o a un método sobrescrito

Si desde la subclase se quiere acceder a una variable oculta de la superclase o a un método sobrescrito de la superclase, se usa la palabra **super** de la siguiente forma:

super.nombreDeLaVariableOculta

super.nombreDelMetodoSobrescrito (*lista de argumentos*)

Ejemplo 1:

```
// Superclase
package semana12;
public class A {
    //Atributos
    public int i, j;

    //Constructor
    public A (int pi, int pj) {
        i = pi;
        j = pj;
    }

    //Método
    public int suma() {
        return i + j;
    }
}

// Subclase
package semana12;
public class B extends A {
    //Atributo
    public int k;

    //Constructor
    public B (int pi, int pj, int pk) {
        super(pi,pj);
        k = pk;
    }

    //Método
    public int suma() {
        return super.suma() + k;
    }
}
```

```
}
```

// Ejemplo de uso

```
import semana12.*;

...

void procesar(){
    A obj1 = new A(2,4);
    B obj2 = new B(1,3,5);

    imprimir(obj1.i + " + " + obj1.j + " = " + obj1.suma());
    imprimir(obj2.i + " + " + obj2.j + " + " + obj2.k + " = " +
        obj2.suma());
}
```

2) Clases y métodos abstractos

Una clase abstracta es una clase que no se puede instanciar, se usa únicamente para crear subclases.

Un método es abstracto cuando no tiene implementación y solamente se define con el objetivo de obligar a que en cada subclase que deriva de la clase abstracta se realice la correspondiente implementación, es decir en las subclases se va a sobre-escribir dicho método. En Java, tanto la clase como el método abstracto se etiqueta con la palabra reservada **abstract**.

Sí se tiene en una clase por lo menos un método abstracto, entonces, la clase tiene que ser abstracta, sino el compilador mostrará un mensaje de error.

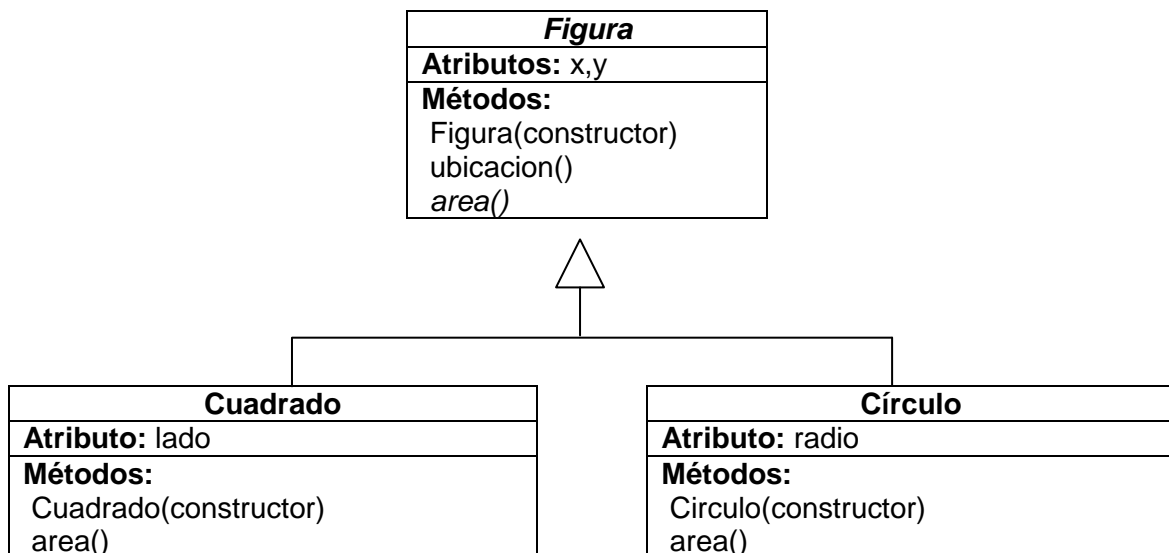


Figura es una clase abstracta porque no tiene sentido calcular su área, pero sí la de un cuadrado o un círculo. Si una subclase de **Figura** no redefine o sobre-escribe **area()**, deberá declararse también como clase abstracta.

Ejemplo 2:**Código de la clase Figura:**

```
package padre;

public abstract class Figura {
    protected int x;
    protected int y;

    public Figura(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public String ubicacion(){
        return x+", "+y;
    }

    public abstract double area();
}
```

Código de la clase Cuadrado:

```
package hijo;

import padre.Figura;

public class Cuadrado extends Figura {

    private double lado;

    public Cuadrado(int x, int y, double lado) {
        super(x, y);
        this.lado = lado;
    }

    //Sobre-escritura
    public double area() {
        return lado*lado;
    }
}
```

Código de la clase Circulo:

```
package hijo;

import padre.Figura;

public class Circulo extends Figura {

    private double radio;
```

```

    public Circulo(int x, int y, double radio) {
        super(x, y);
        this.radio=radio;
    }

    //Sobre-escritura
    public double area() {
        return Math.PI*radio*radio;
    }
}

```

Código de la clase Principal:

```

import hijo.*;
...
...

void procesar(){
    Cuadrado obj1=new Cuadrado(100,20,12);
    imprimir("Ubicación del cuadrado:"+obj1.ubicacion());
    imprimir("Area del cuadrado:"+obj1.area());

    Circulo obj2=new Circulo(300,200,20);
    imprimir("Ubicación del círculo:"+obj2.ubicacion());
    imprimir("Area del círculo:"+obj2.area());
}

```

Ejemplo 3:

Implemente la **clase abstracta Vehiculo** en el paquete padre que contenga el atributo capacidad del vehículo y un método abstracto `caracteristicas()` que se sobrescribirá en las subclases Carro y Avión. Por último, cree la clase **Principal** que contiene el `actionPerformed()` para crear los objetos de tipo Carro y Avión e imprima las características de dichos objetos.

```

// Superclase
package padre;

public abstract class Vehiculo {
    protected int capacidad;

    public Vehiculo(int capacidad) {
        this.capacidad=capacidad;
    }

    public abstract String caracteristicas();
}

```

```
// Subclase
package hijo;
import padre.Vehiculo;

public class Carro extends Vehiculo{

    public Carro(int cap) {
        super(cap);
    }

    // Sobre-escritura
    public String características(){
        return "Tengo una capacidad de "+capacidad+ "
                pasajeros \n"+
                "Tengo 4 ruedas y una buena velocidad \n"+
                "pero no puedo volar";
    }
}

// Subclase
package hijo;
import padre.Vehiculo;

public class Avion extends Vehiculo{

    public Avion(int cap) {
        super(cap);
    }

    // Sobre-escritura
    public String características(){
        return "Tengo una capacidad de "+capacidad+ "
                pasajeros \n"+
                "Puedo volar, tengo una velocidad enorme \n"+
                "pero consumo mucho combustible";
    }
}

// Ejemplo de uso – Clase Principal

import hijo.*;
...
...

void procesar(){
    Carro obj1 = new Carro(6);
    imprimir("Objeto Carro:");
    imprimir(obj1.características());

    Avion obj2 = new Avion(300);
    imprimir("\nObjeto Avión:");
    imprimir(obj2.características());
}
```

3) Relación tiene-un

La relación tiene-un consiste en que la clase A tenga una referencia a un objeto de la clase B por ejemplo:

```
package semanal2;

public class A {
    //Atributo privado
    private B obj;
}
```

De tal forma que, a través de la referencia **obj**, se puede acceder a los miembros de la clase B.

Ejemplo 4:

Diseñe un programa que, a la pulsación del botón Procesar, permita abrir la cuenta de un cliente. Para ello, considere que la cuenta tiene los siguientes datos: número de la cuenta, saldo de la cuenta, fecha de apertura (fecha del sistema en el formato: dd/mm/aaaa) y titular de la cuenta.

El titular de la cuenta es un cliente que tiene los siguientes datos: código de cliente, nombre, apellidos y teléfono.

Código de la clase Cuenta:

```
package semanal2;

public class Cuenta {
    //Atributos privados
    private int nro;
    private Cliente titular;
    private double saldo;
    private String fecApertura;
    private Fecha fec;

    //Constructor
    public Cuenta(int nro, Cliente titular, double saldo) {
        this.nro = nro;
        this.titular = titular;
        this.saldo = saldo;
        fec = new Fecha();
        fecApertura = fec.getFecha();
    }

    //Métodos de acceso: get
    public int getNro(){
        return nro;
    }

    public String getDatosTitular(){
        return "Codigo Cliente:"+titular.getCodigo()+"\n"+
            "Nombre Cliente:"+titular.getNombre()+"\n"+
            "Apellidos Cliente:"+ titular.getApellidos()+
```

```
        "\n"+ "Telefono Cliente:"+
        titular.getTelefono();
    }

    public double getSaldo(){
        return saldo;
    }

    public String getFechaApertura(){
        return fecApertura;
    }
}
```

Código de la clase Cliente:

```
package semanal2;

public class Cliente {
    //Atributos privados
    private int cod;
    private String nom,ape,telf;

    //Constructor
    public Cliente(int cod, String nom, String ape, String
                    telf) {
        this.cod=cod;
        this.nom=nom;
        this.ape=ape;
        this.telf=telf;
    }

    //Métodos de acceso: get
    public int getCodigo(){
        return cod;
    }

    public String getNombre(){
        return nom;
    }

    public String getApellidos(){
        return ape;
    }

    public String getTelefono(){
        return telf;
    }
}
```

Código de la clase Fecha:

```
package semanal2;

import java.util.*;
```



```

public class Fecha {
    //Atributos privados
    private int dd,mm,aa;
    private GregorianCalendar gc;

    //Constructor
    public Fecha() {
        gc = new GregorianCalendar();
        dd = gc.get(Calendar.DAY_OF_MONTH);
        mm = gc.get(Calendar.MONTH);
        aa = gc.get(Calendar.YEAR);
    }

    //Método
    public String getFecha(){
        return dd+"/"+(mm+1)+"/"+aa;
    }
}

```

Código de la clase Principal:

```

import semanal2.*;
...
...

void procesar(){
    Cliente cli = new Cliente(100,"Rodolfo","Garcia","4451234");
    Cuenta cta = new Cuenta(925671,cli,3500);
    listar(cta);
}

void listar(Cuenta x){
    imprimir("Codigo de cuenta:"+x.getNro());
    imprimir("Titular de la cuenta:");
    imprimir(x.getDatosTitular());
    imprimir("Saldo de la cuenta:"+x.getSaldo());
    imprimir("Fecha de apertura de la cuenta:" +
    x.getFechaApertura());
}

```

Importante:

- ✓ La clase **Cuenta** tiene un objeto de tipo **Cliente** por lo que podemos acceder a los métodos de la clase Cliente con el método `getDatosTitular()`.
- ✓ La clase **Cuenta** tiene un objeto de tipo **Fecha** por lo que podemos acceder al método `getFecha()` de la clase Fecha cuando hacemos `fec.getFecha()`.
- ✓ La clase **Fecha** tiene un objeto de tipo **GregorianCalendar** por lo que podemos obtener el día, mes y año del sistema.

Ejemplo 5:

Dada la clase Cuadrado que permite calcular el área de un cuadrado, se le pide diseñar una clase CuadradoRelleno que herede de la clase Cuadrado y que, además de permitir obtener el área del cuadrado, permita dibujar el cuadrado con un caracter de relleno especificado.

Use, luego, la clase CuadradoRelleno para implementar el programa DibujoCuadrado que lea el lado y el caracter de relleno, y dibuje y muestre el área del cuadrado.

EJEMPLO:

Datos ingresados:

Lado : 4
Caracter : &

Salida de resultados:

&&&&
&&&&
&&&&
&&&&

Area : 16

```
public class Cuadrado {  
  
    // Atributo  
    private int lado;  
  
    // Constructor  
    public Cuadrado( int lado ){  
        this.lado = lado;  
    }  
  
    // Fija el lado  
    public void setLado( int lado ){  
        this.lado = lado;  
    }  
  
    // Retorna el lado  
    public int getLado(){  
        return lado;  
    }  
  
    // Retorna el área  
    public int area(){  
        return lado*lado;  
    }  
}
```

```

public class CuadradoRelleno extends Cuadrado{
    private char car;
    private DibujoCuadrado gui;

    public CuadradoRelleno(int lado, char car, DibujoCuadrado
                                gui){
        super(lado);
        this.car = car;
        this.gui = gui;
    }

    public void dibujar(){
        for(int f=1; f<=getLado(); f++){
            for(int c=1; c<=getLado(); c++){
                gui.txtS.append(""+car);
                gui.txtS.append("\n");
            }
        }
    }
}

```

Programa DibujoCuadrado:

Tenemos los métodos para obtener los datos de la GUI

```

//-----
int getLado(){
    return Integer.parseInt(txtLado.getText());
}
//-----
char getCaracter(){
    return txtCaracter.getText().charAt(0);
}

```

En el método Procesar se crea el objeto de tipo CuadradoRelleno y se llama a los métodos dibujar() y area()

```

void procesar(){
    int area;
    CuadradoRelleno c = new CuadradoRelleno
        (getLado(),getCaracter(),this);
    c.dibujar();
    area = c.area();
    txtS.append("\nEl Area es:"+area);
}

```

Ejercicios

1. Implemente la **clase abstracta Animal** en el paquete padre que contenga el método abstracto `habla()` que se sobreescribirá en las subclases `Perro`, `Gato` y `Pollo` (las subclases se crearán en el paquete hijo).
Por último, cree la clase **Principal** que contiene el `actionPerformed()` para crear los objetos `Perro`, `Gato` y `Pollo` e imprima como habla cada `Animal`.
2. Diseñe un programa que, a la pulsación del botón `Procesar`, registre a una persona en la `Reniec`. Para ello, se debe crear un objeto de tipo `persona` con los siguientes atributos: `apellido paterno`, `apellido materno`, `nombres`, `fecha de nacimiento`, `sexo`, `estado civil`; y los siguientes datos del `DNI`: `Nro de dni`, `fecha de inscripción`, `fecha de emisión` y `fecha de caducidad`.

**UNIDAD DE
APRENDIZAJE****4****SEMANA****14**

HERENCIA Y POLIMORFISMO

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos aplican el concepto de herencia y polimorfismo, la relación de generalización/especialización entre clases, emplea las técnicas de casting y clases abstractas en casos prácticos.

TEMARIO

- Técnicas de casting: Upcasting y Downcasting
- Polimorfismo y enlace dinámico
- Operador instanceof

ACTIVIDADES PROPUESTAS

- Los alumnos emplean el polimorfismo haciendo uso de las técnicas de casting, del operador instanceof y de clases abstractas.

1) Técnicas de Casting: Upcasting y Downcasting

Las técnicas de casting consiste en realizar **conversiones de tipo, no modifican al objeto**, sólo su tipo. Únicamente tienen sentido por la existencia de la herencia.

- Upcasting : Permite interpretar un objeto de una clase derivada como del mismo tipo que la clase base. También se puede ver como la conversión de un tipo en otro superior en la jerarquía de clases. No hace falta especificarlo
- Downcasting: Permite interpretar un objeto de una clase base como del mismo tipo que su clase derivada. También se puede ver como la conversión de un tipo en otro inferior en la jerarquía de clases. Se especifica precediendo al objeto a convertir con el nuevo tipo entre paréntesis.

Ejemplo1:

Clase Animal

```
package semanal4;
public class Animal {

    public String hacerRuido(){
        return "no definido";
    }
}
```

Clase Mamifero

```
package semanal4;
public class Mamifero extends Animal{

    public String mensaje(){
        return "soy mamífero";
    }

}
```

Clase Perro

```
package semanal4;
public class Perro extends Mamifero{

    public String mensaje(){
        return "Soy perro";
    }

    public String hacerRuido(){
        return "Guau";
    }

}
```

Clase Principal

Aplicando Upcasting:

```
void procesar(){
    Mamifero m1 = new Perro();
    imprimir(m1.mensaje());
    imprimir(m1.hacerRuido());
}
```

Se está creando un objeto de tipo Perro. Pero, m1 es de tipo Mamífero (clase superior en la jerarquía de clases del ejemplo). Los métodos mensaje() y hacerRuido() deben existir en la clase Mamifero o se deben heredar de una clase superior (clase Animal) para que compile. Como el objeto es un Perro y los métodos mensaje() y hacerRuido() se están sobre-escribiendo en la clase Perro, la salida es:

Soy perro
Guau

Pero, si en la clase Perro no se hubiera sobre-escrito el método mensaje(), la salida hubiera sido:

soy mamífero
Guau

Y si no se hubiera sobre-escrito ni el método mensaje() ni el método hacerRuido(), la salida hubiera sido:

soy mamífero
no definido

Al ejecutar un método, se busca su implementación de abajo hacia arriba en la jerarquía de clases. Los casting no modifican al objeto. Sólo su tipo, por lo que se siguen ejecutando sobre el mismo objeto.

Aplicando Downcasting:

```
void procesar(){
    Animal a = new Perro();
    Perro p = (Perro)a;
    imprimir(p.hacerRuido());
}
```

Se está creando un objeto de tipo Perro que se almacena en la variable de referencia a de tipo Animal (clase superior en la jerarquía de clases del ejemplo). Luego, la referencia a se convierte a tipo Perro y a través de p se accesa al objeto. Se entenderá su aplicación con el ejemplo completo de polimorfismo que se verá en las siguientes páginas.

El siguiente ejemplo, compila pero no funciona porque a1 hace referencia a un objeto de tipo Animal. No se puede convertir un animal a perro.

```
void procesar(){
    Animal a1 = new Animal();
    Perro p1 = (Perro)a1; //ERROR
    imprimir(p1.hacerRuido());
}
```

2) Polimorfismo y enlace dinámico

El polimorfismo está presente cuando se realiza la llamada a un método de un objeto del que no se sabe su tipo hasta que el programa esté en ejecución. Al tener métodos sobre-escritos, objetos de diferentes tipos pueden responder de forma diferente a la misma llamada, de tal forma que podemos escribir código de forma general sin preocuparnos del método concreto que se ejecutará en cada momento.

El enlace dinámico se da cuando se elige el método a ejecutar en tiempo de ejecución, en función de la clase del objeto; es la implementación del polimorfismo.

Un método es polifórmico cuando actúa de diferentes formas dependiendo del objeto que reciba. En el ejemplo que se muestra a continuación, el método listar de la clase Principal es un método polifórmico, ya que, cuando es llamado por primera vez, lista el comportamiento de un Gato y, la segunda vez, lista el comportamiento de un Perro.

Ejemplo 2:

Clase Mamífero

```
package padre;

public abstract class Mamifero {

    public abstract String hacerRuido();

    public String mensaje(){
        return "soy mamífero";
    }

}
```

Clase Perro

```
package hijo;

import padre.Mamifero;

public class Perro extends Mamifero{

    public String hacerRuido(){
        return "Guau";
    }

}
```


Clase Gato

```
package hijo;

import padre.Mamifero;

public class Gato extends Mamifero{

    public String hacerRuido(){
        return "Miau";
    }

    public String mensaje(){
        return "soy gato";
    }

}
```

Clase Principal

```
import padre.Mamifero;
import hijo.*;
...
...

void procesar(){
    Gato g = new Gato();
    listar(g);

    Perro p = new Perro();
    listar(p);
}

// listar es un método polifórmico
void listar( Mamifero x ){
    imprimir(x.hacerRuido());
    imprimir(x.mensaje());
}
```

Salida:

```
Miau
soy gato
Guau
soy mamífero
```

Importante: El polimorfismo se da en el método listar, ya que la variable de referencia x va a recibir un objeto de diferente tipo cada vez que se llame al método listar y la ejecución de los métodos hacerRuido() y mensaje() se van a comportar dependiendo del objeto al cual está referenciando x.

3) Operador instanceof

Se utiliza el operador instanceof para determinar si el objeto es de la clase esperada antes de realizar el casting.

Ejemplo 3:

Implemente la **clase abstracta Persona** en el paquete semana14 con los siguientes miembros:

- **Atributos protegidos:** apellidos, nombres y edad
- Constructor que inicializa los atributos de la clase
- Un **método abstracto:** mostrarDatos() para mostrar los datos de la persona creada
- Un método generarCorreo() que retorna el correo formado por el primer carácter del nombre, el primer carácter del apellido, la edad y al final "@cibertec.edu.pe".

Luego, implemente dos **subclases de Persona: Docente y Alumno** en el paquete semana14.

Docente presenta los siguientes miembros:

- Atributos privados: horas que dicta por semana y tarifa
- Constructor con parámetros para inicializar los atributos: apellidos, nombres, edad horas y tarifa
- **Redefina** o sobrescriba el método mostrarDatos()
- Método calcularSueldo() que retorna horasXtarifa

Alumno presenta los siguientes miembros:

- Atributos privados: tres notas de tipo double.
- Constructor con parámetros para inicializar los atributos: apellidos, nombres, edad y las tres notas
- **Redefina** o sobrescriba el método mostrarDatos()
- Método calcularPromedio() que retorna el promedio simple de las tres notas.

Por último, implemente el método Procesar de la clase **Principal** que contiene el actionPerformed() para crear los objetos de Docente y Alumno e invoque al método listar para mostrar los datos de los objetos creados.

Clase Persona

```
package semana14;

public abstract class Persona {
    protected String apellido,nombre;
    protected int edad;

    public Persona(String ape, String nom, int ed) {
        apellido=ape;
        nombre=nom;
        edad=ed;
    }

    public abstract String mostrarDatos();
}
```

```
public String generarCorreo(){
    return "+"nombre.charAt(0)+apellido.charAt(0)+edad+
        "@cibertec.edu.pe";
}
}
```

Clase Docente

```
package semana14;

public class Docente extends Persona{
    private double horas,tarifa;

    public Docente(String ape, String nom, int ed, double h,
        double t) {
        super(ape,nom,ed);
        horas=h;
        tarifa=t;
    }

    public double calcularSueldo(){
        return horas*tarifa;
    }

    public String mostrarDatos(){
        return "Apellido: "+apellido+"\n"+
            "Nombre: "+nombre+"\n"+
            "Edad: "+edad+"\n"+
            "Horas: "+horas+"\n"+
            "Tarifa: "+tarifa+"\n";
    }
}
```

Clase Alumno

```
package semana14;

public class Alumno extends Persona{

    private double nota1,nota2,nota3;

    public Alumno(String ape, String nom, int ed, double n1,
        double n2, double n3) {
        super(ape,nom,ed);
        nota1=n1;
        nota2=n2;
        nota3=n3;
    }

    public double calcularPromedio(){
        return (nota1+nota2+nota3)/3;
    }
}
```

```

        public String mostrarDatos(){
            return "Apellido: "+apellido+"\n"+
                "Nombre: "+nombre+"\n"+
                "Edad: "+edad+"\n"+
                "Nota1: "+nota1+"\n"+
                "Nota2: "+nota2+"\n"+
                "Nota3: "+nota3+"\n";
        }
    }
}

```

Clase Principal

```

import semanal4.*;

...

void procesar(){
    Docente obj1=new Docente("Gálvez","Ricardo",35,100,40);
    listar(obj1);

    Alumno obj2=new Alumno("Salazar","Sandra",20,14,16,13);
    listar(obj2);
}

// listar es un método polifórmico
void listar(Persona x){
    imprimir("Datos: \n"+x.mostrarDatos());
    imprimir("su correo es:"+x.generarCorreo());
    if(x instanceof Docente)
        imprimir("Sueldo:"+((Docente)x).calcularSueldo()+"\n");
    else
        imprimir("Promedio:"+((Alumno)x).calcularPromedio()+"\n");
}

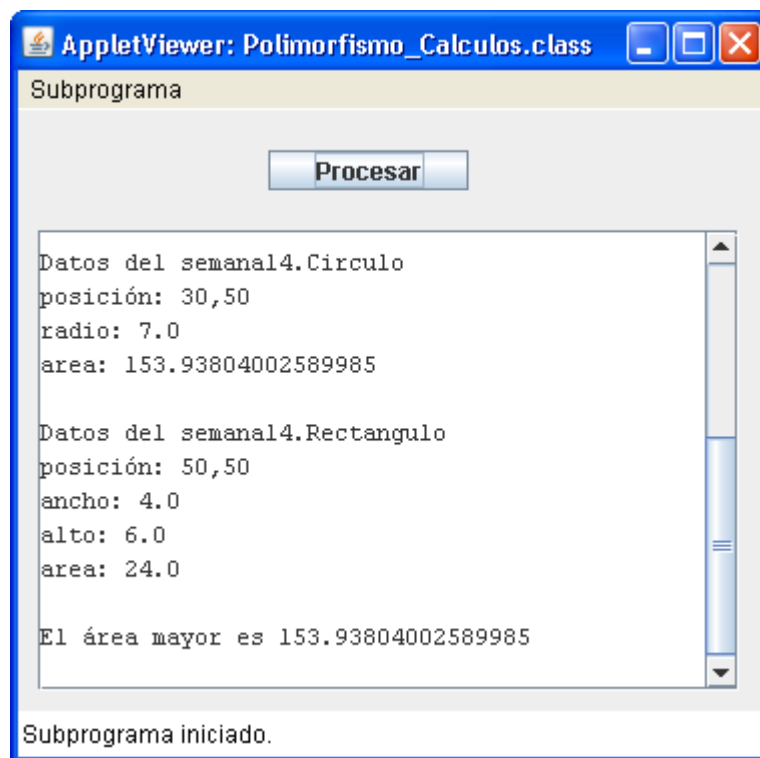
```

Ejemplo 4:

Diseñe un programa que, al pulsar el botón Procesar, cree un arreglo con 4 objetos con datos fijos, 2 objetos de tipo Rectángulo y 2 de tipo Círculo. El programa debe mostrar en el JTextArea lo siguiente:

- Si es un objeto de tipo Rectángulo, debe mostrarse posición x, posición y, ancho, alto y el área del objeto.
- Si es un objeto de tipo Círculo, debe mostrarse posición x, posición y, radio y el área del objeto.

Adicionalmente, debe calcular e imprimir el área mayor de los objetos creados.

GUI del problema:**Clase Figura**

```
package semanal4;
public abstract class Figura {
    protected int x;
    protected int y;

    public Figura(int x, int y) {
        this.x=x;
        this.y=y;
    }

    public abstract double area();

    public abstract String mostrarDatos();
}
```

Clase Rectángulo

```
package semanal4;
public class Rectangulo extends Figura{
    private double ancho, alto;

    public Rectangulo(int x, int y, double ancho, double alto){
        super(x,y);
        this.ancho=ancho;
        this.alto=alto;
    }
}
```

```
    public double area(){
        return ancho*alto;
    }

    public String mostrarDatos(){
        return "posición: "+x+", "+y+"\n"+
            "ancho: "+ancho+"\n"+
            "alto: "+alto+"\n"+
            "area: "+area()+"\n";
    }
}
```

Clase Círculo

```
package semana14;
public class Circulo extends Figura{
    private double radio;

    public Circulo(int x, int y, double radio){
        super(x,y);
        this.radio=radio;
    }

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

    public String mostrarDatos(){
        return "posición: "+x+", "+y+"\n"+
            "radio: "+radio+"\n"+
            "area: "+area()+"\n";
    }
}
```

Clase Principal

```
import semana14.*;
...
...

Figura figuraMayor(Figura[] figuras){
    Figura mFigura=null;
    double areaMayor=0;

    for(int i=0; i<figuras.length; i++){
        if(figuras[i].area()>areaMayor){
            areaMayor=figuras[i].area();
            mFigura=figuras[i];
        }
    }
    return mFigura;
}
```

```
void procesar(){
    Figura[] fig = new Figura[4];

    fig[0] = new Rectangulo(0, 0, 5.0, 7.0);
    fig[1] = new Circulo(10, 20, 5.0);
    fig[2] = new Circulo(30, 50, 7.0);
    fig[3] = new Rectangulo(50, 50, 4.0, 6.0);

    listar(fig);
    Figura fMayor = figuraMayor(fig);
    imprimir("El área mayor es " + fMayor.area());
}

void listar(Figura[] x){
    for(int i=0; i<x.length; i++){
        imprimir("Datos del "+x[i].getClass().getName());
        imprimir(x[i].mostrarDatos());
    }
}
```

Importante:

- fig es un arreglo de objetos de tipo Figura.
- Cuando hacemos **listar(fig)**, estamos enviando el arreglo como parámetro; por esa razón, el método listar debe recibir el arreglo de la forma **void listar(Figura[] x)**
- **getClass()** es un método de la clase Object y se utiliza para obtener el nombre de la clase de un objeto.

Ejercicios

Implemente los ejercicios de la página 108 aplicando clases y métodos abstractos, polimorfismo y el operador `instanceof`.

**UNIDAD DE
APRENDIZAJE****5****SEMANA****15-16**

INTERFACES

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos aplican el concepto de herencia múltiple. Crean e implementan interfaces.

TEMARIO

- Creación e implementación de Interfaces
- Herencia múltiple

ACTIVIDADES PROPUESTAS

- Los alumnos crean e implementan interfaces
- Los alumnos aplican el concepto de herencia múltiple

1) Creación de interfaces

Una interfaz es una clase completamente abstracta, es decir es una clase sin implementación.

Una interfaz es cuando lo único que puede tener son declaraciones de métodos y definiciones de constantes simbólicas. En Java, las interfaces se declaran con la palabra reservada **interface**.

La clase que implementa una o más interfaces utiliza la palabra reservada **implements**. Para ello, es necesario que la clase implemente todos los métodos definidos por la interfaz.

Una interfaz podrá verse, simplemente, como una forma, es como un molde; solamente permite declarar nombres de métodos. En este caso, no es necesario definirlos como abstractos, puesto que lo son implícitamente. Y si adicionalmente tiene miembros datos, éstos serán constantes, es decir, `static` y `final`.

Al utilizar **implements** para el interfaz es como si se hiciese una acción de *copiar y pegar* del código del interfaz, con lo cual no se hereda nada, solamente se pueden usar los métodos.

La ventaja principal del uso de interfaces es que puede ser implementada por cualquier número de clases, permitiendo a cada clase compartir el interfaz de programación sin tener que ser consciente de la implementación que hagan las otras clases que implementen el **interfaz**.

La principal **diferencia** entre **interface** y **clase abstracta** es que una interface proporciona un mecanismo de encapsulamiento sin forzar al usuario a utilizar la herencia.

Ejemplo 1

Código de la interfaz: Constantes

```
package interfaces;
public interface Constantes {

    double pi = 3.14;
    int constanteInt = 5;
}
```

Código de la interfaz: Interfaz1

```
package interfaces;
public interface Interfaz1 {
    void put( int dato );
    int get();
}
```

Código de la clase: ClaseA

```
package clases;
import interfaces.*;
```

```
public class ClaseA implements Constantes, Interfaz1 {
    double dato;

    // El método put del interfaz1
    public void put( int dato ) {
        // Se usa "pi" del interfaz Constantes
        this.dato = dato * pi;
    }

    // El método get del interfaz1
    public int get() {
        return( (int)dato );
    }

    // El método show() no esta declarado en el interfaz1
    public String show() {
        return "imprime " + dato;
    }
}
```

Código de la clase: ClaseB

```
package clases;
import interfaces.*;

public class ClaseB implements Constantes, Interfaz1 {
    int dato;

    // El método put del interfaz1
    public void put( int dato ) {
        // Se usa "constanteInt" del interfaz Constantes
        this.dato = dato * constanteInt;
    }

    // El método get del interfaz1
    public int get() {
        return dato;
    }
}
```

Código de la clase: Principal

```
import interfaces.*;
import clases.*;
...
...

void procesar(){
    // Se crea un objeto de tipo ClaseA
    ClaseA objA = new ClaseA();
    objA.put(2);
    // El método show pertenece a la claseA
    imprimir(objA.show());
    imprimir("dato objA = "+objA.get());
}
```

```

// Se crea un objeto de tipo ClaseB
ClaseB objB = new ClaseB();
objB.put(4);

imprimir("dato objB = " + objB.get());

}

```

Salida:

```

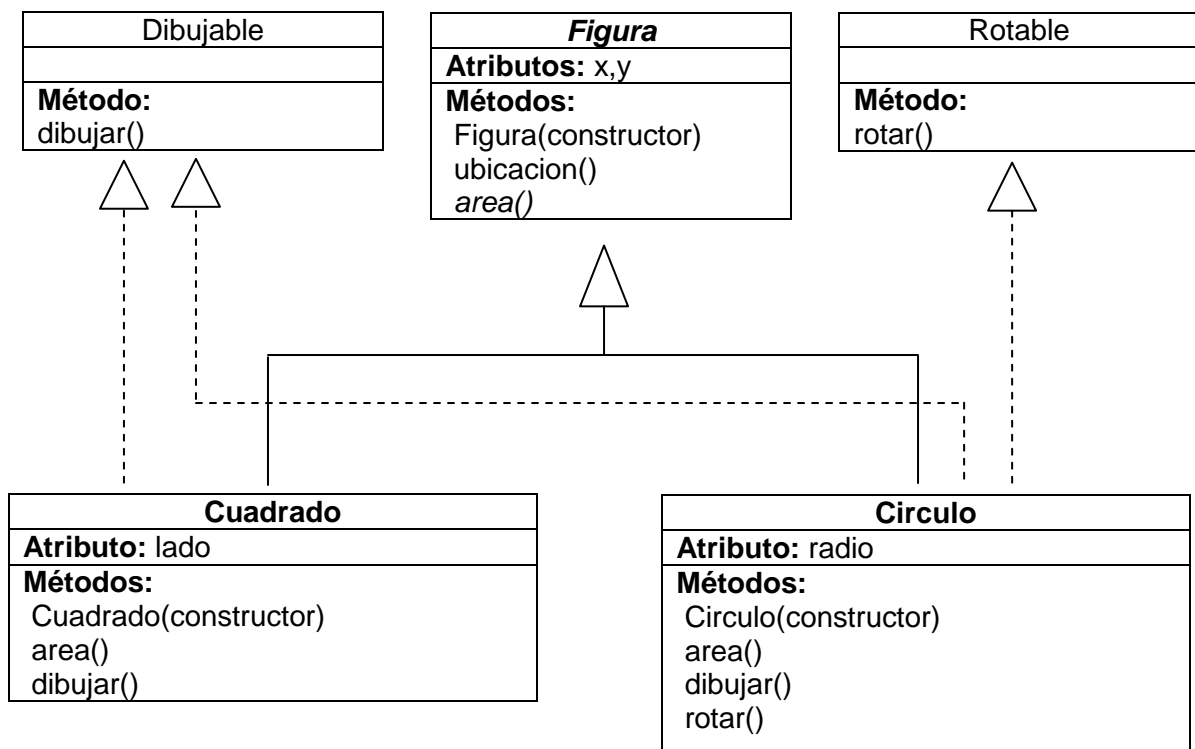
imprime 6.28
dato objA = 6
dato objB = 20

```

2) Herencia múltiple

En Java, realmente, no existe la herencia múltiple. Lo que se puede es crear una clase que implemente (implements) más de un interfaz, pero sólo puede extender una clase (extends).

La siguiente figura muestra la implementación de interfaces:



Dibujable y **Rotable** son interfaces, cuyos métodos serán implementados en otras clases. Por ejemplo: **Cuadrado** implementa la interfaz **Dibujable** y **Círculo** implementa las interfaces **Dibujable** y **Rotable**.

Ejemplo 2

```

package interfaces;
public interface Dibujable {
    String dibujar();
}

```

```
package interfaces;
public interface Rotable {
    String rotar();
}

package clases;
public abstract class Figura{

    protected int x;
    protected int y;

    public Figura(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public String ubicacion(){
        return "figura ubicada en: "+x+", "+y;
    }

    public abstract double area() ;

}

package clases;
import interfaces.Dibujable;

public class Cuadrado extends Figura implements Dibujable{

    private double lado;

    public Cuadrado(int x, int y, double lado) {
        super(x, y);
        this.lado = lado;
    }

    //Sobre-escritura
    public double area() {
        return lado*lado;
    }

    //Sobre-escritura
    public String dibujar() {
        return "El cuadrado puede ser dibujado";
    }

}

package clases;
import interfaces.*;

public class Circulo extends Figura implements Dibujable,
Rotable {

    private double radio;
```

```

    public Circulo(int x, int y, double radio) {
        super(x, y);
        this.radio = radio;
    }

    //Sobre-escritura
    public double area() {
        return Math.PI * radio * radio;
    }

    //Sobre-escritura
    public String dibujar() {
        return "El círculo puede ser dibujado";
    }

    //Sobre-escritura
    public String rotar() {
        return "El círculo puede rotar";
    }
}

```

Clase Principal

```

import clases.*;
...
...

void procesar(){
    Cuadrado obj1 = new Cuadrado(10,20,8);
    listar(obj1);

    Circulo obj2 = new Circulo(40,110,3);
    listar(obj2);
}

void listar(Figura x){
    imprimir(x.ubicacion());
    imprimir("El área es:"+x.area());
    if(x instanceof Cuadrado)
        imprimir(((Cuadrado)x).dibujar());
    else{
        imprimir(((Circulo)x).dibujar());
        imprimir(((Circulo)x).rotar());
    }
    imprimir("");
}
}

```

Aplicación 1:

Diseñe una aplicación que muestre las operaciones de avanzar, detener, retroceder, subir y bajar que puede hacer los siguientes medios de transporte: Auto, Bicicleta, Moto, Avión y Helicóptero. Considere que hay vehículos que son terrestres y otros que son aereos. Por lo tanto, hay operaciones que, por ejemplo, un avión puede hacer, pero no un auto. En la solución, emplee interfaces, clases abstractas y subclasses.

```
package interfaces;
public interface Movimiento{

    String avanzar();
    String detener();
    String retroceder();

}

package interfaces;
public interface Volar{

    String subir();
    String bajar();

}

package clases;
import interfaces.Movimiento;
public abstract class Transporte implements Movimiento{
    protected int capacidad;

    public Transporte(int capacidad){
        this.capacidad=capacidad;
    }

    public abstract String mostrarCapacidad();

    public String avanzar(){
        return "no hay mensaje";
    }

    public String detener(){
        return "no hay mensaje";
    }

    public String retroceder(){
        return "no hay mensaje";
    }
}
```

```
package clases;
public class Auto extends Transporte{

    public Auto(int capacidad){
        super(capacidad);
    }

    public String mostrarCapacidad(){
        return "Capacidad de pasajeros del Auto:"+capacidad;
    }

    public String avanzar(){
        return "el auto esta avanzando";
    }

}
```

```
package clases;
public class Bicicleta extends Transporte{

    public Bicicleta(int capacidad){
        super(capacidad);
    }

    public String mostrarCapacidad(){
        return "Capacidad de pasajeros de la Bicicleta:"+capacidad;
    }

    public String avanzar(){
        return "la bicicleta esta avanzando";
    }

    public String detener(){
        return "la bicicleta se detuvo";
    }

}
```

```
package clases;
public class Moto extends Transporte{

    public Moto(int capacidad){
        super(capacidad);
    }

    public String mostrarCapacidad(){
        return "Capacidad de pasajeros de la Moto:"+capacidad;
    }

    public String avanzar(){
        return "la moto esta avanzando";
    }

}
```



```
    }

    public String detener(){
        return "la moto se detuvo";
    }

    public String retroceder(){
        return "la moto esta retrocediendo";
    }
}

package clases;
import interfaces.Volar;
public class Avion extends Transporte implements Volar{

    public Avion(int capacidad){
        super(capacidad);
    }

    public String mostrarCapacidad(){
        return "Capacidad de pasajeros del Avión:"+capacidad;
    }

    public String avanzar(){
        return "el avión esta avanzando";
    }

    public String detener(){
        return "el avión se detuvo";
    }

    public String retroceder(){
        return "el avión esta retrocediendo";
    }

    public String subir(){
        return "el avión esta subiendo";
    }

    public String bajar(){
        return "el avión esta bajando";
    }
}

package clases;
import interfaces.Volar;
public class Helicoptero extends Transporte implements Volar{

    public Helicoptero(int capacidad){
        super(capacidad);
    }
}
```

```
public String mostrarCapacidad(){
    return "Capacidad de pasajeros del Helicoptero:"+capacidad;
}

public String avanzar(){
    return "el helicoptero esta avanzando";
}

public String detener(){
    return "el helicoptero se detuvo";
}

public String retroceder(){
    return "el helicoptero esta retrocediendo";
}

public String subir(){
    return "el helicoptero esta subiendo";
}

public String bajar(){
    return "el helicoptero esta bajando";
}
}
```

Clase Principal:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import clases.*;

public class Principal extends JApplet implements
ActionListener{

    JButton btnProcesar;
    JTextArea txtS;
    JScrollPane scpScroll;

    // Crea la interfaz gráfica de usuario GUI
    public void init() {
        getContentPane().setLayout(null);
        getContentPane().setBackground(Color.lightGray);

        btnProcesar=new JButton("Procesar");
        btnProcesar.setBounds(170,20,100,20);
        btnProcesar.addActionListener(this);
        getContentPane().add(btnProcesar);

        txtS=new JTextArea();
```

```
        scpScroll=new JScrollPane(txtS);
        scpScroll.setBounds(20,60,430,420);
        getContentPane().add(scpScroll);

    }

    // Procesa eventos de tipo ActionEvent
    public void actionPerformed( ActionEvent e ){
        if(e.getSource()==btnProcesar){
            procesar();
        }
    }

    void procesar(){
        Auto autol=new Auto(5);
        listar(autol);

        Moto motol=new Moto(2);
        listar(motol);

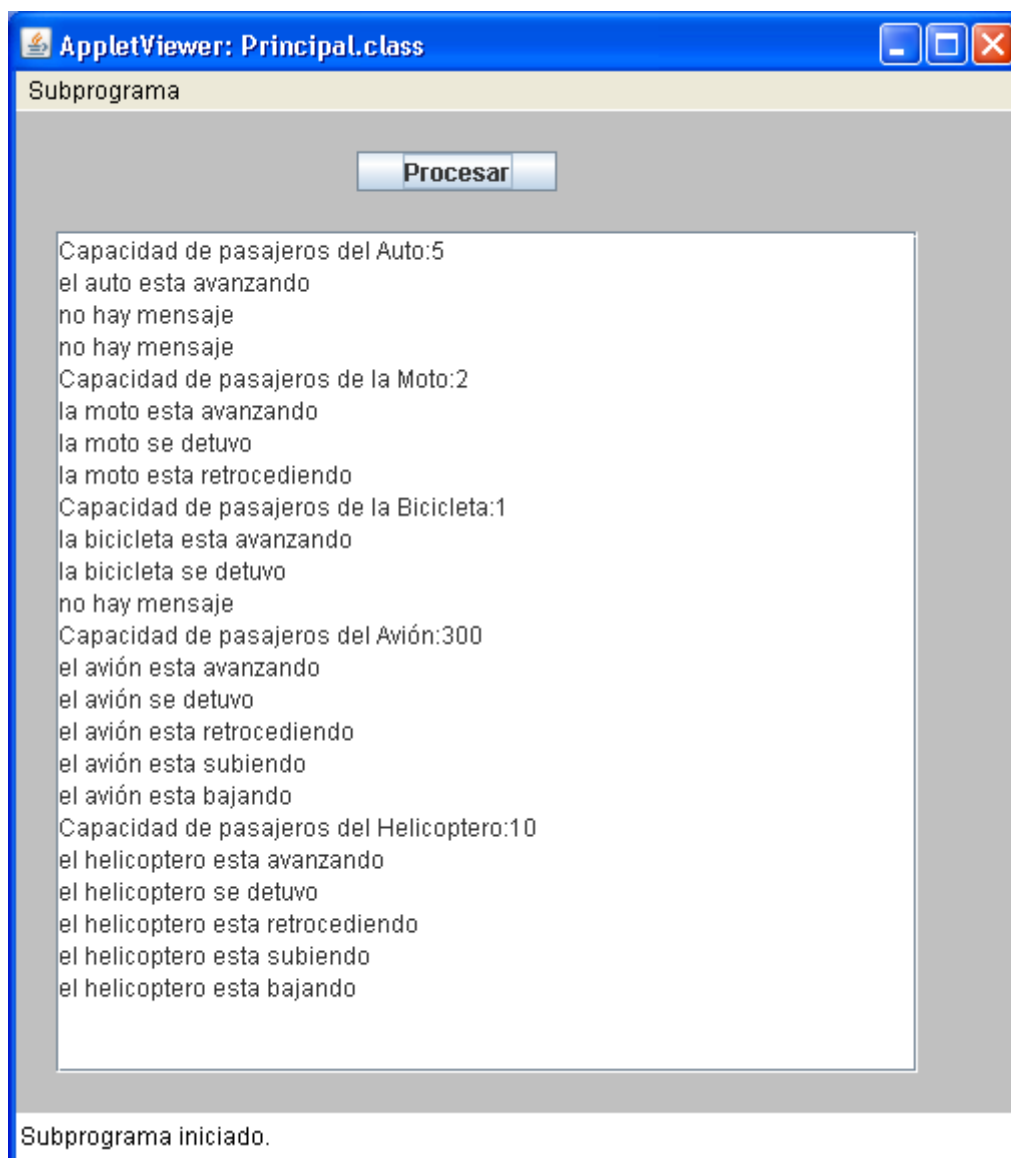
        Bicicleta bicil=new Bicicleta(1);
        listar(bicil);

        Avion avionl=new Avion(300);
        listar(avionl);
        imprimir(avionl.subir());
        imprimir(avionl.bajar());

        Helicoptero helil=new Helicoptero(10);
        listar(helil);
        imprimir(helil.subir());
        imprimir(helil.bajar());
    }

    void listar(Transporte x){
        imprimir(x.mostrarCapacidad());
        imprimir(x.avanzar());
        imprimir(x.detener());
        imprimir(x.retroceder());
    }

    void imprimir(String cad){
        txtS.append(cad + "\n");
    }
}
```

Interfaz Gráfica de usuario:

ANEXO

CASO INMOBILIARIA

OBJETIVO ESPECÍFICO

Solución de un caso práctico donde se emplea la técnica POO

TEMARIO

- Creación de clases, paquetes y objetos
- Modificador Static y referencia this
- Modificadores de acceso : public, private y protected
- Uso de la clase ArrayList y archivos de texto
- Clases y métodos abstractos
- Encapsulamiento, herencia y polimorfismo

Caso inmobiliaria:

Una inmobiliaria desea que le desarrollen una aplicación que le permita dar una información adecuada sobre las propiedades que renta a sus clientes, para lo cual se debe desarrollar lo siguiente:

Mantenimiento de propiedades que renta; es decir, debe considerar las siguientes opciones: ingresos, consultas, modificación y eliminación.

Diariamente acuden muchos clientes a la inmobiliaria buscando Información sobre casas y departamentos que estén disponibles y que cubra sus expectativas; por lo tanto, se desea que la aplicación realice las siguientes **búsquedas**:

- Búsqueda según un intervalo de área
- Búsqueda según un intervalo de precio
- Búsqueda según un intervalo de área y un intervalo de precio
- Búsqueda de la propiedad más barata
- Búsqueda de la propiedad más cara

De cada propiedad se conoce:

Código	: entero y único
Ancho	: real en metros
Largo	: real en metros
Precio	: real en soles
Habitaciones	: entero
Disponibilidad	: true (no rentado) y false (rentado)
Piso	: entero (sólo en el caso de departamentos, se refiere a la ubicación del departamento dentro del edificio)
Jardín	: true (con jardín) y false (sin jardín), sólo para casas

Aplicar los conceptos de la POO aprendidos en el curso.

A continuación, se muestra la interfaz gráfica de usuario (GUI)



En las opciones de mantenimiento de casas y mantenimiento de departamentos, el usuario puede realizar las opciones de ingreso, consulta, modificación y eliminación.

En la opción de búsquedas, puede realizar las consultas de que casas y/o departamentos están disponibles dependiendo de los criterios seleccionados (según el área, precio, el más barato, el más caro, etc).

AppletViewer: Principal.class

Subprograma

Mantenimiento de Casas

Opción: Ingresos ▼

Código:

Ancho:

Largo:

Precio:

Habitaciones:

☐ Disponibilidad
☐ Jardín





Subprograma iniciado.

Utilice el JComboBox de opción para seleccionar la opción de ingreso, consulta, modificación o eliminación.

En el caso de las casas que tengan jardín, debe darle clic a la casilla de verificación Jardín. Sólo cuando una casa no esté disponible porque, por ejemplo, ya se rentó, debe desmarcar la casilla de verificación Disponibilidad.

La información ingresada o modificada alterará a los archivos de texto correspondientes.

AppletViewer: Principal.class

Subprograma

Mantenimiento de Departamentos

Opción: Ingresos

Código:

Ancho:

Largo:

Precio:


Habitaciones:

Piso:

Aceptar

Cancelar

☐ Disponibilidad

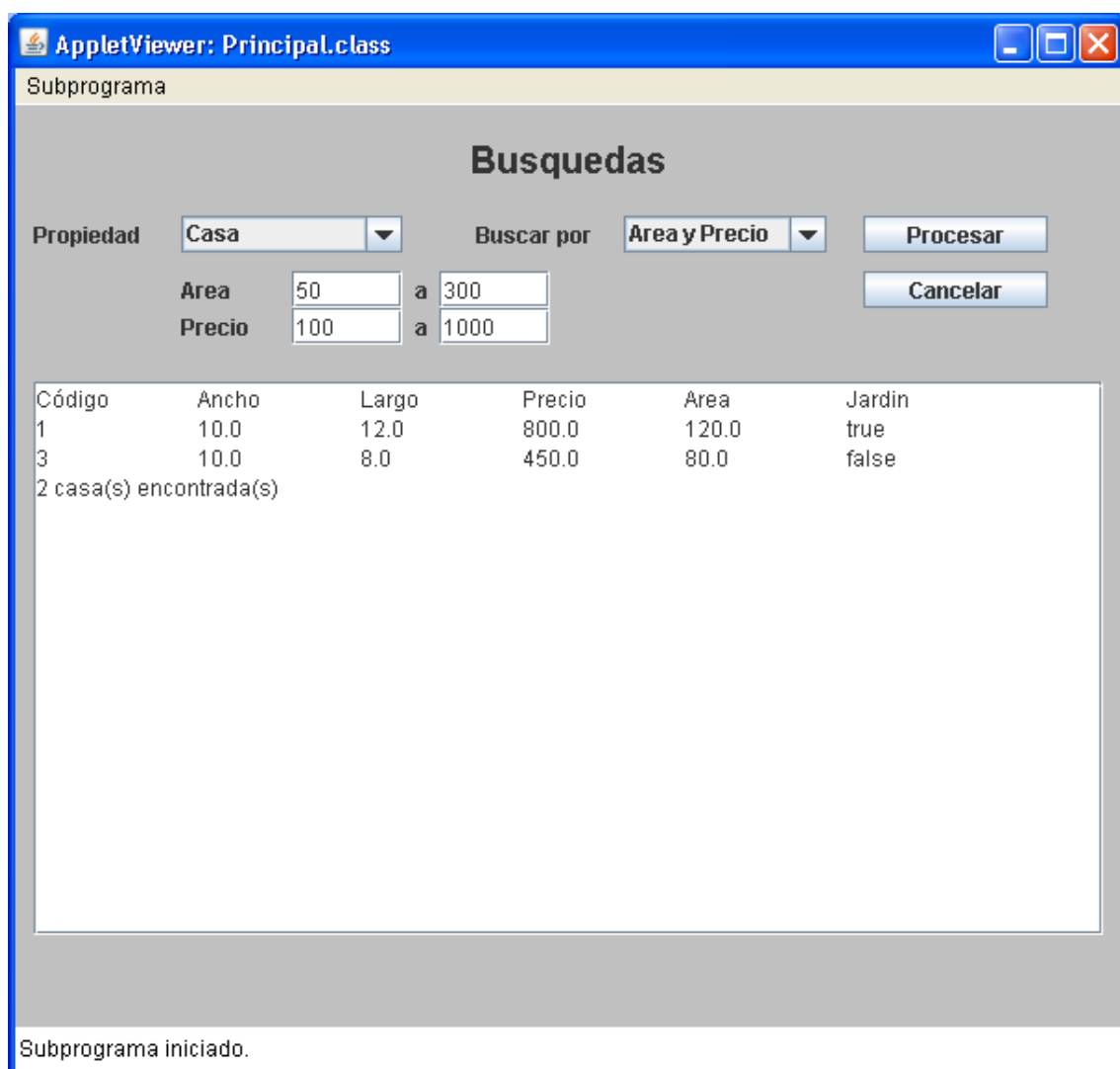


Subprograma iniciado.

Utilice el JComboBox de opción para seleccionar la opción de ingreso, consulta, modificación o eliminación.

El JTextField de piso se refiere a la ubicación del departamento dentro del edificio o condominio.

La información ingresada o modificada alterará a los archivos de texto correspondientes.



Utilice el JComboBox **Propiedad** para seleccionar entre casa o departamento y el JComboBox **Buscar por** para seleccionar el criterio de búsqueda (según área, precio, área y precio, el más barato y el más caro). Ingrese los rangos de área y precio según sea el caso y de clic al botón Procesar para ejecutar la búsqueda.

Solución del caso propuesto:

Clase Propiedad

```
package renta;
public abstract class Propiedad {
    protected int codigo,hab;
    protected double ancho, largo;
    protected double precio;
    protected boolean disp;

    public Propiedad( int codigo, int hab, double ancho, double
    largo, double precio, boolean disp){
        this.codigo = codigo;
        this.hab = hab;
        this.ancho = ancho;
        this.largo = largo;
        this.precio = precio;
        this.disp = disp;
    }

    public void setCodigo( int codigo ){
        this.codigo = codigo;
    }

    public void setHab( int hab ){
        this.hab = hab;
    }

    public void setAncho( double ancho ){
        this.ancho = ancho;
    }

    public void setLargo( double largo ){
        this.largo = largo;
    }

    public void setPrecio( double precio ){
        this.precio = precio;
    }

    public void setDisp( boolean disp ){
        this.disp = disp;
    }

    public int getCodigo(){
        return codigo;
    }

    public int getHab(){
        return hab;
    }

    public double getAncho(){
        return ancho;
    }
}
```

```
    public double getLargo(){
        return largo;
    }

    public double getPrecio(){
        return precio;
    }

    public boolean getDisp(){
        return disp;
    }

    public double area(){
        return ancho*largo;
    }

    public abstract String comoCadena();
}
```

Clase Casa

```
package renta;
public class Casa extends Propiedad {
    private boolean jardin;

    public Casa( int codigo, int hab, double ancho, double
largo, double precio, boolean disp, boolean jardin){
        super(codigo,hab,ancho,largo,precio,disp);
        this.jardin=jardin;
    }

    public void setJardin( boolean jardin ){
        this.jardin = jardin;
    }

    public boolean getJardin(){
        return jardin;
    }

    public String comoCadena(){
        return codigo + "\t" + ancho + "\t" + largo + "\t" +
precio + "\t" + area() + "\t" + jardin;
    }

}
```

Clase Departamento

```
package renta;
public class Departamento extends Propiedad {
    private int piso;

    public Departamento( int codigo, int hab, double ancho,
double largo, double precio, boolean disp, int piso){
```

```
        super(codigo,hab,ancho,largo,precio,disp);
        this.piso=piso;
    }

    public void setPiso( int piso ){
        this.piso = piso;
    }

    public int getPiso(){
        return piso;
    }

    public String comoCadena(){
        return codigo + "\t" + ancho + "\t" + largo + "\t" +
            precio + "\t" + area() + "\t"+ piso;
    }
}
```

Clase ArregloCasas

```
package arreglos;
import java.util.*;
import java.io.*;
import renta.Casa;

public class ArregloCasas{

    private ArrayList <Casa> aCasas;

    public ArregloCasas(){
        aCasas=new ArrayList <Casa> ();
    }

    public void agregar(Casa x){
        aCasas.add(x);
    }

    public Casa obtener(int pos){
        return aCasas.get(pos);
    }

    public int tamaño(){
        return aCasas.size();
    }

    public Casa buscar(int cod){
        for(int i=0; i<aCasas.size(); i++){
            int pcod=obtener(i).getCodigo();
            if(pcod==cod)
                return obtener(i);
        }
        return null;
    }

    public void eliminar(Casa x){
        aCasas.remove(x);
    }
}
```

```

    }

    public double precioMenor(){
        double menor = Double.MAX_VALUE;
        for( int i = 0; i < aCasas.size(); i++ )
            if( aCasas.get(i).getPrecio() < menor &&
                aCasas.get(i).getDisp() )
                menor = aCasas.get(i).getPrecio();
        return menor;
    }

    public double precioMayor(){
        double mayor = Double.MIN_VALUE;
        for( int i = 0; i < aCasas.size(); i++ )
            if( aCasas.get(i).getPrecio() > mayor &&
                aCasas.get(i).getDisp() )
                mayor = aCasas.get(i).getPrecio();
        return mayor;
    }

    public void cargarCasas(){
        try{
            BufferedReader br = new BufferedReader(new
                FileReader("casas.txt"));
            String linea;
            while( (linea = br.readLine()) != null ){
                StringTokenizer tokens = new
                    StringTokenizer(linea, ",");
                int codigo = Integer.parseInt(
                    tokens.nextToken().trim());
                int hab    = Integer.parseInt(
                    tokens.nextToken().trim());
                double ancho = Double.parseDouble(
                    tokens.nextToken().trim());
                double largo = Double.parseDouble(
                    tokens.nextToken().trim());
                double precio = Double.parseDouble(
                    tokens.nextToken().trim());
                boolean disp = Boolean.parseBoolean(
                    tokens.nextToken().trim());
                boolean jardin = Boolean.parseBoolean(
                    tokens.nextToken().trim());
                aCasas.add(new Casa(codigo,hab,ancho,
                    largo,precio,disp,jardin));
            }
            br.close();
        }
        catch( Exception x ){
            System.out.println("Error: " + x);
        }
    }

    public void grabarCasa(){
        try{
            PrintWriter pw = new PrintWriter(new
                FileWriter("casas.txt"));

```

```

        String linea;
        for(int i=0; i<aCasas.size(); i++){
            linea =aCasas.get(i).getCodigo()+ ","+
            aCasas.get(i).getHab()+ ","+
            aCasas.get(i).getAncho()+ ","+
            aCasas.get(i).getLargo()+ ","+
            aCasas.get(i).getPrecio()+ ","+
            aCasas.get(i).getDisp()+ ","+
            aCasas.get(i).getJardin();
            pw.println(linea);
        }
        pw.close();
    }
    catch( Exception x ){
    }
}
}

```

Clase ArregloDepartamentos

```

package arreglos;
import java.util.*;
import java.io.*;
import renta.Departamento;

public class ArregloDepartamentos{

    private ArrayList <Departamento> aDptos;

    public ArregloDepartamentos(){
        aDptos=new ArrayList <Departamento> ();
    }

    public void agregar(Departamento x){
        aDptos.add(x);
    }

    public Departamento obtener(int pos){
        return aDptos.get(pos);
    }

    public int tamaño(){
        return aDptos.size();
    }

    public Departamento buscar(int cod){
        for(int i=0; i<aDptos.size(); i++){
            int pcod=obtener(i).getCodigo();
            if(pcod==cod)
                return obtener(i);
        }
        return null;
    }
}

```

```

public void eliminar(Departamento x){
    aDptos.remove(x);
}

public double precioMenor(){
    double menor = Double.MAX_VALUE;
    for( int i = 0; i < aDptos.size(); i++ )
        if( aDptos.get(i).getPrecio() < menor &&
            aDptos.get(i).getDisp() )
            menor = aDptos.get(i).getPrecio();
    return menor;
}

public double precioMayor(){
    double mayor = Double.MIN_VALUE;
    for( int i = 0; i < aDptos.size(); i++ )
        if( aDptos.get(i).getPrecio() > mayor &&
            aDptos.get(i).getDisp() )
            mayor = aDptos.get(i).getPrecio();
    return mayor;
}

public void cargarDepartamentos(){
    try{
        BufferedReader br = new BufferedReader(new
            FileReader("departamentos.txt"));
        String linea;
        while( (linea = br.readLine()) != null ){
            StringTokenizer tokens = new
                StringTokenizer(linea, ",");
            int codigo = Integer.parseInt(
                tokens.nextToken().trim());
            int hab    = Integer.parseInt(
                tokens.nextToken().trim());
            double ancho = Double.parseDouble(
                tokens.nextToken().trim());
            double largo = Double.parseDouble(
                tokens.nextToken().trim());
            double precio = Double.parseDouble(
                tokens.nextToken().trim());
            boolean disp = Boolean.parseBoolean(
                tokens.nextToken().trim());
            int piso = Integer.parseInt(
                tokens.nextToken().trim());
            aDptos.add(new Departamento(codigo,hab,
                ancho,largo,precio,disp,piso));
        }
        br.close();
    }
    catch( Exception x ){
        System.out.println("Error: " + x);
    }
}

public void grabarDepartamento(){
    try{

```



```

        PrintWriter pw = new PrintWriter(new
        FileWriter("departamentos.txt"));
        String linea;
        for(int i=0; i<aDptos.size(); i++){
            linea =aDptos.get(i).getCodigo()+ ","+
            aDptos.get(i).getHab()+ ","+
            aDptos.get(i).getAncho()+ ","+
            aDptos.get(i).getLargo()+ ","+
            aDptos.get(i).getPrecio()+ ","+
            aDptos.get(i).getDisp()+ ","+
            aDptos.get(i).getPiso();
            pw.println(linea);
        }
        pw.close();
    }
    catch( Exception x ){
        System.out.println("Error: " + x);
    }
}
}

```

Clase LibGUI (Librería)

```

package compartido;
import javax.swing.*;

public class LibGUI {

    private LibGUI(){

    }

    public static int getInteger( JTextField t ){
        return Integer.parseInt(t.getText());
    }

    public static double getDouble( JTextField t ){
        return Double.parseDouble(t.getText());
    }

}

```

Clase Principal

```

import java.awt.*;
import java.applet.*;
import javax.swing.*;
import arreglos.*;

public class Principal extends JApplet {

    PPrincipal pnlPrincipal;
    PCasa pnlCasa;
    PDepartamento pnlDpto;
    PBusqueda pnlBusqueda;
}

```

```
public ArregloCasas aCas = new ArregloCasas();
public ArregloDepartamentos aDpt = new
    ArregloDepartamentos();

public void init() {
    setLayout(null);

    pnlPrincipal=new PPrincipal(this);
    pnlPrincipal.setBounds(0,0,600,500);
    pnlPrincipal.setVisible(true);
    add(pnlPrincipal);

    pnlCasa=new PCasa(this);
    pnlCasa.setBounds(0,0,600,500);
    pnlCasa.setVisible(false);
    add(pnlCasa);

    pnlDpto=new PDepartamento(this);
    pnlDpto.setBounds(0,0,600,500);
    pnlDpto.setVisible(false);
    add(pnlDpto);

    pnlBusqueda=new PBusqueda(this);
    pnlBusqueda.setBounds(0,0,600,500);
    pnlBusqueda.setVisible(false);
    add(pnlBusqueda);

}

}
```

Clase PPrincipal

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PPrincipal extends JPanel implements
    ActionListener {

    private JLabel lblTitulo, lblFoto1, lblFoto2, lblFoto3;
    private JButton btnMtoCasas, btnMtoDptos, btnBusquedas;
    private Principal pri;

    public PPrincipal(Principal x){
        pri=x;

        setLayout(null);
        setBackground(Color.lightGray);

        lblTitulo=new JLabel("Caso Inmobiliaria",
            JLabel.CENTER);
        lblTitulo.setFont(new Font("Arial",Font.BOLD,20));
        lblTitulo.setBounds(0,20,600,20);
        add(lblTitulo);
```

```
        btnMtoCasas=new JButton("Mantenimiento de Casas");
        btnMtoCasas.setBounds(175,100,250,25);
        btnMtoCasas.addActionListener(this);
        add(btnMtoCasas);

        btnMtoDptos=new JButton("Mantenimiento de
        Departamentos");
        btnMtoDptos.setBounds(175,140,250,25);
        btnMtoDptos.addActionListener(this);
        add(btnMtoDptos);

        btnBusquedas=new JButton("Busquedas");
        btnBusquedas.setBounds(175,180,250,25);
        btnBusquedas.addActionListener(this);
        add(btnBusquedas);

        lblFoto1=new JLabel(new ImageIcon("foto1.jpg"));
        lblFoto1.setBounds(50,260,150,100);
        add(lblFoto1);

        lblFoto2=new JLabel(new ImageIcon("foto2.jpg"));
        lblFoto2.setBounds(220,260,150,100);
        add(lblFoto2);

        lblFoto3=new JLabel(new ImageIcon("foto3.jpg"));
        lblFoto3.setBounds(380,260,150,100);
        add(lblFoto3);
    }

    public void actionPerformed(ActionEvent e){
        if(e.getSource()==btnMtoCasas)
            mtoCasas();

        if(e.getSource()==btnMtoDptos)
            mtoDptos();

        if(e.getSource()==btnBusquedas)
            busquedas();
    }

    public void mtoCasas(){
        pri.pnlPrincipal.setVisible(false);
        pri.pnlCasa.setVisible(true);
    }

    public void mtoDptos(){
        pri.pnlPrincipal.setVisible(false);
        pri.pnlDpto.setVisible(true);
    }

    public void busquedas(){
        pri.pnlPrincipal.setVisible(false);
        pri.pnlBusqueda.setVisible(true);
    }
}
```

```
}
```

Clase PFormulario

```
import javax.swing.*;
import java.awt.*;

public class PFormulario extends JPanel{

    protected JComboBox cboOpcion;
    protected JLabel lblTitulo, lblOpcion, lblCodigo,
        lblAncho, lblLargo, lblPrecio, lblHab, lblFoto1,
        lblFoto2, lblFoto3;
    protected JTextField txtCodigo, txtAncho, txtLargo,
        txtPrecio, txtHab;
    protected JCheckBox chkDispo;
    protected JButton btnAceptar, btnCancelar,
        btnModificar, btnEliminar;

    public PFormulario() {

        setLayout(null);
        setBackground(Color.lightGray);

        lblTitulo=new JLabel("",JLabel.CENTER);
        lblTitulo.setFont(new Font("Arial",Font.BOLD,20));
        lblTitulo.setBounds(0,20,600,20);
        add(lblTitulo);

        lblOpcion=new JLabel("Opción:");
        lblOpcion.setBounds(10,70,150,20);
        add(lblOpcion);

        lblCodigo=new JLabel("Codigo:");
        lblCodigo.setBounds(10,92,150,20);
        add(lblCodigo);

        lblAncho=new JLabel("Ancho:");
        lblAncho.setBounds(10,112,150,20);
        add(lblAncho);

        lblLargo=new JLabel("Largo:");
        lblLargo.setBounds(10,132,150,20);
        add(lblLargo);

        lblPrecio=new JLabel("Precio:");
        lblPrecio.setBounds(10,152,150,20);
        add(lblPrecio);

        lblHab=new JLabel("Habitaciones:");
        lblHab.setBounds(10,172,150,20);
        add(lblHab);

        cboOpcion=new JComboBox();
        cboOpcion.setBounds(150,70,150,20);
        cboOpcion.addItem("Ingresos");
```

```
cboOpcion.addItem("Consultas");
cboOpcion.addItem("Modificación");
cboOpcion.addItem("Eliminación");
add(cboOpcion);

txtCodigo=new JTextField();
txtCodigo.setBounds(150,92,150,20);
add(txtCodigo);

txtAncho=new JTextField();
txtAncho.setBounds(150,112,150,20);
add(txtAncho);

txtLargo=new JTextField();
txtLargo.setBounds(150,132,150,20);
add(txtLargo);

txtPrecio=new JTextField();
txtPrecio.setBounds(150,152,150,20);
add(txtPrecio);

txtHab=new JTextField();
txtHab.setBounds(150,172,150,20);
add(txtHab);

chkDispo=new JCheckBox("Disponibilidad");
chkDispo.setBounds(400,152,150,20);
chkDispo.setBackground(Color.lightGray);
add(chkDispo);

btnAceptar = new JButton("Aceptar");
btnAceptar.setBounds(400,80,100,25);
add(btnAceptar);

btnCancelar = new JButton("Cancelar");
btnCancelar.setBounds(400,110,100,25);
add(btnCancelar);

btnModificar = new JButton("Modificar");
btnModificar.setBounds(400,80,100,25);
btnModificar.setVisible(false);
add(btnModificar);

btnEliminar = new JButton("Eliminar");
btnEliminar.setBounds(400,80,100,25);
btnEliminar.setVisible(false);
add(btnEliminar);

lblFoto1=new JLabel(new ImageIcon("foto1.jpg"));
lblFoto1.setBounds(50,260,150,100);
add(lblFoto1);

lblFoto2=new JLabel(new ImageIcon("foto2.jpg"));
lblFoto2.setBounds(220,260,150,100);
add(lblFoto2);
```

```

        lblFoto3=new JLabel(new ImageIcon("foto3.jpg"));
        lblFoto3.setBounds(380,260,150,100);
        add(lblFoto3);
    }

    protected void habilitarControles(){
        txtAncho.setEditable(true);
        txtLargo.setEditable(true);
        txtPrecio.setEditable(true);
        txtHab.setEditable(true);
        chkDispo.setEnabled(true);
    }

    protected void desabilitarControles(){
        txtAncho.setEditable(false);
        txtLargo.setEditable(false);
        txtPrecio.setEditable(false);
        txtHab.setEditable(false);
        chkDispo.setEnabled(false);
    }

    protected void limpiarControles(){
        txtCodigo.setText("");
        txtAncho.setText("");
        txtLargo.setText("");
        txtPrecio.setText("");
        txtHab.setText("");
        chkDispo.setSelected(false);
        cboOpcion.setSelectedIndex(0);
    }

    protected void cambio(){
        btnAceptar.setVisible(true);
        btnModificar.setVisible(false);
        btnEliminar.setVisible(false);
        txtCodigo.setEditable(true);
        cboOpcion.setEnabled(true);
    }
}

```

Clase PCasa

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import compartido.LibGUI;
import renta.Casa;

public class PCasa extends PFormulario implements
ActionListener, ItemListener{
    private Principal pri;
    private boolean flagDispo=false, flagJardin=false,
    existe=false;
    private JCheckBox chkJardin;

    public PCasa(Principal x) {

```

```
        pri=x;

        lblTitulo.setText("Mantenimiento de Casas");

        btnAceptar.addActionListener(this);
        btnCancelar.addActionListener(this);
        cboOpcion.addItemListener(this);
        chkDispo.addItemListener(this);
        btnModificar.addActionListener(this);
        btnEliminar.addActionListener(this);

        chkJardin=new JCheckBox("Jardin");
        chkJardin.setBounds(400,172,150,20);
        chkJardin.setBackground(Color.lightGray);
        chkJardin.addItemListener(this);
        add(chkJardin);

        pri.aCas.cargarCasas();

    }

    public void actionPerformed(ActionEvent e){
        if(e.getSource()==btnAceptar)
            aceptar();

        if(e.getSource()==btnCancelar)
            cancelar();

        if(e.getSource()==btnModificar)
            modificarCasa();

        if(e.getSource()==btnEliminar)
            eliminarCasa();
    }

    public void itemStateChanged(ItemEvent e){
        if(e.getItemSelectable()==chkDispo){
            flagDispo=!flagDispo;
        }

        if(e.getItemSelectable()==chkJardin){
            flagJardin=!flagJardin;
        }

        if(e.getSource()==cboOpcion){
            int indice=cboOpcion.getSelectedIndex();
            if(indice==1 || indice==3)
                desabilitarControles();
            else
                habilitarControles();
        }
    }

    protected void habilitarControles(){
        super.habilitarControles();
    }
}
```

```

        chkJardin.setEnabled(true);
    }

    protected void desabilitarControles(){
        super.desabilitarControles();
        chkJardin.setEnabled(false);
    }

    protected void limpiarControles(){
        super.limpiarControles();
        chkJardin.setSelected(false);
    }

    public void aceptar(){
        int indice=cboOpcion.getSelectedIndex();

        switch(indice){
            case 0: ingresar(); break;
            case 1: consultar(); break;
            case 2: modificar(); break;
            default: eliminar();
        }
    }

    public void cancelar(){
        pri.pnlCasa.setVisible(false);
        pri.pnlPrincipal.setVisible(true);
        habilitarControles();
        limpiarControles();
        existe=false;
        cambio();
    }

    public void ingresar() {
        int cod=LibGUI.getInteger(txtCodigo);
        Casa Ocaso=pri.aCas.buscar(cod);
        if(Ocaso==null){
            double ancho=LibGUI.getDouble(txtAncho);
            double largo=LibGUI.getDouble(txtLargo);
            double precio=LibGUI.getDouble(txtPrecio);
            int hab=LibGUI.getInteger(txtHab);
            Ocaso=new Casa(cod,hab,ancho,largo,
                precio,flagDispo,flagJardin);
            pri.aCas.agregar(Ocaso);
            pri.aCas.grabarCasa();
            JOptionPane.showMessageDialog(this,"Casa
                Agregada");
        }
        else
            JOptionPane.showMessageDialog(this,"Código ya
                existe");

        limpiarControles();
    }

    public void consultar() {

```



```

int cod=LibGUI.getInteger(txtCodigo);
Casa Ocas=pri.aCas.buscar(cod);
if(Ocas!=null){
    txtAncho.setText(""+Ocas.getAncho());
    txtLargo.setText(""+Ocas.getLargo());
    txtPrecio.setText(""+Ocas.getPrecio());
    txtHab.setText(""+Ocas.getHab());
    boolean flag1=Ocas.getDisp();
    boolean flag2=Ocas.getJardin();
    if(flag1)
        chkDispo.setSelected(true);
    else
        chkDispo.setSelected(false);
    if(flag2)
        chkJardin.setSelected(true);
    else
        chkJardin.setSelected(false);
    existe=true;
}
else{
    JOptionPane.showMessageDialog(this,"Código no
    existe");
    limpiarControles();
    existe=false;
}
}

public void modificar(){
    consultar();
    if(existe){
        btnAceptar.setVisible(false);
        btnModificar.setVisible(true);
        txtCodigo.setEditable(false);
        cboOpcion.setEnabled(false);
    }
}

public void modificarCasa() {
    int cod=LibGUI.getInteger(txtCodigo);
    Casa Ocas=pri.aCas.buscar(cod);
    Ocas.setAncho(LibGUI.getDouble(txtAncho));
    Ocas.setLargo(LibGUI.getDouble(txtLargo));
    Ocas.setPrecio(LibGUI.getDouble(txtPrecio));
    Ocas.setHab(LibGUI.getInteger(txtHab));
    Ocas.setDisp(flagDispo);
    Ocas.setJardin(flagJardin);
    pri.aCas.grabarCasa();
    JOptionPane.showMessageDialog(this,"Cambio
    efectuado");
    cambio();
    limpiarControles();
}

public void eliminar(){
    consultar();
    if(existe){

```

```

        btnAceptar.setVisible(false);
        btnEliminar.setVisible(true);
        txtCodigo.setEditable(false);
        cboOpcion.setEnabled(false);
    }
}

public void eliminarCasa(){
    int cod=LibGUI.getInteger(txtCodigo);
    pri.aCas.eliminar(pri.aCas.buscar(cod));
    pri.aCas.grabarCasa();
    JOptionPane.showMessageDialog(this,"Casa eliminada");
    cambio();
    limpiarControles();
}
}

```

Clase PDepartamento

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import compartido.LibGUI;
import renta.Departamento;

public class PDepartamento extends PFormulario implements
ActionListener, ItemListener {

    private Principal pri;
    private boolean flagDispo=false, existe=false;
    private JLabel lblPiso;
    private JTextField txtPiso;

    public PDepartamento(Principal x) {

        pri=x;

        lblTitulo.setText("Mantenimiento de Departamentos");

        lblPiso=new JLabel("Piso:");
        lblPiso.setBounds(10,192,150,20);
        add(lblPiso);

        txtPiso=new JTextField();
        txtPiso.setBounds(150,192,150,20);
        add(txtPiso);

        btnAceptar.addActionListener(this);
        btnCancelar.addActionListener(this);
        cboOpcion.addItemListener(this);
        chkDispo.addItemListener(this);
        btnModificar.addActionListener(this);
        btnEliminar.addActionListener(this);

        pri.aDpt.cargarDepartamentos();
    }
}

```

```
}

public void actionPerformed(ActionEvent e){
    if(e.getSource()==btnAceptar)
        aceptar();

    if(e.getSource()==btnCancelar)
        cancelar();

    if(e.getSource()==btnModificar)
        modificarDpto();

    if(e.getSource()==btnEliminar)
        eliminarDpto();
}

public void itemStateChanged(ItemEvent e){
    if(e.getItemSelectable()==chkDispo){
        flagDispo=!flagDispo;
    }

    if(e.getSource()==cboOpcion){
        int indice=cboOpcion.getSelectedIndex();
        if(indice==1 || indice==3)
            desabilitarControles();
        else
            habilitarControles();
    }
}

protected void habilitarControles(){
    super.habilitarControles();
    txtPiso.setEditable(true);
}

protected void desabilitarControles(){
    super.desabilitarControles();
    txtPiso.setEditable(false);
}

protected void limpiarControles(){
    super.limpiarControles();
    txtPiso.setText("");
}

public void aceptar(){
    int indice=cboOpcion.getSelectedIndex();

    switch(indice){
        case 0: ingresar(); break;
        case 1: consultar(); break;
        case 2: modificar(); break;
        default: eliminar();
    }
}
```

```

public void cancelar(){
    pri.pnlDpto.setVisible(false);
    pri.pnlPrincipal.setVisible(true);
    habilitarControles();
    limpiarControles();
    existe=false;
    cambio();
}

public void ingresar() {
    int cod=LibGUI.getInteger(txtCodigo);
    Departamento Odepa=pri.aDpt.buscar(cod);
    if(Odepa==null){
        double ancho=LibGUI.getDouble(txtAncho);
        double largo=LibGUI.getDouble(txtLargo);
        double precio=LibGUI.getDouble(txtPrecio);
        int hab=LibGUI.getInteger(txtHab);
        int piso=LibGUI.getInteger(txtPiso);
        Odepa=new Departamento(cod,hab,ancho,largo,
            precio,flagDispo,piso);
        pri.aDpt.agregar(Odepa);
        pri.aDpt.grabarDepartamento();
        JOptionPane.showMessageDialog(this,"Departamento
            Agregado");
    }
    else
        JOptionPane.showMessageDialog(this,"Código ya
            existe");

    limpiarControles();
}

public void consultar() {
    int cod=LibGUI.getInteger(txtCodigo);
    Departamento Odepa=pri.aDpt.buscar(cod);
    if(Odepa!=null){
        txtAncho.setText(""+Odepa.getAncho());
        txtLargo.setText(""+Odepa.getLargo());
        txtPrecio.setText(""+Odepa.getPrecio());
        txtHab.setText(""+Odepa.getHab());
        txtPiso.setText(""+Odepa.getPiso());
        boolean flag1=Odepa.getDisp();
        if(flag1)
            chkDispo.setSelected(true);
        else
            chkDispo.setSelected(false);

        existe=true;
    }
    else{
        JOptionPane.showMessageDialog(this,"Código no
            existe");
        limpiarControles();
        existe=false;
    }
}

```

```

    public void modificar(){
        consultar();
        if(existe){
            btnAceptar.setVisible(false);
            btnModificar.setVisible(true);
            txtCodigo.setEditable(false);
            cboOpcion.setEnabled(false);
        }
    }

    public void modificarDpto() {
        int cod=LibGUI.getInteger(txtCodigo);
        Departamento Odepa=pri.aDpt.buscar(cod);
        Odepa.setAncho(LibGUI.getDouble(txtAncho));
        Odepa.setLargo(LibGUI.getDouble(txtLargo));
        Odepa.setPrecio(LibGUI.getDouble(txtPrecio));
        Odepa.setHab(LibGUI.getInteger(txtHab));
        Odepa.setPiso(LibGUI.getInteger(txtPiso));
        Odepa.setDisp(flagDispo);
        pri.aDpt.grabarDepartamento();
        JOptionPane.showMessageDialog(this, "Cambio
        efectuado");
        cambio();
        limpiarControles();
    }

    public void eliminar(){
        consultar();
        if(existe){
            btnAceptar.setVisible(false);
            btnEliminar.setVisible(true);
            txtCodigo.setEditable(false);
            cboOpcion.setEnabled(false);
        }
    }

    public void eliminarDpto() {
        int cod=LibGUI.getInteger(txtCodigo);
        pri.aDpt.eliminar(pri.aDpt.buscar(cod));
        pri.aDpt.grabarDepartamento();
        JOptionPane.showMessageDialog(this, "Departamento
        eliminado");
        cambio();
        limpiarControles();
    }
}

```

Clase PBusqueda

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import compartido.LibGUI;
import renta.Propiedad;

```

```

import arreglos.*;

public class PBusqueda extends JPanel implements
ActionListener, ItemListener {

    private Principal pri;
    private JLabel lblTitulo, lblTipo, lblBusqueda, lblArea,
    lblA1, lblA2, lblPrecio;
    private JComboBox cboTipo, cboBusqueda;
    private JTextField txtAreaMax, txtAreaMin,
    txtPrecioMax, txtPrecioMin;
    private JButton btnProcesar, btnCancelar;
    private JTextArea txtS;
    private JScrollPane scpScroll;

    public PBusqueda(Principal x) {

        pri=x;

        setLayout(null);
        setBackground(Color.lightGray);

        lblTitulo=new JLabel("Busquedas",JLabel.CENTER);
        lblTitulo.setFont(new Font("Arial",Font.BOLD,20));
        lblTitulo.setBounds(0,20,600,20);
        add(lblTitulo);

        lblTipo = new JLabel("Propiedad");
        lblTipo.setBounds(10,60,60,20);
        add(lblTipo);

        cboTipo=new JComboBox();
        cboTipo.setBounds(90,60,120,20);
        cboTipo.addItem("Casa");
        cboTipo.addItem("Departamento");
        add(cboTipo);

        lblBusqueda = new JLabel("Buscar por");
        lblBusqueda.setBounds(250,60,80,20);
        add(lblBusqueda);

        cboBusqueda = new JComboBox();
        cboBusqueda.setBounds(330,60,110,20);
        cboBusqueda.addItem("Area");
        cboBusqueda.addItem("Precio");
        cboBusqueda.addItem("Area y Precio");
        cboBusqueda.addItem("Mas barato");
        cboBusqueda.addItem("Mas caro");
        cboBusqueda.addItemListener(this);
        add(cboBusqueda);

        lblArea = new JLabel("Area");
        lblArea.setBounds(90,90,60,20);
        add(lblArea);
    }

```

```
txtAreaMin = new JTextField();
txtAreaMin.setBounds(150,90,60,20);
add(txtAreaMin);

lblA1 = new JLabel("a", JLabel.CENTER);
lblA1.setBounds(210,90,20,20);
add(lblA1);

txtAreaMax = new JTextField();
txtAreaMax.setBounds(230,90,60,20);
add(txtAreaMax);

lblPrecio = new JLabel("Precio");
lblPrecio.setBounds(90,110,60,20);
add(lblPrecio);

txtPrecioMin = new JTextField();
txtPrecioMin.setBounds(150,110,60,20);
add(txtPrecioMin);

lblA2 = new JLabel("a", JLabel.CENTER);
lblA2.setBounds(210,110,20,20);
add(lblA2);

txtPrecioMax = new JTextField();
txtPrecioMax.setBounds(230,110,60,20);
add(txtPrecioMax);

txtS = new JTextArea();
txtS.setEditable(false);

scpScroll=new JScrollPane(txtS);
scpScroll.setBounds(10, 150, 580, 300);
add(scpScroll);

btnProcesar = new JButton("Procesar");
btnProcesar.setBounds(460,60,100,20);
btnProcesar.addActionListener(this);
btnProcesar.setEnabled(false);
add(btnProcesar);

btnCancelar = new JButton("Cancelar");
btnCancelar.setBounds(460,90,100,20);
btnCancelar.addActionListener(this);
add(btnCancelar);

deshabilita_area();
deshabilita_precio();
}

public void actionPerformed(ActionEvent e){
    if(e.getSource()==btnProcesar)
        procesar();
    if(e.getSource()==btnCancelar)
        cancelar();
}
```

```

public void itemStateChanged(ItemEvent e){
    if(e.getSource()==cboBusqueda){
        int indice=cboBusqueda.getSelectedIndex();
        switch(indice){
            case 0: habilita_area();
                    deshabilita_precio();
                    break;
            case 1: deshabilita_area();
                    habilita_precio();

                    break;
            case 2: habilita_area();
                    habilita_precio();
                    break;
            default:deshabilita_area();
                    deshabilita_precio();
                    break;
        }
        btnProcesar.setEnabled(true);
    }
}

public void habilita_area(){
    lblArea.setVisible(true);
    lblA1.setVisible(true);
    txtAreaMax.setVisible(true);
    txtAreaMin.setVisible(true);
}

public void deshabilita_area(){
    lblArea.setVisible(false);
    lblA1.setVisible(false);
    txtAreaMax.setVisible(false);
    txtAreaMin.setVisible(false);
}

public void habilita_precio(){
    lblPrecio.setVisible(true);
    lblA2.setVisible(true);
    txtPrecioMax.setVisible(true);
    txtPrecioMin.setVisible(true);
}

public void deshabilita_precio(){
    lblPrecio.setVisible(false);
    lblA2.setVisible(false);
    txtPrecioMax.setVisible(false);
    txtPrecioMin.setVisible(false);
}

public void cancelar(){
    pri.pnlBusqueda.setVisible(false);
    pri.pnlPrincipal.setVisible(true);
    deshabilita_area();
    deshabilita_precio();
}

```



```

        btnProcesar.setEnabled(false);
        limpiar();
    }

    public void limpiar(){
        txtAreaMax.setText("");
        txtAreaMin.setText("");
        txtPrecioMax.setText("");
        txtPrecioMin.setText("");
    }

    public void procesar(){
        int indice=cboBusqueda.getSelectedIndex();
        switch(indice){
            case 0: buscar_area();
                    break;
            case 1: buscar_precio();
                    break;
            case 2: buscar_area_precio();
                    break;
            case 3: buscar_mas_barato();
                    break;
            default: buscar_mas_caro();
        }
    }

    public void buscar_area() {
        int indice=cboTipo.getSelectedIndex();
        double areamax = LibGUI.getDouble(txtAreaMax);
        double areamin = LibGUI.getDouble(txtAreaMin);
        int conta = 0;

        imprimir();

        if(indice==0){
            imprimir("Código\tAncho\tLargo\tPrecio\tArea\tJardin");
            for( int i = 0; i < pri.aCas.tamaño(); i++ ){
                Propiedad prop=pri.aCas.obtener(i);

                if(prop.getDisp() && prop.area() >=
                   areamin && prop.area() <= areamax){

                    imprimir(prop.comoCadena());
                    conta++;
                }
            }
            imprimir(conta + " casa(s) encontrada(s)");
        }
        else{
            imprimir("Código\tAncho\tLargo\tPrecio\tArea\tPiso");
            for( int i = 0; i < pri.aDpt.tamaño(); i++ ){
                Propiedad prop=pri.aDpt.obtener(i);

```

```

        if(prop.getDisp() && prop.area() >=
        areamin && prop.area() <= areamax){

            imprimir(prop.comoCadena());
            conta++;

        }
    }
    imprimir(conta + " departamento(s)
    encontrado(s)");
}

}

public void buscar_precio() {
    int indice=cboTipo.getSelectedIndex();
    double preciomax = LibGUI.getDouble(txtPrecioMax);
    double preciomin = LibGUI.getDouble(txtPrecioMin);
    int conta = 0;

    imprimir();
    if(indice==0){

        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tJardin");
        for( int i = 0; i < pri.aCas.tamaño(); i++ ){
            Propiedad prop=pri.aCas.obtener(i);

            if(prop.getDisp() && prop.getPrecio() >=
            preciomin && prop.getPrecio() <= preciomax){

                imprimir(prop.comoCadena());
                conta++;

            }
        }
        imprimir(conta + " casa(s) encontrada(s)");
    }
    else{

        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tPiso");
        for( int i = 0; i < pri.aDpt.tamaño(); i++ ){
            Propiedad prop=pri.aDpt.obtener(i);

            if(prop.getDisp() && prop.getPrecio() >=
            preciomin && prop.getPrecio() <= preciomax){

                imprimir(prop.comoCadena());
                conta++;

            }
        }
        imprimir(conta + " departamento(s)
        encontrado(s)");
    }
}

```

```

public void buscar_area_precio() {
    int indice = cboTipo.getSelectedIndex();
    double areamax = LibGUI.getDouble(txtAreaMax);
    double areamin = LibGUI.getDouble(txtAreaMin);
    double preciomax = LibGUI.getDouble(txtPrecioMax);
    double preciomin = LibGUI.getDouble(txtPrecioMin);
    int conta = 0;

    imprimir();
    if (indice == 0) {
        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tJardin");
        for (int i = 0; i < pri.aCas.tamaño(); i++) {
            Propiedad prop = pri.aCas.obtener(i);

            if (prop.getDisp() && prop.area() >=
                areamin && prop.area() <= areamax &&
                prop.getPrecio() >= preciomin &&
                prop.getPrecio() <= preciomax) {

                imprimir(prop.comoCadena());
                conta++;
            }
        }
        imprimir(conta + " casa(s) encontrada(s)");
    }
    else {
        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tPiso");
        for (int i = 0; i < pri.aDpt.tamaño(); i++) {
            Propiedad prop = pri.aDpt.obtener(i);

            if (prop.getDisp() && prop.area() >=
                areamin && prop.area() <= areamax &&
                prop.getPrecio() >= preciomin &&
                prop.getPrecio() <= preciomax) {

                imprimir(prop.comoCadena());
                conta++;
            }
        }
        imprimir(conta + " departamento(s) encontrado(s)");
    }
}

public void buscar_mas_carro() {
    int indice = cboTipo.getSelectedIndex();
    int conta = 0;
    imprimir();
    if (indice == 0) {

        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tJardin");
    }
}

```

```

        for( int i = 0; i < pri.aCas.tamaño(); i++ ){
            Propiedad prop=pri.aCas.obtener(i);
            if( prop.getDisp() && prop.getPrecio() ==
                pri.aCas.precioMayor() ){
                imprimir(prop.comoCadena());
                conta++;
            }
        }
        imprimir(conta + " casa(s) encontrada(s)");
    }
    else{

        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tPiso");
        for( int i = 0; i < pri.aDpt.tamaño(); i++ ){
            Propiedad prop=pri.aDpt.obtener(i);
            if( prop.getDisp() && prop.getPrecio() ==
                pri.aDpt.precioMayor() ){
                imprimir(prop.comoCadena());
                conta++;
            }
        }
        imprimir(conta + " departamento(s)
        encontrado(s)");
    }
}

public void buscar_mas_barato() {
    int indice=cboTipo.getSelectedIndex();
    int conta = 0;
    imprimir();
    if(indice==0){

        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tJardin");
        for( int i = 0; i < pri.aCas.tamaño(); i++ ){
            Propiedad prop=pri.aCas.obtener(i);
            if( prop.getDisp() && prop.getPrecio() ==
                pri.aCas.precioMenor() ){
                imprimir(prop.comoCadena());
                conta++;
            }
        }
        imprimir(conta + " casa(s) encontrada(s)");
    }
    else{

        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tPiso");
        for( int i = 0; i < pri.aDpt.tamaño(); i++ ){
            Propiedad prop=pri.aDpt.obtener(i);
            if( prop.getDisp() && prop.getPrecio() ==
                pri.aDpt.precioMenor() ){
                imprimir(prop.comoCadena());
                conta++;
            }
        }
    }
}

```

```
        imprimir(conta + " departamento(s)
        encontrado(s)");
    }

    public void imprimir(){
        txtS.setText("");
    }

    public void imprimir(String cad){
        txtS.append(cad + "\n");
    }
}
```