



Algoritmos y Estructura de Datos

Índice

PRESENTACIÓN	4
RED DE CONTENIDOS	5
CLASES Y OBJETOS	6
1.1. CONCEPTOS BÁSICOS DE POO	7
1.1.1. INTRODUCCIÓN	7
1.1.2. CLASE	9
1.1.3. OBJETO	10
1.1.4. DECLARACIÓN Y CREACIÓN DE OBJETOS (OPERADOR NEW)	10
1.1.5. ACCESO A LOS MIEMBROS DE UNA CLASE (OPERADOR PUNTO)	12
1.1.6. PAQUETE	12
1.1.7. ESPECIFICADOR DE ACCESO PUBLIC	13
1.1.8. SENTENCIA IMPORT	14
1.2. CONTROL DE ACCESO Y ENCAPSULAMIENTO	20
1.2.1. DIRECCIÓN DE MEMORIA DE UN OBJETO	20
1.2.2. ASIGNACIÓN ENTRE REFERENCIAS	20
1.2.3. CONSTRUCTOR	22
1.2.4. CREACIÓN E INICIALIZACIÓN DE OBJETOS	23
1.2.5. ESPECIFICADOR DE ACCESO PRIVATE	24
1.2.6. ENCAPSULAMIENTO	25
1.2.7. MÉTODOS DE ACCESO PÚBLICO: SET / GET	25
1.3. MIEMBROS DE CLASE, CONSTANTES Y SOBRECARGA	32
1.3.1. REFERENCIA THIS	32
1.3.2. MODIFICADOR STATIC (ELEMENTOS ÚNICOS)	33
1.3.3. BLOQUE DE INICIALIZACIÓN STATIC	34
1.3.4. SOBRECARGA	34
1.3.5. USO DEL THIS EN SOBRECARGA	36
1.3.6. MODIFICADOR FINAL (CONSTANTES)	36
1.3.7. LIBRERÍA	37
1.4. CLASE STRING	44
1.4.1. DESCRIPCIÓN	44
1.4.2. MÉTODOS BÁSICOS DE LA CLASE STRING	44
1.4.3. CONCATENACIÓN	46
1.4.4. RECORRIDO	46
ARREGLO LINEAL	57
2.1. CONCEPTOS Y OPERACIONES SIMPLES	58
2.1.1. DESCRIPCIÓN	58
2.1.2. DECLARACIÓN E INICIALIZACIÓN	58
2.1.3. DECLARACIÓN PRIVADA E INICIALIZACIÓN	59
2.1.4. RECORRIDO	59
2.1.5. REMPLAZO	60
2.1.6. OPERACIONES PÚBLICAS BÁSICAS	60
2.1.7. OPERACIONES PÚBLICAS COMPLEMENTARIAS	60
2.2. ARTIFICIOS Y OPERACIONES VARIADAS	66
2.2.1. DESCRIPCIÓN	66
2.2.2. DECLARACIÓN, CREACIÓN Y RESERVA	66
2.2.3. DECLARACIÓN PRIVADA, CREACIÓN Y RESERVA	67
2.2.4. INGRESO PERSONALIZADO	67
2.2.5. RECORRIDO	68
2.2.6. REDIMENSIONAMIENTO	68
2.2.7. MÉTODO PRIVADO AMPLIARARREGLO	69
2.2.8. OPERACIONES PÚBLICAS BÁSICAS	69
2.2.9. OPERACIONES PÚBLICAS COMPLEMENTARIAS	69

2.3. ARTIFICIOS Y OPERACIONES ESPECIALES	74
2.3.1. MÉTODO PRIVADO BUSCAR	74
2.3.2. MÉTODO PRIVADO INTERCAMBIAR	74
2.3.3. OPERACIONES PÚBLICAS COMPLEMENTARIAS	75
2.4. ARTIFICIOS Y OPERACIONES COMPLEMENTARIAS	78
2.4.1. MÉTODO PRIVADO ELIMINAR	78
2.4.2. MÉTODO PRIVADO INSERTAR	78
2.4.3. OPERACIONES PÚBLICAS COMPLEMENTARIAS	79
CLASE ARRAYLIST	83
3.1. CONCEPTOS Y OPERACIONES SIMPLES	84
3.1.1. DESCRIPCIÓN	84
3.1.2. COLECCIONISTA DE OBJETOS DISTINTOS	84
3.1.3. COLECCIONISTA DE OBJETOS IGUALES	85
3.1.4. DECLARACIÓN PRIVADA Y CREACIÓN	85
3.1.5. MÉTODOS BÁSICOS DE LA CLASE ARRAYLIST	87
3.1.6. OPERACIONES PÚBLICAS BÁSICAS	88
3.1.7. OPERACIONES PÚBLICAS COMPLEMENTARIAS	88
3.2. OPERACIONES VARIADAS	97
3.2.1. MÉTODOS ADICIONALES DE LA CLASE ARRAYLIST	97
3.2.2. OPERACIONES PÚBLICAS BÁSICAS	99
3.3. MANTENIMIENTO	107
3.3.1. DISEÑO BÁSICO DE UN PROYECTO	107
TÉCNICAS AVANZADAS DE POO	120
4.1. HERENCIA Y POLIMORFISMO	121
4.1.1. GENERALIZACIÓN / ESPECIALIZACIÓN	121
4.1.2. HERENCIA	121
4.1.3. RELACIÓN “ES-UN” O “ES-UNA”	122
4.1.4. USO DE SUPER	123
4.1.5. SOBRESCRITURA DE MÉTODOS	127
4.1.6. CLASES ABSTRACTAS Y MÉTODOS ABSTRACTOS	129
4.1.7. TÉCNICAS DE CASTING	131
4.1.8. POLIMORFISMO Y USO DE INSTANCEOF	135
4.2. INTERFACES	138
4.2.1. DEFINICIÓN	138
4.2.2. HERENCIA MÚLTIPLE	138
ARCHIVOS	146
5.1. MANEJO DE ARCHIVOS DE TEXTO	147
5.1.1. DESCRIPCION	147
5.1.2. CLASES PRINTWRITER Y FILEWRITER	147
5.1.3. CLASES BUFFEREDREADER Y FILEREADER	149
5.1.4. MÉTODO SPLIT DE LA CLASE STRING	150
5.1.5. CLASE FILE	151
5.2. MANTENIMIENTO	152
5.2.1. DISEÑO BÁSICO DE UN PROYECTO CON ARCHIVOS DE TEXTO	152
ANEXO	168
6.1. CASO INMOBILIARIA	169
6.2. MISCELANEA	196

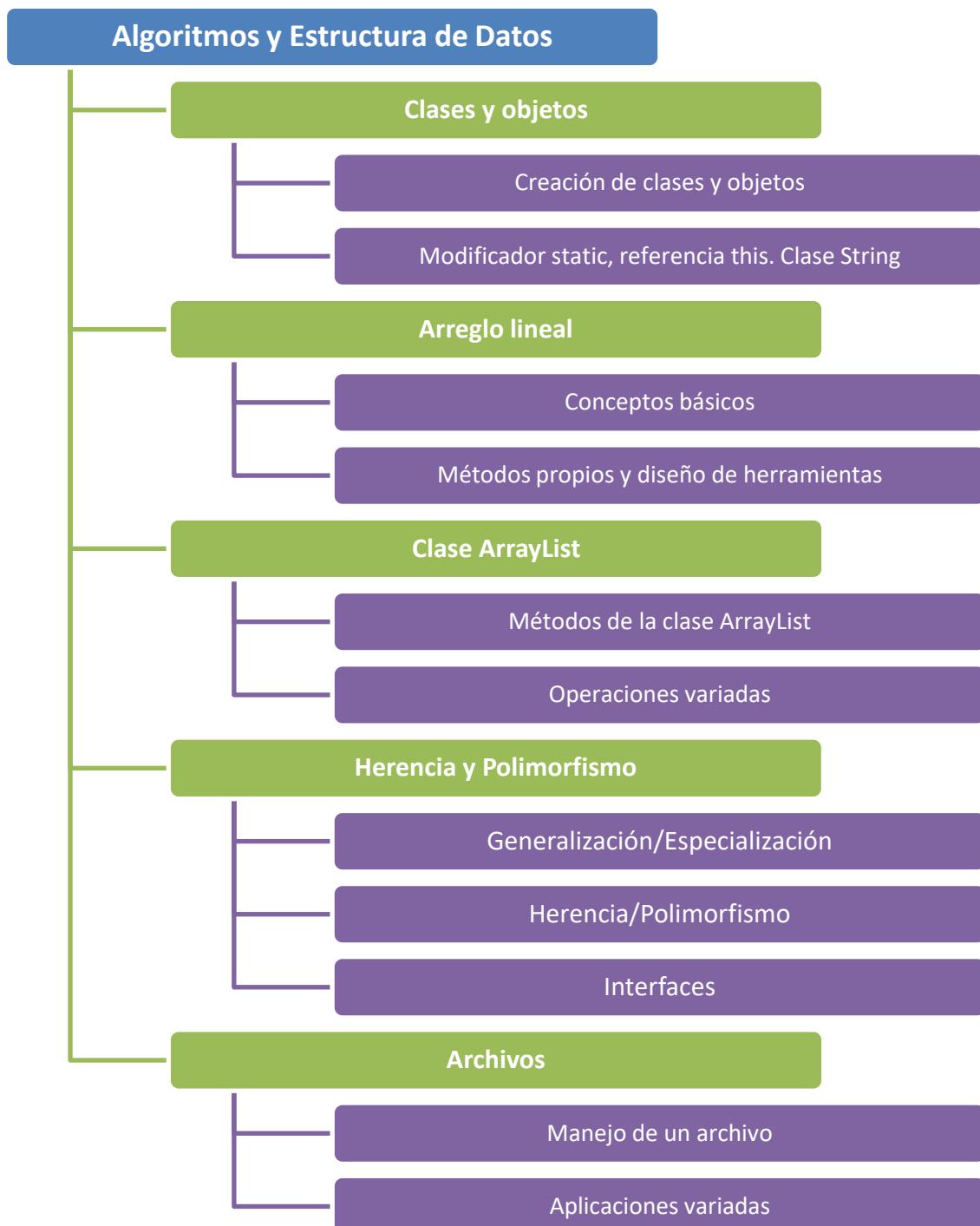
Presentación

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, Administración y Sistemas., Análisis de Datos Empresariales y Arquitectura de Datos Empresariales. 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. Esta 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





CLASES Y OBJETOS

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 la referencia *this* en Java. Finalmente, entienden el concepto de encapsulamiento.

TEMARIO

1.1. Tema 1 : Conceptos básicos de POO

- 1.1.1 : Introducción
- 1.1.2 : Clase
- 1.1.3 : Objeto
- 1.1.4 : Declaración y creación de objetos
- 1.1.5 : Acceso a los miembros de la clase
- 1.1.6 : Paquete
- 1.1.7 : Especificador de acceso *public*
- 1.1.8 : Sentencia *import*

1.2. Tema 2 : Control de acceso y encapsulamiento

- 1.2.1 : Dirección de memoria de un objeto
- 1.2.2 : Asignación entre referencias
- 1.2.3 : Constructor
- 1.2.4 : Creación e inicialización de objetos
- 1.2.5 : Especificador de acceso *private*
- 1.2.6 : Encapsulamiento
- 1.2.7 : Métodos de acceso: *set/get*

1.3. Tema 3 : Miembros de clase, constantes y sobrecarga

- 1.3.1 : Referencia *this*
- 1.3.2 : Modificador *static*
- 1.3.3 : Bloque de inicialización *static*
- 1.3.4 : Sobrecarga
- 1.3.5 : Uso del *this* en sobrecarga
- 1.3.6 : Modificador *final*
- 1.3.7 : Librería

1.4. Tema 4 : Clase String

- 1.4.1 : Descripción
- 1.4.2 : Métodos básicos de la clase String
- 1.4.3 : Concatenación
- 1.4.4 : Recorrido

ACTIVIDADES PROPUESTAS

- Crear objetos de diversas clases.
- Emplear y diferenciar los especificadores de acceso: *public* y *private*.
- Realizar operaciones con objetos creados y entre referencias.
- Entender el concepto de sobrecarga.
- Emplear el modificador e inicializador *static* y reconocer una librería

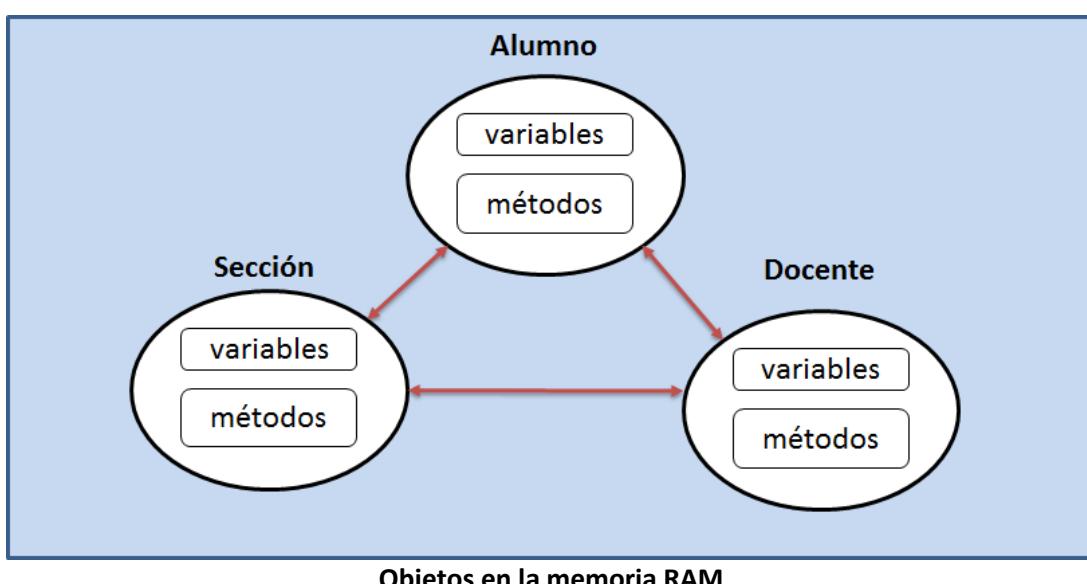
1.1. Conceptos Básicos de POO

1.1.1. Introducción

En el mundo real vivimos rodeados de objetos: personas, sillas, computadoras, tazas, platos, etc.



La POO (Programación Orientada a Objetos) es una metodología de programación que permite crear programas de computadora tomando al objeto como unidad esencial de programación.



Los **objetos** en la POO son entidades o unidades independientes que poseen atributos y operaciones.

Los **atributos** son las características del objeto que se representan mediante variables.

Las **operaciones** son las formas de operar del objeto que se representan mediante métodos.



Persona	
Atributos	Operaciones
nombre	hablar
edad	caminar
peso	dormir
estatura	comer

Clase Persona con atributos y Operaciones



CuentaBancaria	
Atributos	Operaciones
titular	depositar
numero	retirar
saldo	

Clase CuentaBancaria con Atributos y Operaciones

En el mundo real, para poder fabricar objetos, se requiere de un plano o un conjunto de planos.



En forma similar, para poder crear un objeto software, se requiere de una clase o modelo.



1.1.2. Clase

Una clase es una plantilla (más propiamente, un **tipo de dato**) que especifica los *atributos* (mediante variables denominadas *variables miembro*) y las *operaciones* (mediante métodos denominados *métodos miembro*) de un tipo de objeto a crear.

1.1.2.1. Declaración y definición

Una clase se declara y define de la siguiente manera:

```
class NombreClase {
    ...
}
```

Todo nombre de clase debe comenzar con una letra Mayúscula.

Una clase se puede codificar en cualquier parte del programa.

1.1.2.2. Implementación

Los **atributos** se especifican mediante la declaración de “*variables miembro*” y las **operaciones** mediante la implementación de “*métodos miembro*”.

Una clase es una plantilla que especifica los atributos y el comportamiento de un determinado tipo de objeto.

Ejercicio 01: Declare la clase Alumno con:

Atributos:

código,
nombre, y
dos notas

Método: Retorne la nota promedio.

```
class Alumno {
    // Atributos
    int codigo, nota1, nota2;
    String nombre;

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

1.1.3. Objeto

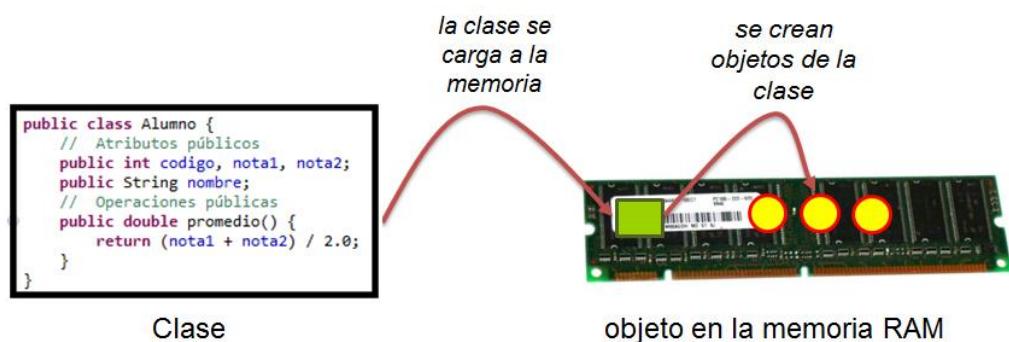
Un objeto es una instancia, es decir, un ejemplar de la clase.

La JVM (*Java Virtual Machine*) 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.

Para cada objeto se creará una copia de cada una de las variables miembro.

Los métodos miembro son compartidos por todos los objetos de la clase.

La clase se carga a la memoria RAM la primera vez que se menciona en el código, en adelante será utilizada desde la RAM para crear objetos (se convierte en una fábrica de objetos).



1.1.4. Declaración y creación de objetos (Operador new)

1.1.4.1. Declaración

Un objeto se declara de la siguiente manera:

```
NombreClase nombreObjeto;
```

Todo nombre de objeto debe comenzar con una letra minúscula.

1.1.4.2. Creación

Un objeto creado referencia una *dirección de memoria*. El operador **new** crea el objeto dinámicamente (en tiempo de ejecución).

Un objeto se crea de la siguiente manera:

```
nombreObjeto = new NombreClase();
```

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.

1.1.4.3. Declaración y creación

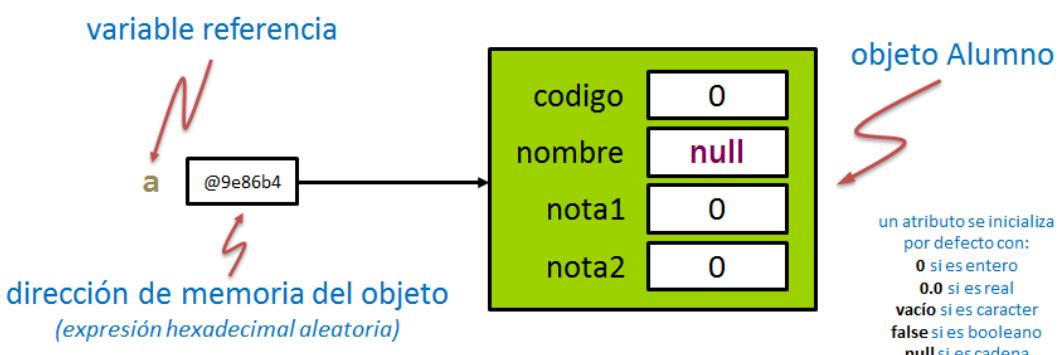
Un objeto se declara y crea al mismo tiempo de la siguiente manera:

```
NombreClase nombreObjeto = new NombreClase();
```

```
Alumno a;
```

a ? Si es global, se inicializa automáticamente con el valor **null**. Si es local, no contiene nada.

```
a = new Alumno();
```



1.1.5. Acceso a los miembros de una clase (Operador punto)

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

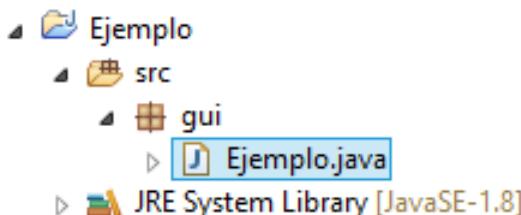
Ejercicio 02: Declare y cree un objeto de tipo Alumno e ingrese datos fijos a sus atributos. Finalmente muestre su dirección de memoria, los datos completos del alumno y también su promedio.

```
protected void actionPerformedBtnProcesar(ActionEvent arg0) {
    Alumno a = new Alumno();
    a.codigo = 12345;
    a.nombre = "Juan";
    a.nota1 = 13;
    a.nota2 = 15;

    imprimir("DirMem : " + a);
    imprimir("código : " + a.codigo);
    imprimir("nombre : " + a.nombre);
    imprimir("nota 1 : " + a.nota1);
    imprimir("nota 2 : " + a.nota2);
    imprimir("promedio : " + a.promedio());
}
```

1.1.6. 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 tiene que colocarse al inicio del archivo y es la primera sentencia en ejecutarse. Ejemplo:

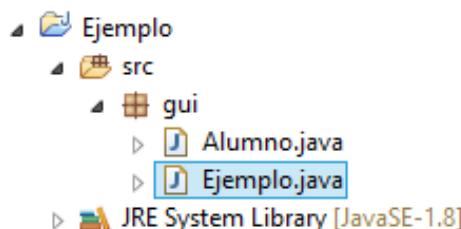


```
package gui;
...
public class Ejemplo extends JFrame {
...
}
```

Al crearse un paquete se crea una carpeta con el mismo nombre del paquete. Si coloca la misma instrucción **package** al inicio de otras clases logrará agrupar varias clases en el mismo paquete. Todo nombre de **package** debe comenzar con una letra minúscula.

1.1.6.1. Archivo independiente

Es recomendable implementar la clase en un archivo independiente.



Ejercicio 03: Declare la clase Alumno en un archivo independiente.

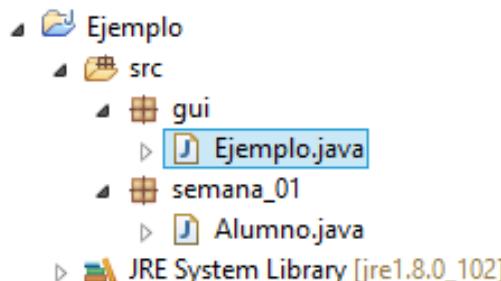
```
package gui;

class Alumno {
    // Atributos
    int codigo, nota1, nota2;
    String nombre;

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

1.1.6.2. Paquete independiente

En ese caso debemos se antepone a la clase la palabra reservada **public**.



Ejercicio 04: Declare la clase Alumno en el paquete independiente semana_01.

```
package semana_01;

public class Alumno {
    // Atributos
    int codigo, nota1, nota2;
    String nombre;

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

1.1.7. Especificador de acceso **public**

Permite el acceso a la clase desde cualquier lugar del programa..

Los atributos y operaciones también tienen que ser declarados como **public**.

Ejercicio 05: Permite el acceso a todos los elementos de la clase Alumno.

```
package semana_01;

public class Alumno {
    // Atributos públicos
    public int codigo, nota1, nota2;
    public String nombre;

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

1.1.8. Sentencia *import*

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

Ejemplo:

```
import semana_01.Alumno;
```

El carácter asterisco indica que se importe todas las clases del paquete.

Ejemplo:

```
import semana_01.*;
```

```
package gui;

import semana_01.Alumno;

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Font;

public class Ejemplo extends JFrame implements ActionListener {
```

```
private static final long serialVersionUID = 1L;

private JPanel contentPane;
private JButton btnProcesar;
private JScrollPane scrollPane;
private JTextArea txtS;

/**
 * Launch the application.
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Ejemplo frame = new Ejemplo();
                frame.setVisible(true);
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
 * Create the frame.
 */
public Ejemplo() {
    setTitle("Ejemplo");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 500);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    btnProcesar = new JButton("Procesar");
    btnProcesar.addActionListener(this);
    btnProcesar.setBounds(173, 11, 89, 23);
    contentPane.add(btnProcesar);

    scrollPane = new JScrollPane();
    scrollPane.setBounds(10, 48, 414, 403);
    contentPane.add(scrollPane);

    txtS = new JTextArea();
    txtS.setFont(new Font("Monospaced", Font.PLAIN, 13));
    scrollPane.setViewportView(txtS);
}

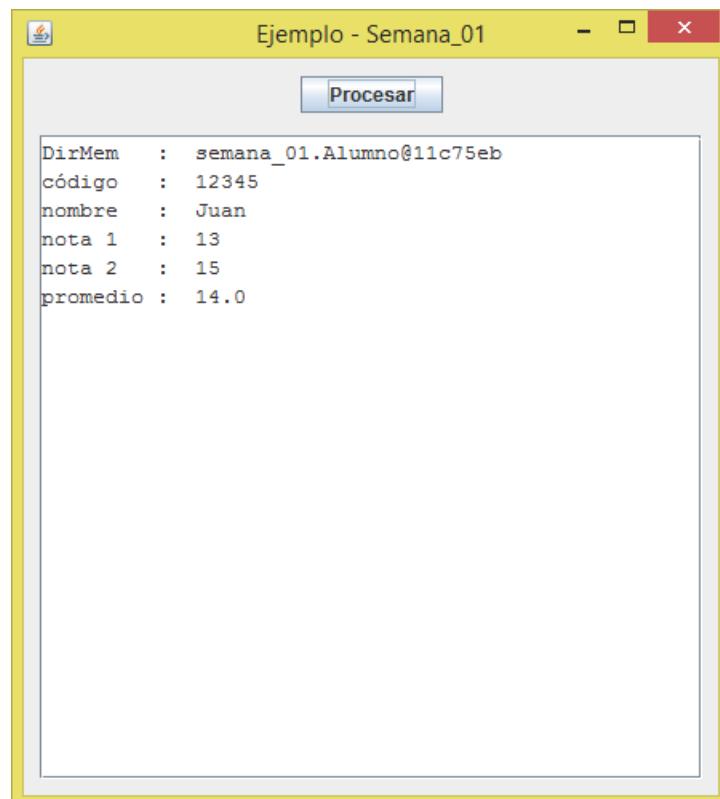
public void actionPerformed(ActionEvent arg0) {
    if (arg0.getSource() == btnProcesar) {
        actionPerformedBtnProcesar(arg0);
    }
}

protected void actionPerformedBtnProcesar(ActionEvent arg0) {
    Alumno a = new Alumno();
```

```
a.codigo = 12345;
a.nombre = "Juan";
a.nota1 = 13;
a.nota2 = 15;

imprimir("DirMem : " + a);
imprimir("código : " + a.codigo);
imprimir("nombre : " + a.nombre);
imprimir("nota 1 : " + a.nota1);
imprimir("nota 2 : " + a.nota2);
imprimir("promedio : " + a.promedio());
}

// Métodos tipo void con parámetros
void imprimir(String s) {
    txtS.append(s + "\n");
}
}
```



Ejercicios

Ejercicio Propuesto 1

Diseñe la clase **Trabajador** en el paquete **semana_01** con los atributos públicos: código (*int*), nombre (*String*), horas trabajadas (*int*) y tarifa horaria (*double*).

Implemente además:

- Un método que retorne el sueldo bruto (horas * tarifa).
- Un método que retorne el descuento de acuerdo a la siguiente tabla:

Sueldo bruto	Descuento
< 4500	12% del sueldo bruto
≥ 4500 y < 6500	14% del sueldo bruto
≥ 6500	16% del sueldo bruto

- Un método que retorne el sueldo neto (sueldo bruto - descuento).

En la clase principal, a la pulsación del botón Procesar:

- Declare y cree un objeto de tipo Trabajador.
- Ingrese datos fijos.
- Visualice todos sus datos.

Ejercicio Propuesto 2

Diseñe la clase **Filmacion** en el paquete **semana_01** con los atributos públicos: código (*int*), título (*String*), duración en minutos (*entero*) y precio en soles (*double*).

Implemente además:

- Un método que retorne el precio del video en dólares. Considere que: 1 dólar = 3.38 soles.

En la clase principal, a la pulsación del botón Procesar:

- Declare y cree un objeto de tipo Filmacion.
- Ingrese datos fijos.
- Visualice todos sus datos.

Ejercicio Propuesto 3

Diseñe la clase **Persona** en el paquete **semana_01** con los atributos públicos: nombre (*String*), apellido (*String*), edad (*int*), estatura (*double*) y peso (*double*).

Implemente además:

- Un método que retorne el estado de la persona entre: "menor de edad" o "mayor de edad".
- Un método que retorne el índice de masa corporal de la persona (peso/estatura²).

En la clase principal, a la pulsación del botón Procesar:

- Declare y cree un objeto de tipo Persona.
- Ingrese datos fijos.
- Visualice todos sus datos.

Ejercicio Propuesto 4

Diseñe la clase **Coordinador** en el paquete **semana_01** con los atributos públicos: código (*int*), nombre (*String*), categoría (*int*) y número de celular (*int*).

Implemente además:

- Un método que retorne el sueldo del coordinador sabiendo que:

categoría	sueldo
0	S/. 8500
1	S/. 6850
2	S/. 5500

En la clase principal, a la pulsación del botón Procesar:

- Declare y cree un objeto de tipo Coordinador.
- Ingrese datos fijos.
- Visualice todos sus datos.

Ejercicio Propuesto 5

Diseñe la clase **Expositor** en el paquete **semana_01** con los atributos públicos: código (*int*), nombre (*String*), horas trabajadas (*int*) y tarifa por hora (*double*).

Implemente además:

- Un método que retorne el sueldo bruto (horas * tarifa).
- Un método que retorne el descuento por AFP (10% del sueldo bruto).
- Un método que retorne el descuento por EPS (5% del sueldo bruto).
- Un método que retorne el sueldo neto (sueldo bruto – descuentoAFP – descuentoEPS).

En la clase principal, a la pulsación del botón Procesar:

- Declare y cree un objeto de tipo Expositor.
- Ingrese datos fijos.
- Visualice todos sus datos.

Ejercicio Propuesto 6

Diseñe la clase **Docente** en el paquete **semana_01** con los atributos públicos: código (*int*), nombre (*String*), horas trabajadas (*int*) y tarifa por hora (*double*).

Implemente además:

- Un método público que retorne el sueldo (horas * tarifa).

En la clase principal, a la pulsación del botón Procesar:

- Declare y cree un objeto de tipo Docente.
- Ingrese datos fijos.
- Visualice todos sus datos.

Ejercicio Propuesto 7

Diseñe la clase **Numeros** en el paquete **semana_01** con los atributos públicos: numero1 (*int*), numero2 (*int*) y numero3 (*int*).

Implemente además los siguientes métodos:

- Un método que retorne el número menor.
- Un método que retorne el número mayor.
- Un método que retorne el número del medio (ubicado entre el menor y el mayor).

En la clase principal, a la pulsación del botón Procesar:

- Declare y cree un objeto de tipo Numeros.
- Ingrese datos fijos.
- Visualice todos sus datos.

Ejercicio Propuesto 8

Diseñe la clase **Caja** en el paquete **semana_01** con los atributos públicos: largo en centímetros (*double*), ancho en centímetros (*double*), alto en centímetros (*double*) y peso de balanza en kilogramos (*double*).

Implemente además los siguientes métodos:

- Un método que retorne el volumen de la caja en centímetros cúbicos (largo * ancho * alto).
- Un método que retorne el peso volumétrico en kilogramos: (volumen en centímetros cúbicos / 5000).
- Un método que retorne el peso facturable que es el mayor peso entre el peso de balanza y el peso volumétrico.

En la clase principal, a la pulsación del botón Procesar:

- Declare y cree un objeto de tipo Caja.
- Ingrese datos fijos.
- Visualice todos sus datos.

Ejercicio Propuesto 9

Diseñe la clase **Equipo** en el paquete **semana_01** con los atributos públicos: codigo (*int*), marca (*String*), color (*String*) y precio en dólares (*double*).

Implemente además:

- Un método que retorne el precio del equipo en soles. Considere que: 1 dólar = 3.38 soles.
- Un método que retorne el precio del equipo en euros. Considere que: 1 euro = 1.20 dólares.

En la clase principal, a la pulsación del botón Procesar:

- Declare y cree un objeto de tipo Equipo.
- Ingrese datos fijos.
- Visualice todos sus datos.

1.2. Control de Acceso y Encapsulamiento

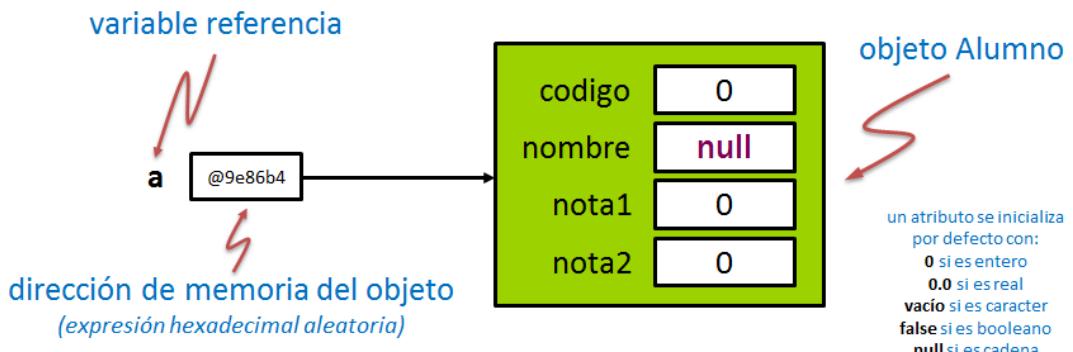
1.2.1. Dirección de memoria de un objeto

```
Alumno a;
```

a ?

Si es global, se inicializa automáticamente con el valor **null**. Si es local, no contiene nada.

```
a = new Alumno();
```



1.2.2. Asignación entre referencias

```
Alumno b;
```

b ?

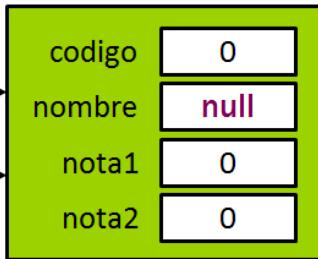
sin valor

```
b = a;
```

variables referencia

a @1c30de5

b @1c30de5

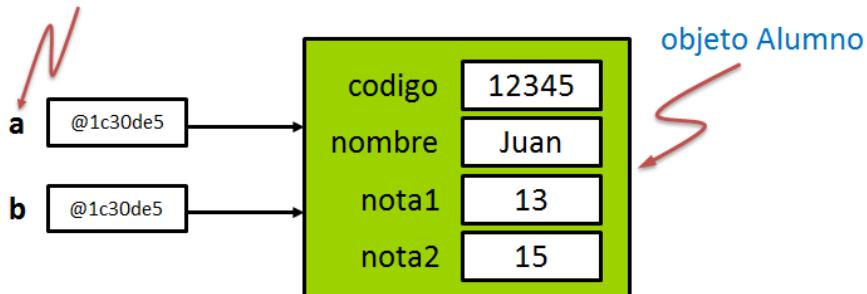


objeto Alumno

b recibe la dirección de memoria almacenada en **a**,
luego de lo cual, tanto **b** como **a** observan o controlan al mismo objeto.

```
b.codigo = 12345; b.nombre = "Juan"; b.nota1 = 13; b.nota2 = 15;
```

variables referencia



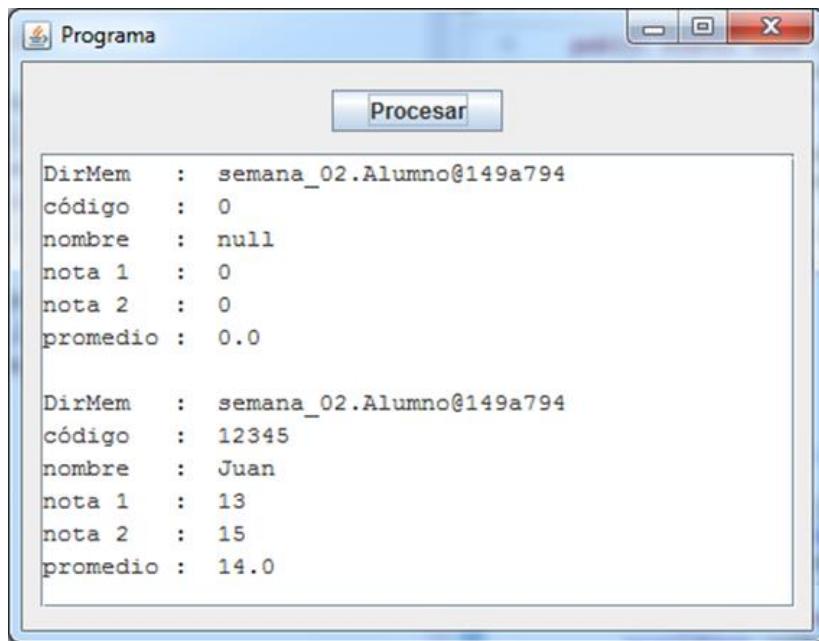
A través de la variable referencia **b**
se han asignado los datos al alumno

Ejercicio 06: Declare y cree el objeto a de tipo Alumno. Visualice la información por defecto a través del método listado que recibe en el parámetro x la referencia del objeto. Declare luego el Alumno b y asígnelo directamente al objeto a. A través del objeto b ingrese datos al alumno y visualice la información del objeto a.

```
protected void actionPerformedBtnProcesar(ActionEvent arg0) {
    Alumno a = new Alumno();
    listado(a);

    Alumno b = a;
    b.codigo = 12345;
    b.nombre = "Juan";
    b.nota1 = 13;
    b.nota2 = 15;
    listado(a);
}

// Métodos tipo void con parámetros
void listado(Alumno x) {
    imprimir("DirMem : " + x);
    imprimir("código : " + x.codigo);
    imprimir("nombre : " + x.nombre);
    imprimir("nota 1 : " + x.nota1);
    imprimir("nota 2 : " + x.nota2);
    imprimir("promedio : " + x.promedio());
}
```



1.2.3. Constructor

Es el mecanismo que permite crear un objeto. 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 constructor, Java asume por defecto un constructor sin parámetros que no realiza actividad adicional. Un constructor permite además inicializar los atributos del objeto al momento de crearlo.

Ejercicio 07: Implemente un constructor en la clase Alumno que reciba información (a través de parámetros) e inicialice a los atributos.

```
package semana_02;

public class Alumno {
    // Atributos públicos
    public int código, nota1, nota2;
    public String nombre;

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

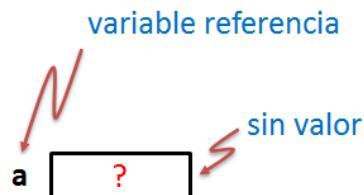
    // Operaciones públicas
    public double promedio() {
        return (nota1 + nota2) / 2.0;
    }
}
```

1.2.4. Creación e Inicialización de Objetos

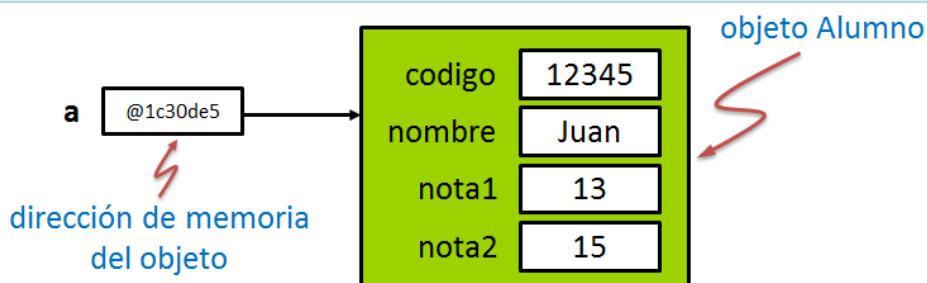
Un objeto puede ser declarado, creado e inicializado de dos maneras.

Forma 1

```
Alumno a;
```

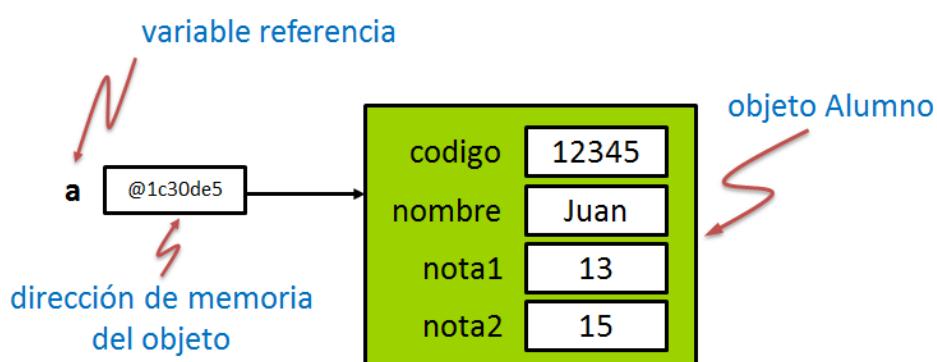


```
a = new Alumno(12345, "Juan", 13, 15);
```



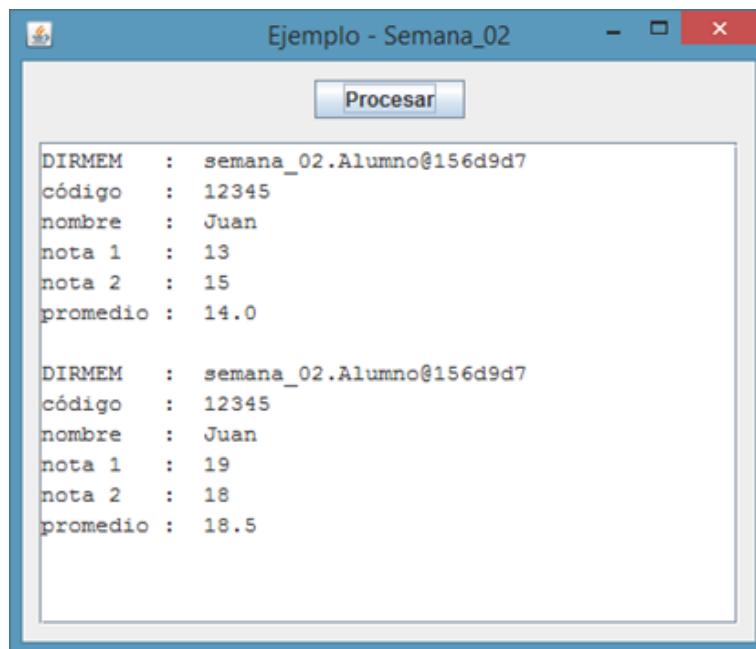
Forma 2

```
Alumno a = new Alumno(12345, "Juan", 13, 15);
```



Ejercicio 08: A la pulsación del botón Procesar declare, cree e inicialice un objeto de tipo Alumno con datos fijos, modifique luego sus notas y visualice nuevamente la información actual.

```
protected void actionPerformedBtnProcesar(ActionEvent arg0) {  
    Alumno a = new Alumno(12345, "Juan", 13, 15);  
    listado(a);  
  
    a.nota1 = 19;  
    a.nota2 = 18;  
    listado(a);  
}
```



1.2.5. Especificador de acceso **private**

Prohibe el acceso externo a los elementos declarados de esta forma. Al declarar los atributos como privados no podrán ser invocados desde afuera de la clase. En el interior de la clase el **private** no tiene efecto.

Ejemplo:

```
package semana_02;  
  
public class Alumno {  
    // Atributos privados  
    private int codigo, nota1, nota2;  
    private String nombre;  
    ...  
}
```

1.2.6. 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.

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 de acceso	acceso a los miembros			
	desde la misma clase	Desde una sub clase	desde una clase del mismo paquete	desde el exterior de la misma clase
public	<i>si</i>	<i>Si</i>	<i>si</i>	<i>si</i>
private	<i>si</i>	<i>no</i>	<i>no</i>	<i>no</i>
protected	<i>si</i>	<i>si</i>	<i>si</i>	<i>no</i>
sin especificador	<i>si</i>	<i>no</i>	<i>si</i>	<i>no</i>

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.

1.2.7. Métodos de acceso público: *set / get*

Para que una clase se considere encapsulada debe tener:

1. **Atributos privados.** Ejemplo:

```
private int codigo;
```

2. **Métodos públicos para acceder a los atributos privados.** Se forman con los prefijos **set** o **get** seguido del nombre del atributo.

- 2.1. Todo método de acceso público **set** es del tipo void y a través de un parámetro recibe un nuevo contenido a imponer en el atributo privado.

Ejemplo:

```
public void setCodigo(int cod) {  
    codigo = cod;  
}
```

- 2.2. Todo método de acceso público **get** responde al tipo de dato del atributo privado, no usa parámetros y retorna el contenido actual del atributo privado.

Ejemplo:

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

Ejercicio 09: Haga que las variables miembro de la clase Alumno sean de uso privado y declare sus respectivos métodos de acceso set/get.

```
package semana_02;  
  
public class Alumno {  
    // Atributos privados  
    private int codigo, nota1, nota2;  
    private String nombre;  
  
    // Constructor  
    public Alumno(int cod, String nom, int n1, int n2) {  
        codigo = cod;  
        nombre = nom;  
        nota1 = n1;  
        nota2 = n2;  
    }  
  
    // Métodos públicos de acceso: set/get  
    public void setCodigo(int cod) {  
        codigo = cod;  
    }  
  
    public void setNombre(String nom) {  
        nombre = nom;  
    }  
  
    public void setNota1(int n1) {  
        nota1 = n1;  
    }  
  
    public void setNota2(int n2) {  
        nota2 = n2;  
    }  
  
    public int getCodigo() {  
        return codigo;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
}
```

```

public int getNota1() {
    return nota1;
}

public int getNota2() {
    return nota2;
}

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

```

Ejercicio 10: A la pulsación del botón Procesar declare, cree e inicialice un objeto de tipo Alumno con datos fijos, modifique luego sus notas y visualice nuevamente la información actual.

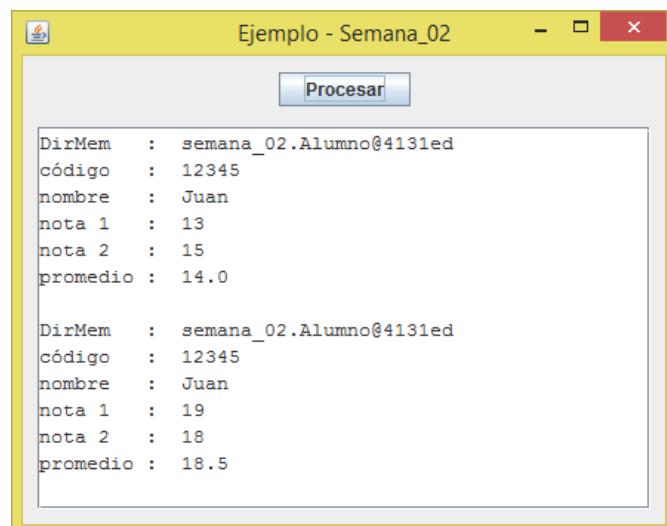
```

protected void actionPerformedBtnProcesar(ActionEvent arg0) {
    Alumno a = new Alumno(12345, "Juan", 13, 15);
    listado(a);

    a.setNota1(19);
    a.setNota2(18);
    listado(a);
}

// Métodos tipo void con parámetros
void listado(Alumno x) {
    imprimir("DirMem : " + x);
    imprimir("código : " + x.getCódigo());
    imprimir("nombre : " + x.getNombre());
    imprimir("nota 1 : " + x.getNota1());
    imprimir("nota 2 : " + x.getNota2());
    imprimir("promedio : " + x.promedio());
}

```



Ejercicios

Ejercicio Propuesto 1

Diseñe la clase **Celular** en el paquete **semana_02** con los atributos privados: número (*int*), usuario (*String*), segundos consumidos (*int*) y precio por segundo (*double*).

Implemente además:

- Un Constructor que inicialice los atributos.
- Métodos de acceso público set/get para todos los atributos privados.
- Un método que retorne el costo por consumo (segundos*precio).
- Un método que retorne el impuesto por IGV (18% del costo por consumo).
- Un método que retorne el total a pagar (costo por consumo + impuesto por IGV).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice un objeto de tipo Celular (con datos fijos).
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Celular y visualice sus datos completos.
- Aumente en 20 el número de segundos consumidos, disminuya en 5% el precio por segundo e invoque nuevamente al método listado.

Ejercicio Propuesto 2

Diseñe la clase **Computadora** en el paquete **semana_02** con los atributos privados: código (*int*), marca (*String*), color (*String*) y precio en dólares (*double*).

Implemente además:

- Un Constructor que inicialice los atributos.
- Métodos de acceso público set/get para todos los atributos privados.
- Un método que retorne el precio de la computadora en soles (1 dólar = 3.25 soles).
- Un método que retorne el precio de la computadora en euros (1 euro = 1.20 dólares).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice un objeto de tipo Computadora (con datos fijos).
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Computadora y visualice sus datos completos.
- Disminuya en 10% el precio en dólares de la computadora e invoque nuevamente al método listado.

Ejercicio Propuesto 3

Diseñe la clase **Edificio** en el paquete **semana_02** con los atributos privados: código (*int*), número de departamentos (*int*), cantidad de pisos del edificio (*int*) y precio de un departamento en dólares (*double*).

Implemente además:

- Un Constructor que inicialice los atributos.
- Métodos de acceso público set/get para todos los atributos privados.
- Un método que retorne el precio en dólares del edificio (#dptos * precio de un dpto).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice un objeto de tipo Edificio (con datos leídos por GUI).

- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Edificio y visualice sus datos completos.
- Incremente en 20% el precio de un departamento e invoque nuevamente al método listado.

Ejercicio Propuesto 4

Diseñe la clase **Obrero** en el paquete **semana_02** con los atributos privados: código (*int*), nombre (*String*), horas trabajadas (*int*) y tarifa por hora (*double*). Implemente además:

- Un Constructor que inicialice los atributos.
- Métodos de acceso público set/get para todos los atributos privados.
- Un método que retorne el sueldo bruto (horas*tarifa).
- Un método que retorne el descuento por AFP (10% del sueldo bruto).
- Un método que retorne el descuento por EPS (5% del sueldo bruto).
- Un método que retorne el sueldo neto (sueldoBruto – descuentoAFP – descuentoEPS).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice un objeto de tipo Obrero (con datos fijos).
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Obrero y visualice sus datos completos.
- Aumente en 8 el número horas trabajadas, disminuya en 1.5% la tarifa por hora e invoque nuevamente al método listado.

Ejercicio Propuesto 5

Diseñe la clase **Video** en el paquete **semana_02** con los atributos privados: código (*int*), nombre del video (*String*), duración (*double*), precio en soles (*double*) y tipo de cambio (*double*).

Implemente además:

- Un Constructor que inicialice los atributos.
- Métodos de acceso público set/get para todos los atributos privados.
- Un método que retorne el precio del video en dólares.

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice un objeto de tipo Video (con datos fijos).
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Video y visualice sus datos completos.
- Aumente en 5.50 el precio del video e invoque nuevamente al método listado.

Ejercicio Propuesto 6

Diseñe la clase **Paciente** en el paquete **semana_02** con los atributos privados: nombre (*String*), apellido (*String*), edad (*int*), talla (*double*) y peso (*double*).

Implemente además:

- Un Constructor que inicialice los atributos.
- Métodos de acceso público set/get para todos los atributos privados.
- Un método que retorne el texto: “menor de edad” o “mayor de edad”.

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice un objeto de tipo Paciente (con datos fijos).

- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Paciente y visualice sus datos completos.
- Modifique la edad del paciente e invoque nuevamente al método listado.

Ejercicio Propuesto 7

Diseñe la Clase **Empleado** en el paquete **semana_02** con los atributos privados: código (*int*), nombre (*String*), número de celular (*int*) y sueldo en soles (*double*).

Implemente además:

- Un Constructor que inicialice los atributos.
- Métodos de acceso público set/get para todos los atributos privados.
- Un método que retorne un texto indicando si el sueldo es “mayor a 3500”, “menor a 3500” o “igual a 3500”.

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice un objeto de tipo Empleado (con datos leídos por GUI).
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Empleado y visualice sus datos completos.
- Cambie el número de celular por 999888777, aumente el sueldo en 200 soles e invoque nuevamente al método listado.

Ejercicio Propuesto 8

Diseñe la Clase **Asesor** en el paquete **semana_02** con los atributos privados: código (*int*), nombre (*String*), horas trabajadas (*int*) y tarifa por hora (*double*).

Implemente además:

- Un Constructor que inicialice los atributos.
- Métodos de acceso público set/get para todos los atributos privados.
- Un método que retorne el sueldo bruto (horas*tarifa).
- Un método que retorne el descuento (15% del sueldo bruto).
- Un método que retorne el sueldo neto (sueldo bruto - descuento).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice un objeto de tipo Asesor (con datos fijos).
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Asesor y visualice sus datos completos.
- Aumente en 10 el número de horas, disminuya en 15% la tarifa e invoque nuevamente al método listado.

Ejercicio Propuesto 9

Diseñe la Clase **Pelota** en el paquete **semana_02** con atributos privados: marca (*String*), peso en gramos (*double*), presión en libras (*double*), diámetro en centímetros (*double*) y precio (*double*).

Implemente además:

- Un Constructor que inicialice los atributos.
- Métodos de acceso público set/get para todos los atributos privados.
- Un método que retorne el radio (diámetro / 2).
- Un método que retorne el volumen del balón ($4 * 3.1416 * \text{radio} * \text{radio} * \text{radio} / 3$).
- Un método que retorne el descuento (10% del precio).
- Un método que retorne el importe a pagar (precio - descuento).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice un objeto de tipo Pelota (con datos fijos).
- invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Pelota y visualice sus datos completos.
- Disminuya en 25% el precio, aumente en 1 centímetro el diámetro e invoque nuevamente al método listado.

1.3. Miembros de Clase, Constantes y Sobrecarga

1.3.1. Referencia *this*

Es un parámetro oculto añadido por el compilador. Una de sus aplicaciones más comunes es cuando el nombre de un parámetro coincide en sintaxis con el atributo. En este caso, su objetivo es diferenciar la *variable miembro* del parámetro en sí. Lo veremos en el constructor y cada método *set*.

Ejemplo:

```
package semana_03;

public class Alumno {

    // Atributos privados
    private int codigo, nota1, nota2;
    private String nombre;

    // Constructor
    public Alumno(int codigo, String nombre, int nota1, int nota2) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.nota1 = nota1;
        this.nota2 = nota2;
    }

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

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

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

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

Dentro de un método de la clase, la referencia **this** contiene la dirección de memoria del objeto que invocó al método. Se usa también para referenciar al objeto que invocó a un método.

```
public class Alumno {
    ...
    public String mejorAlumno(Alumno x) {
        if (this.promedio() > x.promedio())
            return this.nombre;
        else
            return x.nombre;
    }
}

protected void actionPerformedBtnProcesar(ActionEvent arg0) {
    Alumno a = new Alumno(12345, "Juan", 13, 15);
    Alumno b = new Alumno(67890, "Pedro", 12, 13);

    String mejor = a.mejorAlumno(b);
    imprimir("Mejor alumno : " + mejor);
}
```

→ Mejor alumno : Juan

1.3.2. Modificador **static** (elementos únicos)

Se usa para hacer que un atributo o método se convierta en único para todos los objetos. Este mecanismo permite manejar contadores y acumuladores desde el interior de la Clase.

Al anteponer **static** a un atributo la declaración se convierte en **variable de clase**. Al anteponer **static** a un método la declaración se convierte en **método de clase**. Si la **variable de clase** es pública se accede a ella por medio de la misma Clase o por medio del objeto. Pero si la **variable de clase** es privada debemos implementar un **método de clase** público **get**.

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

```
public class Alumno {
    // Variable de clase privada
    private static int cantidad = 0;

    // Constructor
    public Alumno(..., ..., ..., ...) {
        cantidad++;
    }
    ...
    // Métodos públicos de clase: set/get
    public static void setCantidad(int cantidad) {
        Alumno.cantidad = cantidad;
    }

    public static int getCantidad() {
        return cantidad;
    }
    ...
}
```

```

protected void actionPerformedBtnProcesar(ActionEvent arg0) {
    Alumno a = new Alumno(12345, "Juan", 13, 15);
    Alumno b = new Alumno(67890, "Pedro", 12, 13);
    Alumno c = new Alumno(38214, "María", 14, 20);

    imprimir("# objetos creados : " + Alumno.getCantidad());
    imprimir("# objetos creados : " + a.getCantidad());
}

```

→ # de objetos creados : 3

1.3.3. Bloque de inicialización *static*

Lo conforman una serie de códigos que se ejecutan antes de cualquier llamado a la clase o algún objeto de la clase.

Se utiliza para inicializar variables, crear conexiones a base de datos o ejecutar cualquier código que sea prioritario antes de cualquier tipo de ejecución en la clase que se define.

Ejemplo:

```

public class Alumno {
    // Variable de clase privada
    private static int cantidad;

    // Bloque de inicialización
    static {
        cantidad = 0;
    }
    ...

    // Constructor
    public Alumno(..., ..., ..., ...) {
        ...
        cantidad++;
    }
    ...
}

```

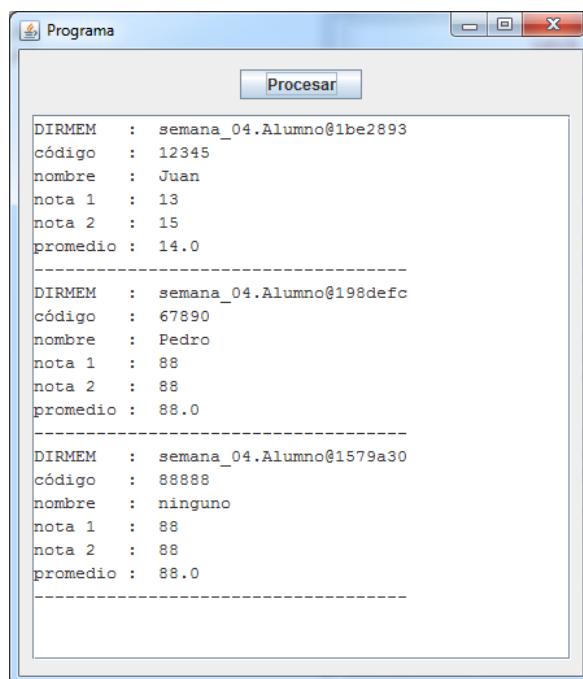
Toda clase se carga en memoria al crearse el primer objeto o al utilizar un atributo *static* o método *static*. Al cargarse la clase primero se ejecutan los inicializadores *static* y luego el constructor de la clase.

1.3.4. Sobrecarga

Consiste en crear varios constructores o métodos con el mismo nombre siempre y cuando no coincidan sus parámetros en orden de tipología. El riesgo de aplicar sobrecarga es que más de un atributo se pueda quedar con el valor de inicialización por defecto asumido por Java.

Ejemplo:

```
public class Alumno {  
    // Atributos privados  
    private int codigo, nota1, nota2;  
    private String nombre;  
    ...  
    // Constructores  
    public Alumno(int codigo, String nombre, int nota1, int nota2) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
        this.nota1 = nota1;  
        this.nota2 = nota2;  
    }  
  
    public Alumno(int codigo, String nombre) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
    }  
  
    public Alumno() {  
    }  
    ...  
}  
  
protected void actionPerformedBtnProcesar(ActionEvent arg0) {  
    Alumno a = new Alumno(12345, "Juan", 13, 15);  
    listado(a);  
  
    Alumno b = new Alumno(67890, "Pedro");  
    listado(b);  
  
    Alumno c = new Alumno();  
    listado(c);  
}
```



También se puede aplicar sobrecarga a los métodos. Ejemplo:

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

1.3.5. Uso del *this* en sobrecarga

También se puede utilizar la referencia *this* para hacer que un constructor invoque a otro constructor. Esto evita que algún atributo quede inicializado por defecto.

Ejemplo:

```
public class Alumno {  
    // Atributos privados  
    private int codigo, nota1, nota2;  
    private String nombre;  
    ...  
  
    // Constructores  
    public Alumno(int codigo, String nombre, int nota1, int nota2) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
        this.nota1 = nota1;  
        this.nota2 = nota2;  
    }  
  
    public Alumno(int codigo, String nombre) {  
        this(codigo, nombre, 88, 88);  
    }  
  
    public Alumno() {  
        this(88888, "ninguno");  
    }  
    ...  
}
```

1.3.6. Modificador *final* (constantes)

Se usa para hacer que un atributo *static* se convierta en constante para todos los objetos. Es decir, una vez que el atributo asume un valor no podrá ser modificado. Al anteponer *static final* a un atributo público, la variable única se convierte en *constante de Clase*.

Ejemplo:

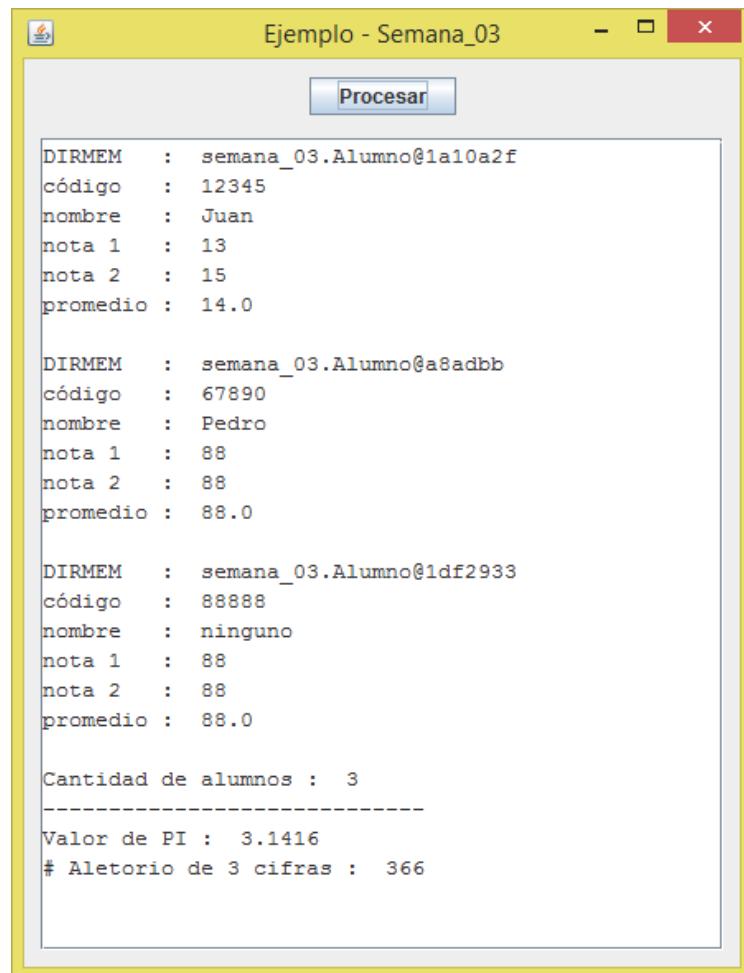
```
public class Libreria {  
    // Constante pública de clase  
    public static final double PI = 3.1416;  
    ...  
}  
  
protected void actionPerformedBtnProcesar(ActionEvent arg0) {  
    imprimir("Valor de PI : " + Libreria.PI);  
}  
→ Varlor de PI : 3.1416
```

1.3.7. Librería

Es una clase que implementa métodos **static** los cuales podrán ser invocados desde cualquier parte del programa a través de la clase sin necesidad de tener que crear un objeto.

Ejemplo:

```
public class Libreria {  
    // Constructor  
    public Libreria() {  
    }  
  
    // Métodos públicos de clase  
    public static int aleatorio(int min, int max) {  
        return (int)((max - min + 1) * Math.random()) + min;  
    }  
}  
  
protected void actionPerformedBtnProcesar(ActionEvent arg0) {  
    imprimir("# aleatorio de 3 cifras : " +  
            Libreria.aleatorio(100, 999));  
}  
→ # aleatorio de 3 cifras : 495
```



The screenshot shows a Windows application window titled "Ejemplo - Semana_03". The window contains a text area with the following content:

```
DIRMEM : semana_03.Alumno@1a10a2f
código : 12345
nombre : Juan
nota 1 : 13
nota 2 : 15
promedio : 14.0

DIRMEM : semana_03.Alumno@a8adbb
código : 67890
nombre : Pedro
nota 1 : 88
nota 2 : 88
promedio : 88.0

DIRMEM : semana_03.Alumno@1df2933
código : 88888
nombre : ninguno
nota 1 : 88
nota 2 : 88
promedio : 88.0

Cantidad de alumnos : 3
-----
Valor de PI : 3.1416
# Aleatorio de 3 cifras : 366
```

Ejercicios

Ejercicio Propuesto 1

Diseñe la clase **Factura** en el paquete **semana_03** con los atributos privados: ruc (*String*), empresa (*String*), unidades (*int*) y precio unitario (*double*).

Implemente además:

- Una variable privada de clase que cuente la cantidad de objetos tipo Factura creadas (*int*).
- Una variable privada de clase que acumule la suma de los importes facturados (*double*).
- Una constante pública de clase (*String*).
- Un bloque de inicialización *static* para asignarle a la constante el texto “Sunat” e inicializar con cero las variables privadas de clase.
- Un constructor que inicialice a todos los atributos, cuente la cantidad de objetos creados y acumule los importes facturados. Haga uso de la referencia *this*.
- Un constructor con dos parámetros que inicialice sólo los atributos ruc y empresa, invocando al primer constructor usando la referencia *this*, enviando con el valor 10 las unidades y con 50.0 el precio unitario.
- Un constructor sin parámetros que invoque al segundo constructor usando la referencia *this*, enviando con “1111111111” el ruc y con “MN-Global SRL” la empresa.
- Métodos de acceso público *set* para todos los atributos privados. Use de la referencia *this*.
- Métodos de acceso público *get* para todos los atributos privados.
- Métodos públicos de clase *set/get* para las variables privadas de clase.
- Un método que retorne el importe facturado (unidades * precio unitario).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice tres objetos de tipo Factura (con datos fijos) haciendo uso de los tres constructores.
- invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Factura y visualice sus datos completos.
- Muestre el valor de la constante, la cantidad de objetos creados y la suma de los importes facturados.

Ejercicio Propuesto 2

Diseñe la clase **Empleado** en el paquete **semana_03** con los atributos privados: codigo (*int*), nombre (*String*), categoria (*int*) y número de celular (*int*).

Implemente además:

- Una variable privada de clase que cuente la cantidad de objetos tipo Empleado creados (*int*).
- Una variable privada de clase que acumule la suma de los sueldos netos (*double*).
- Una constante pública de clase para un factor de descuento (*double*).
- Un bloque de inicialización *static* para asignarle a la constante el valor 0.15 e inicializar con cero las variables privadas de clase.
- Un constructor que inicialice a todos los atributos, cuente la cantidad de objetos creados y acumule los sueldos netos. Haga uso de la referencia *this*.
- Un constructor con dos parámetros que inicialice sólo los atributos codigo y nombre, invocando al primer constructor usando la referencia *this*, enviando con el valor 2 la categoría y con 999999999 el número de celular.

- Un constructor sin parámetros que invoque al segundo constructor usando la referencia *this*, enviando con 4444 el código y con “Ninguno” el nombre.
- Métodos de acceso público set para todos los atributos privados. Use la referencia *this*.
- Métodos de acceso público get para todos los atributos privados.
- Métodos públicos de clase set/get para las variables privadas de clase.
- Un método que retorne el sueldo bruto del empleado sabiendo que:

categoría	sueldo bruto
0	S/. 7200
1	S/. 6300
2	S/. 5100

- Un método que retorne el descuento (aplicado al sueldo bruto).
- Un método que retorne el sueldo neto (sueldo bruto – descuento).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice tres objetos de tipo Empleado (con datos fijos) haciendo uso de los tres constructores.
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Empleado y visualice sus datos completos.
- Muestre el factor de descuento, la cantidad de objetos creados y la suma de los sueldos netos.

Ejercicio Propuesto 3

Diseñe la clase **Movil** en el paquete **semana_03** con los atributos privados: número de móvil (*int*), nombre del cliente (*String*), segundos consumidos (*int*) y precio por segundo (*double*).

Implemente además:

- Una variable privada de clase que cuente la cantidad de objetos tipo Movil creados (*int*).
- Una variable privada de clase que acumule la suma de todos los importes a pagar (*double*).
- Una constante pública de clase para el factor del IGV (*double*).
- Un bloque de inicialización *static* para asignarle a la constante el valor 0.18 e inicializar con cero las variables privadas de clase.
- Un constructor que inicialice a todos los atributos, cuente la cantidad de objetos creados y acumule los importes a pagar. Haga uso de la referencia *this*.
- Un constructor con dos parámetros que inicialice sólo los atributos número y nombre, invocando al primer constructor usando la referencia *this*, enviando con el valor 75 los segundos consumidos y con 0.28 el precio por segundo.
- Un constructor sin parámetros que invoque al segundo constructor usando la referencia *this*, enviando con 987656789 el número de móvil y con “Juan” el nombre del cliente.
- Métodos de acceso público set para todos los atributos privados. Use de la referencia *this*.
- Métodos de acceso público get para todos los atributos privados.
- Métodos públicos de clase set/get para las variables privadas de clase.
- Un método que retorne el costo por consumo (segundos * precio).
- Un método que retorne el impuesto por IGV (aplicado al costo por consumo).
- Un método que retorne el importe a pagar (costo por consumo + impuesto por IGV).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice tres objetos de tipo Movil (con datos fijos) haciendo uso de los tres constructores.
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Movil y visualice sus datos completos.
- Muestre el valor del IGV, la cantidad de objetos creados y la suma de los importes a pagar.

Ejercicio Propuesto 4

Diseñe la clase **Asesor** en el paquete **semana_03** con los atributos privados: nombre del asesor (*String*), dni (*int*), codigo (*int*), y remuneración en dólares (*double*).

Implemente además:

- Una variable privada de clase que cuente la cantidad de objetos tipo Asesor creados (*int*).
- Una variable privada de clase que acumule la suma de todas las remuneraciones (*double*).
- Una constante pública de clase para el nombre de la institución (*String*).
- Un bloque de inicialización *static* para asignarle a la constante el nombre de la institución e inicializar con cero las variables privadas de clase.
- Un constructor que inicialice a todos los atributos, cuente la cantidad de objetos creados y acumule todas las remuneraciones. Haga uso de la referencia *this*.
- Un constructor con dos parámetros que inicialice sólo los atributos nombre y dni, invocando al primer constructor usando la referencia *this*, enviando con el valor 55555 el código y con 2000.0 la remuneración en dólares.
- Un constructor sin parámetros que invoque al segundo constructor usando la referencia *this*, enviando con "NN" el nombre del asesor y con 88888888 el número del dni.
- Métodos de acceso público set para todos los atributos privados. Use de la referencia *this*.
- Métodos de acceso público get para todos los atributos privados.
- Métodos públicos de clase set/get para las variables privadas de clase.

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice tres objetos de tipo Asesor (con datos fijos) haciendo uso de los tres constructores.
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Asesor y visualice sus datos completos.
- Muestre el nombre de la institución, la cantidad de objetos creados y la suma de todas las remuneraciones.

Ejercicio Propuesto 5

Diseñe la clase **Balon** en el paquete **semana_03** con los atributos privados: marca (*String*), peso en gramos (*double*), presión en libras (*double*), diámetro en centímetros (*double*) y precio (*double*).

Implemente además:

- Una variable privada de clase que cuente la cantidad de objetos tipo Balon creados (*int*).
- Una variable privada de clase que acumule los importes a pagar (*double*).
- Una constante pública de clase para el valor de PI (*double*) y otra para el factor de descuento (*double*).

- Un bloque de inicialización *static* para asignarle a la constante PI el valor 3.1416, a la segunda constante el factor 0.05 e inicializar con cero las variables privadas de clase.
- Un constructor que inicialice a todos los atributos, cuente la cantidad de objetos creados y acumule los importes a pagar. Haga uso de la referencia *this*.
- Un constructor con tres parámetros que inicialice sólo los atributos marca, peso y presión, invocando al primer constructor usando la referencia *this*, enviando con el valor 18.5 el diámetro y con 100.0 el precio.
- Un constructor sin parámetros que invoque al segundo constructor usando la referencia *this*, enviando con “Adidas” la marca, con 1.5 el peso y con 4.8 la presión.
- Métodos de acceso público *set* para todos los atributos privados. Use la referencia *this*.
- Métodos de acceso público *get* para todos los atributos privados.
- Métodos públicos de clase *set/get* para las variables privadas de clase.
- Un método que retorne el radio (diámetro / 2).
- Un método que retorne el volumen del balón ($4 * PI * radio * radio * radio / 3$).
- Un método que retorne el descuento (aplicado al precio).
- Un método que retorne el importe a pagar (precio - descuento).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice tres objetos de tipo Balon (con datos fijos) haciendo uso de los tres constructores.
- invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Balon y visualice sus datos completos.
- Muestre el valor de PI, el factor de descuento, la cantidad de objetos creados y el importe a pagar acumulado.

Ejercicio Propuesto 6

Diseñe la clase **Consultor** en el paquete **semana_03** con los atributos privados: *codigo* (*int*), *nombre* (*String*), *horas trabajadas* (*int*) y *tarifa por hora* (*double*).

Implemente además:

- Una variable privada de clase que cuente la cantidad de objetos tipo Consultor creados (*int*).
- Una variable privada de clase que acumule los sueldos netos (*double*).
- Dos constantes públicas de clase para factores de descuentos por AFP y EPS (*double*).
- Un bloque de inicialización *static* para asignarle a la constante AFP el factor 0.10, a la constante EPS el factor 0.05 e inicializar con cero las variables privadas de clase.
- Un constructor que inicialice a todos los atributos, cuente la cantidad de objetos creados y acumule los sueldos netos. Haga uso de la referencia *this*.
- Un constructor con tres parámetros que inicialice sólo los atributos código, nombre y horas trabajadas, invocando al primer constructor usando la referencia *this*, enviando con el valor 65.0 la tarifa.
- Un constructor sin parámetros que invoque al segundo constructor usando la referencia *this*, enviando con 333 el código, con “Juan” el nombre y con 30 las horas.
- Métodos de acceso público *set* para todos los atributos privados. Use la referencia *this*.
- Métodos de acceso público *get* para todos los atributos privados.
- Métodos públicos de clase *set/get* para las variables privadas de clase.
- Un método que retorne el sueldoBruto (*horas*tarifa*).
- Un método que retorne el descuentoAFP (aplicado al sueldo bruto).
- Un método que retorne el descuentoEPS (aplicado al sueldo bruto).
- Un método que retorne el sueldoNeto = (sueldoBruto – descuentoAFP – descuentoEPS).

En la clase principal, a la pulsación del botón Procesar:

- Declare, cree e inicialice tres objetos de tipo Consultor (con datos fijos) haciendo uso de los tres constructores.
- Invoque a un método listado que reciba (como parámetro) la referencia a un objeto de tipo Consultor y visualice sus datos completos.
- Muestre el valor de los factores de descuentos por AFP y EPS, la cantidad de objetos creados y el acumulado de los sueldos netos.

1.4. Clase String

1.4.1. Descripción

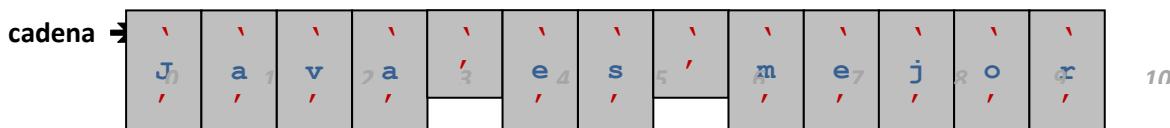
La Clase **String** cuenta con métodos para manipular cadenas de texto.

Una cadena de texto es un conjunto de caracteres dispuestos uno a continuación de otro, donde cada carácter conserva su propio espacio (tamaño en bytes).

Internamente, la Clase *String* ubica a cada carácter en un espacio independiente y enumera de izquierda a derecha las posiciones, empezando de cero.

Ejemplo:

```
String cadena = "Java es mejor";
```



1.4.2. Métodos básicos de la clase String

1) `public int length() {`
 `}`

→ Retorna la cantidad de caracteres de la cadena

Ejemplo:

```
int longitud;
longitud = cadena.length(); // longitud = 13
```

2) `public char charAt(int) {`
 `}`

→ Retorna una copia del carácter ubicado en la posición indicada

Ejemplo:

```
char primerCaracter;
primerCaracter = cadena.charAt(0); // primerCaracter = 'J'

char ultimoCaracter;
ultimoCaracter = cadena.charAt(longitud - 1); // ultimoCaracter = 'r'
```

```
3) public boolean equals(String) {
    }
```

→ Retorna true o false si la cadena que invoca al método coincide en texto con la cadena enviada como parámetro

Ejemplo:

```
String cad1 = "Java es lo máximo";
boolean sino1 = cadena.equals(cad1); // sino1 = false

String cad2 = "Java es mejor";
boolean sino2 = cadena.equals(cad2); // sino2 = true

String cad3 = "Java gusta a todos";
boolean sino3 = cadena.equals(cad3); // sino3 = false

String cad4 = "Java es mejor hoy";
boolean sino4 = cadena.equals(cad4); // sino4 = false
```

```
4) public int compareTo(String) {
    }
```

→ Retorna un número entero (luego de comparar alfabéticamente la cadena que invoca al método con la cadena enviada como parámetro)

Ejemplo:

```
cadena → "Java es mejor";
cad1 → "Java es lo máximo";
int ok1 = cadena.compareTo(cad1); // ok1 = 1
Nota: el valor de ok1 es la diferencia ASCII de los caracteres m y l
```

```
cadena → "Java es mejor";
cad2 → "Java es mejor";
int ok2 = cadena.compareTo(cad2); // ok2 = 0
Nota: el valor de ok2 es la diferencia de longitudes de cadena y cad2
```

```
cadena → "Java es mejor";
cad3 → "Java gusta a todos";
int ok3 = cadena.compareTo(cad3); // ok3 = -2
Nota: el valor de ok3 es la diferencia ASCII de los caracteres e y g
```

```
cadena → "Java es mejor";
cad4 → "Java es mejor hoy";
int ok4 = cadena.compareTo(cad4); // ok4 = -4
Nota: el valor de ok4 es la diferencia de longitudes de cadena y cad4
```

1.4.3. Concatenación

Se utiliza el símbolo más para acoplar información. Una variable cadena puede concatenar textos, caracteres y números. Al final todo sigue siendo un String.

Ejemplo:

```
String cad5 = "Ciber" + "Java" + '_' + 2019;
```

El resultado es: **"CiberJava_2019"**

cad5 →	'C'	'i'	'b'	'e'	'r'	'J'	'a'	'v'	'a'	'_'	'2'	'0'	'1'	'9'
	0	1	2	3	4	5	6	7	8	9	10	11	12	13

1.4.4. Recorrido

Consiste en contar las posiciones de izquierda a derecha o viceversa. Ejemplo:

cadena →	'J'	'a'	'v'	'a'	' '	'e'	's'	' '	'm'	'e'	'j'	'o'	'r'
i →	0	1	2	3	4	5	6	7	8	9	10	11	12

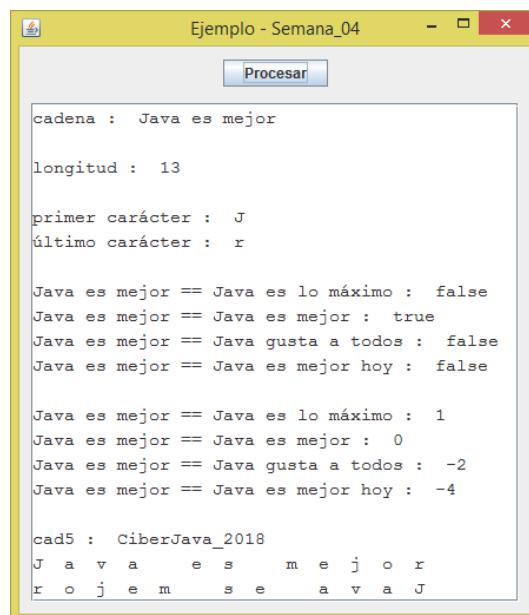
EN ASCENSO (de izquierda a derecha)

```
for (int i=0; i<cadena.length(); i++) {
    ...
}
```


cadena →	'J'	'a'	'v'	'a'	' '	'e'	's'	' '	'm'	'e'	'j'	'o'	'r'
	0	1	2	3	4	5	6	7	8	9	10	11	12

EN DESCENSO (de derecha a izquierda)

```
for (int i=cadena.length()-1; i>=0; i--) {
    ...
}
```



1.4.4.1. Todos los métodos de la clase String

```
String cadena = "Java es mejor";
```

METODO	DESCRIPCION
length()	Devuelve la longitud de la cadena. <code>int longitud = cadena.length();</code> <code>longitud ← 13</code>
charAt(int)	Devuelve una copia del carácter que encuentre en la posición indicada por el parámetro. <code>char caracter = cadena.charAt(8);</code> <code>caracter ← 'm'</code>
equals(String)	Comprueba si dos cadenas son iguales. En este caso comprueba que el objeto dado como argumento sea de tipo <i>String</i> y contenga la misma cadena de caracteres que el objeto actual. <code>String s = "Java";</code> <code>boolean b = cadena.equals(s);</code> <code>b ← false</code>
compareTo(String)	Devuelve un entero menor que cero si la cadena es alfabéticamente menor que la dada como argumento, cero si las dos cadenas son léxicamente iguales y un entero mayor que cero si la cadena es mayor alfabéticamente. <code>String s1 = "Java es lo máximo",</code> <code>s2 = "Java es mejor",</code> <code>s3 = "Java es ok";</code> <code>int i = cadena.compareTo(s1),</code> <code>j = cadena.compareTo(s2),</code> <code>k = cadena.compareTo(s3);</code> <code>i ← 1 // cadena mayor que s1 alfabéticamente</code> <code>j ← 0 // cadena contiene lo mismo que s2</code> <code>k ← -2 // cadena menor que s3 alfabéticamente</code>
equalsIgnoreCase(String)	Realiza la misma tarea que <i>equals</i> pero sin tener en cuenta las mayúsculas o minúsculas. <code>String s = "Java es mejor";</code> <code>boolean b = cadena.equalsIgnoreCase(s);</code> <code>b ← true</code>
startsWith(String)	Comprueba si el comienzo de la cadena actual coincide con la cadena pasada como parámetro. <code>String s = "JavaX";</code> <code>boolean b = cadena.startsWith(s);</code> <code>b ← false</code>
endsWith(String)	Comprueba si el final de la cadena actual coincide con la cadena pasada como parámetro. <code>String s = "Xmejor";</code> <code>boolean b = cadena.endsWith(s);</code> <code>b ← true</code>
indexOf(char)	Devuelve la posición que por primera vez aparece el carácter (expresado como entero) pasado como parámetro. En caso no exista devuelve -1. <code>int i = cadena.indexOf('e');</code> <code>i ← 5</code>

indexOf(char, int)	Devuelve la posición que por primera vez aparece el carácter (expresado como entero) a partir de la posición especificada como segundo parámetro. <code>int i = cadena.indexOf('e', 6); i ← 9</code>
indexOf(String)	Devuelve la posición que por primera vez aparece la cadena pasada como parámetro. <code>int i = cadena.indexOf("va"); i ← 2</code>
indexOf(String, int)	Devuelve la posición que por primera vez aparece la cadena pasada como parámetro, pudiendo especificar en un segundo parámetro a partir de dónde buscar. <code>int i = cadena.indexOf("ej", 5); i ← 9</code>
lastIndexOf(char) lastIndexOf(char, int) lastIndexOf(String) lastIndexOf(String, int)	Devuelve la última vez que aparece el carácter (expresado como entero) o cadena pasada como parámetro, pudiendo especificar en un segundo parámetro, a partir de dónde buscar (búsqueda hacia atrás). <code>String s = "e"; int i = cadena.lastIndexOf(s); i ← 9</code>
toLowerCase()	Convierte la cadena a minúsculas. <code>String s = "CiberJava – Lima - Perú"; s = s.toLowerCase(); s ← "ciberjava – lima – perú"</code>
toUpperCase()	Convierte la cadena a mayúsculas. <code>String s = "CiberJava – Lima - Perú"; s = s.toUpperCase(); s ← "CIBERJAVA – LIMA – PERÚ"</code>
trim()	Elimina espacios al principio y al final de la cadena. <code>String s = " CiberJava Lima "; s = s.trim(); s ← "CiberJava Lima"</code>
substring(int) substring(int, int)	Devuelve una subcadena de la cadena actual, empezando por el primer índice indicado hasta antes del segundo índice (si se especifica) o hasta el final de la cadena. <code>String s1 = "viva el Perú", s2 = s1.substring(5), s3 = s1.substring(3, 9); s2 ← "el Perú" s3 ← "a el P"</code>
replace(char, char)	Reemplaza todos los caracteres iguales al primer parámetro y los sustituye por el carácter que pasamos en segundo lugar, teniendo en cuenta lo mismo una mayúscula que una minúscula. <code>String s = "biba el Perú"; s = s.replace('b', 'v'); s ← "viva el Perú"</code>

split(String)	Busca un tope en una cadena y distribuye una copia de las subcadenas en un arreglo lineal de cadenas. <code>String linea = "123;Ana;20;55.0"; String[] s; s = linea.split(","); s[0] ← "123" s[1] ← "Ana" s[2] ← "20" s[3] ← "55.0"</code>
toCharArray()	Convierte la cadena a un vector de caracteres. <code>char[] arreglo = cadena.toCharArray();</code>
Métodos estáticos de conversión	La clase <i>String</i> dispone de métodos para transformar valores de otros tipos de datos a cadena. Todos se llaman <i>valueOf</i> y son estáticos.
<i>String.valueOf(boolean)</i>	
<i>String.valueOf(int)</i>	
<i>String.valueOf(long)</i>	
<i>String.valueOf(float)</i>	
<i>String.valueOf(double)</i>	
<i>String.valueOf(Object)</i>	
<i>String.valueOf(char[])</i>	
<i>String.valueOf(char[], int, int)</i>	Transforma una subcadena de un arreglo de caracteres, especificando una posición y la longitud. <code>char[] c = {'C','i','b','e','r','J','a','v','a'}; String s = String.valueOf(c, 3, 5); s ← "erJav"</code>

```
package libreria;

public class Formato {

    // Métodos que retornan valor (con parámetros)
    public static String estilo(double num) {
        String cadena = String.format("%.2f", num);
        String[] s = cadena.split(",");
        return estilo(Long.parseLong(s[0])) + "." + s[1];
    }

    public static String estilo(long num) {
        String cad = "";
        int x = 0;

        while (num > 0) {
            x++;
            if (num < 1000)
                cad = num%1000 + cad;
            else
                cad = String.format("%03d", num%1000) + cad;

            num /= 1000;

            if (num > 0)
                if (x == 2)
                    cad = "1" + cad;
                else
                    if (x == 4)
                        cad = "2" + cad;
                    else
                        if (x == 6)
                            cad = "3" + cad;
                        else
                            cad = "," + cad;
        }

        return cad;
    }

    public static String romano(int num) {
        byte unidad, decena, centena, millar;
        String uu = "", dd = "", cc = "", mm = "";

        if (num > 0 && num < 4000) {
            unidad = (byte)(num % 10);
            num /= 10;
            decena = (byte)(num % 10);
            num /= 10;
            centena = (byte)(num % 10);
            millar = (byte)(num / 10);

            switch(millar) {
                case 1: mm = "M"; break;
                case 2: mm = "MM"; break;
                case 3: mm = "MMM"; break;
            }
        }
    }
}
```

```

        switch(centena) {
            case 1: cc = "C"; break;
            case 2: cc = "CC"; break;
            case 3: cc = "CCC"; break;
            case 4: cc = "CD"; break;
            case 5: cc = "D"; break;
            case 6: cc = "DC"; break;
            case 7: cc = "DCC"; break;
            case 8: cc = "DCCC"; break;
            case 9: cc = "CM"; break;
        }

        switch(decena) {
            case 1: dd = "X"; break;
            case 2: dd = "XX"; break;
            case 3: dd = "XXX"; break;
            case 4: dd = "XL"; break;
            case 5: dd = "L"; break;
            case 6: dd = "LX"; break;
            case 7: dd = "LXX"; break;
            case 8: dd = "LXXX"; break;
            case 9: dd = "XC"; break;
        }

        switch(unidad) {
            case 1: uu = "I"; break;
            case 2: uu = "II"; break;
            case 3: uu = "III"; break;
            case 4: uu = "IV"; break;
            case 5: uu = "V"; break;
            case 6: uu = "VI"; break;
            case 7: uu = "VII"; break;
            case 8: uu = "VIII"; break;
            case 9: uu = "IX"; break;
        }

        return mm + cc + dd + uu;
    } else
        return "fuera de rango";
}

public static String texto(double num) {
    String cadena = String.format("%.2f", num);
    String[] s = cadena.split(",");
    return texto(Long.parseLong(s[0])) + " con " + s[1] + "/100";
}

public static String texto(long num) {
    String cad = "", c;
    byte x = 0;
    int n;

    while (num > 0) {
        x++;
        n = (int)(num%1000);
        c = "";
        if (n > 0) {
            if (n < 10) c = "00" + n;
            else if (n < 100) c = "0" + n;
            else c = n;
        }
        cad = c + cad;
        num = num / 1000;
    }
    return cad;
}

```

```

switch(x) {
    case 2:
    case 4:
    case 6:
    case 8: if (n > 0)
        c = " mil";
        break;
    case 3: if (num%1000000 > 0)
        if (num == 1)
            c = " millón";
        else
            c = " millones";
        break;
    case 5: if (num%1000000 > 0)
        if (num == 1)
            c = " billón";
        else
            c = " billones";
        break;
    case 7: if (num == 1)
        c = " trillón";
        else
            c = " trillones";
        break;
    }
}

cad = Letras(n) + c + cad;

if (x == 1 && num%10 == 1)
    cad += "o";

num /= 1000;

if (n > 0 && num > 0)
    cad = " " + cad;
}

return cad;
}

private static String Letras(int num) {
    byte centena, decena, unidad;
    String cadena, cc, dd, uu;
    centena = (byte)(num / 100);
    num %= 100;
    decena = (byte)(num / 10);
    unidad = (byte)(num % 10);

    cc = "";
    switch(centena) {
        case 1: cc = "cien";
            if (decena != 0 || unidad != 0)
                cc += "to"; break;
        case 2: cc = "doscientos"; break;
        case 3: cc = "trescientos"; break;
        case 4: cc = "cuatrocientos"; break;
        case 5: cc = "quinientos"; break;
        case 6: cc = "seiscientos"; break;
        case 7: cc = "seiscientos"; break;
        case 8: cc = "seiscientos"; break;
        case 9: cc = "seiscientos"; break;
    }
}

```

```
        case 7: cc = "setecientos"; break;
        case 8: cc = "ochocientos"; break;
        case 9: cc = "novecientos"; break;
    }

    dd = "";
    switch(decena) {
        case 1: switch(unidad) {
            case 0: dd = "diez"; break;
            case 1: dd = "once"; break;
            case 2: dd = "doce"; break;
            case 3: dd = "trece"; break;
            case 4: dd = "catorce"; break;
            case 5: dd = "quince"; break;
            default: dd = "dieci";
        } break;
        case 2: dd = "veint";
            if (unidad == 0)
                dd += "e";
            else
                dd += "i"; break;
        case 3: dd = "treint"; break;
        case 4: dd = "cuarent"; break;
        case 5: dd = "cincuent"; break;
        case 6: dd = "sesent"; break;
        case 7: dd = "setent"; break;
        case 8: dd = "ochent"; break;
        case 9: dd = "novent"; break;
    }

    if (decena > 2) {
        dd += "a";
        if (unidad > 0)
            dd += " y ";
    }

    uu = "";
    if (decena != 1 || unidad > 5)
        switch(unidad) {
            case 1: uu = "un"; break;
            case 2: uu = "dos"; break;
            case 3: uu = "tres"; break;
            case 4: uu = "cuatro"; break;
            case 5: uu = "cinco"; break;
            case 6: uu = "seis"; break;
            case 7: uu = "siete"; break;
            case 8: uu = "ocho"; break;
            case 9: uu = "nueve"; break;
        }

    if (decena == 0 && unidad == 0)
        cadena = cc;
    else if (centena == 0)
        cadena = cc + dd + uu;
    else
        cadena = cc + " " + dd + uu;
    return cadena;
}
```

```
package gui;

import libreria.Formato;

import java.awt.EventQueue;
import java.awt.Font;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.EmptyBorder;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Conversiones extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;

    private JButton btnProcesar;
    private JPanel contentPane;
    private JScrollPane scrollPane;
    private JTextArea txtS;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Conversiones frame = new Conversiones();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Conversiones() {
        setResizable(false);
        setTitle("Conversiones");
        setIconImage(new ImageIcon("imagenes/PrimaTaxi.png").getImage());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(1000, 370);
        this.setLocationRelativeTo(null);

        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);
    }
}
```

```
btnProcesar = new JButton("procesar");
btnProcesar.addActionListener(this);
btnProcesar.setBounds(445, 10, 100, 23);
getContentPane().add(btnProcesar);

scrollPane = new JScrollPane();
scrollPane.setBounds(10, 45, 975, 285);
contentPane.add(scrollPane);

txtS = new JTextArea();
txtS.setFont(new Font("Monospaced", Font.PLAIN, 16));
scrollPane.setViewportView(txtS);

procesar();
}

public void actionPerformed(ActionEvent arg0) {
    if (arg0.getSource() == btnProcesar) {
        actionPerformedBtnProcesar(arg0);
    }
}

protected void actionPerformedBtnProcesar(ActionEvent arg0) {
    procesar();
}

// Métodos tipo void (sin parámetros)
void procesar() {
    txtS.setText("");
    int entero;
    entero = aleatorio(1, 3999);
    imprimir("Entero: " + entero + "\tEstilo: " +
             Formato.estilo(entero) + "\tRomano: " +
             Formato.romano(entero));
    imprimir(Formato.texto(entero));
    imprimir("");

    long enteroExtenso;
    enteroExtenso = aleatorio(0, 89999999999999999999L);
    imprimir("Entero extenso: " + enteroExtenso + "\t" +
             "Estilo: " + Formato.estilo(enteroExtenso));
    imprimir(Formato.texto(enteroExtenso));
    imprimir("");

    double real;
    real = aleatorio(0, 8999999999L) / 10000.0;
    imprimir("Real: " + real + "\t" + "Estilo: " +
             Formato.estilo(real));
    imprimir(Formato.texto(real));
    imprimir("");

    double realExtenso;
    realExtenso = aleatorio(0, 89999999999999999999L) / 10000.0;
    imprimir("Real extenso: " + realExtenso + "\t" + "Estilo: " +
             Formato.estilo(realExtenso));
    imprimir(Formato.texto(realExtenso));
}
```

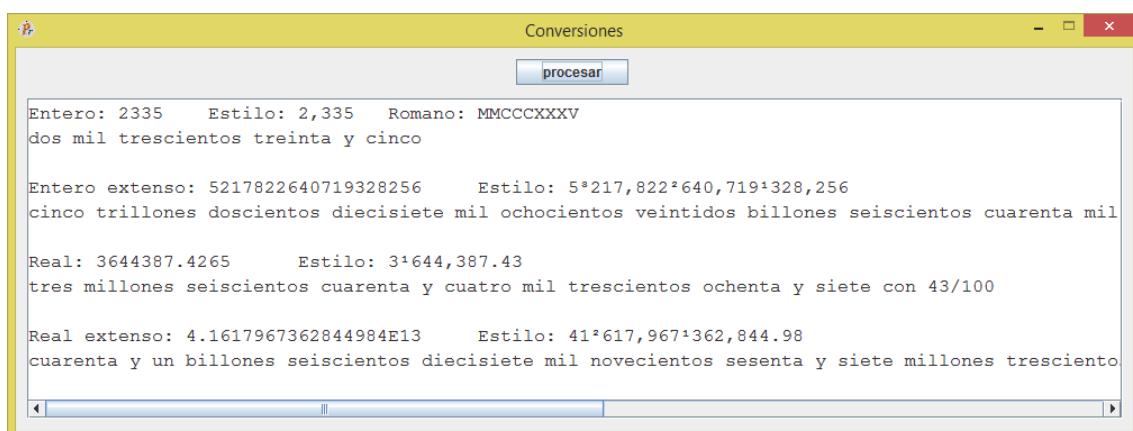
```
void imprimir() {
    txtS.append("");
}

// Métodos tipo void (con parámetros)
void imprimir(String s) {
    txtS.append(s + "\n");
}

// Métodos que retornan valor (con parámetros)
int aleatorio(int min, int max) {
    return (int)((max - min + 1) * Math.random()) + min;
}

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

}
```





ARREGLO LINEAL

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, los alumnos manipulan arreglos lineales con tipos de datos primitivos.

TEMARIO

2.1. Tema 5 : Conceptos y operaciones simples

- 2.1.1 : Descripción
- 2.1.2 : Declaración e inicialización
- 2.1.3 : Declaración privada e inicialización
- 2.1.4 : Recorrido
- 2.1.5 : Reemplazo
- 2.1.6 : Operaciones públicas básicas
- 2.1.7 : Operaciones públicas complementarias

2.2. Tema 6 : Artificios y operaciones variadas

- 2.2.1 : Descripción
- 2.2.2 : Declaración, creación y reserva
- 2.2.3 : Declaración privada, creación y reserva
- 2.2.4 : Ingreso personalizado
- 2.2.5 : Recorrido
- 2.2.6 : Redimensionamiento
- 2.2.7 : Método privado ampliarArreglo
- 2.2.8 : Operaciones públicas básicas
- 2.2.9 : Operaciones públicas complementarias

2.3. Tema 7 : Artificios y operaciones especiales

- 2.3.1 : Método privado Buscar
- 2.3.2 : Método privado Intercambiar
- 2.3.3 : Operaciones públicas complementarias

Evaluación Laboratorio EL2

2.4. Tema 8 : Artificios complementarios

- 2.4.1 : Método privado Eliminar
- 2.4.2 : Método privado Insertar
- 2.4.3 : Operaciones públicas complementarias

ACTIVIDADES PROPUESTAS

- Reconocer un arreglo lineal.
- Emplear arreglos lineales en diversas aplicaciones.

2.1. Conceptos y Operaciones Simples

2.1.1. Descripción

Un arreglo lineal o unidimensional 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.

Un arreglo permite almacenar diferentes valores pero del mismo tipo de dato.

El tipo más simple de un arreglo es el *lineal* o *unidimensional*.



2.1.2. Declaración e inicialización

Con la declaración sólo se define la variable que hace referencia al arreglo y su contenido es *null*. Se puede inicializar un arreglo directamente siempre y cuando se haga al momento de su declaración.

Los símbolos [] indican a Java que la variable es de tipo arreglo lineal. Es decir que representa a un conjunto de datos. Pueden adherirse al tipo de dato o la variable. Entre los símbolos { } y separados por comas se indica los valores de asignación.

Forma 1:

```
tipo[] arreglo = { valores asignados };
```

Ejemplo:

```
int[] n = { 25, 27, 22, 24, 29, 20, 23 };
```

Forma 2:

```
tipo arreglo[] = { valores asignados };
```

Ejemplo:

```
int n[] = { 25, 27, 22, 24, 29, 20, 23 };
```

2.1.3. Declaración privada e inicialización

```
private int[] n = {25, 27, 22, 24, 29, 20, 23};
```

Gráficamente:



Java enumera internamente las posiciones a partir de 0

A esta posición la denominaremos genéricamente índice i

Los siete elementos del arreglo son:

```
n[0], n[1], n[2], n[3], n[4], n[5], n[6]
```

n[7] no está definido en el arreglo

`n.length` devuelve el tamaño del arreglo (*en este caso: 7*)

El contador i se encuentra en el rango: $0 \leq i < n.length$

2.1.4. Recorrido

Consiste en contar las posiciones de izquierda a derecha o viceversa.



EN ASCENSO (*de izquierda a derecha*)

```
for (int i=0; i<n.length; i++) {
```

...

}



EN DESCENSO (*de derecha a izquierda*)

```
for (int i=n.length-1; i>=0; i--) {
```

...

}

2.1.5. Remplazo

Consiste en modificar el contenido de un valor por otro valor. Se realiza a través de la posición.



n[i] = valor;

Ejemplo:

n[2] = 88;



2.1.6. Operaciones públicas básicas

Son aquellos métodos que permiten acceder al arreglo desde el exterior de la clase que lo contiene.

Ejemplo: método que retorna la cantidad de elementos

```
public int tamaño() {
    return n.length;
}
```

Ejemplo: método que recibe una posición y retorna una copia del número que allí se ubica

```
public int obtener(int i) {
    return n[i];
}
```

2.1.7. Operaciones públicas complementarias

Son los métodos que realizan operaciones adicionales.

Ejemplo: método que retorna la suma de todos los números del arreglo

```
public int sumaNumeros() {
    int suma = 0;
    for (int i=0; i<tamaño(); i++)
        suma += n[i];
    return suma;
}
```

Ejemplo: método que retorna la posición del primer número menor a 25. En caso no existe retorna -1

```
public double posPrimerNumeroMenorA25() {
    for (int i=0; i<tamaño(); i++)
        if (n[i] < 25)
            return i;
    return -1;
}
```

Ejemplo: método que remplaza a todos los números del arreglo por otros aleatorios de 3 cifras

```
public void generar() {
    for (int i=0; i<tamaño(); i++)
        n[i] = aleatorio(100, 999);
}
private int aleatorio(int max, int min) {
    return (int)((max - min + 1) * Math.random()) + min;
}
```

Ejercicio 11: Implemente la clase Arreglo en el paquete semana_05 con el atributo privado:

- Arreglo lineal **n** de tipo *int* con los números asignados: 25, 27, 22, 24, 29, 20, 23

Implemente como públicos:

- Un constructor que no haga nada
- Un método público **tamaño** que retorne la capacidad máxima del arreglo **n**.
- Un método público **obtener** que reciba la posición y retorne el número registrado en dicha posición.
- Un método público **sumaNumeros** que retorne la suma de los números.
- Un método público **posPrimerNumeroMenorA25** que retorne la posición del primer número menor a 25. En caso no exista retorne -1.
- Un método público **generar** que reemplace los números actuales por otros aleatorios de 3 cifras.
- Un método privado **aleatorio** que reciba como parámetros los valores enteros mínimo y máximo. Retorne luego un número al azar comprendido en ese intervalo cerrado.

```
package semana_05;

public class Arreglo {
    // Atributo privado
    private int[] n = {25, 27, 22, 24, 29, 20, 23};

    // Constructor
    public Arreglo() { }

    // Operaciones públicas
    public int tamaño() {
        return n.length;
    }

    public int obtener(int i) {
        return n[i];
    }

    public int sumaNumeros() {
        int suma = 0;
        for (int i=0; i<tamaño(); i++)
            suma += n[i];
        return suma;
    }

    public int posPrimerNumeroMenorA25() {
        for (int i=0; i<tamaño(); i++)
            if (n[i] < 25)
                return i;

        return -1;
    }
}
```

```

public void generar() {
    for (int i=0; i<tamaño(); i++)
        n[i] = aleatorio(100, 999);
}

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

```

Ejercicio 12: En el programa principal declare y cree como variable global un objeto de tipo Arreglo y a la pulsación del botón respectivo:

- Visualice los números del arreglo.
- Muestre la capacidad máxima del arreglo, suma y promedio de los números.
- Reemplace los números por otros aleatorios de tres cifras.

```

package gui;

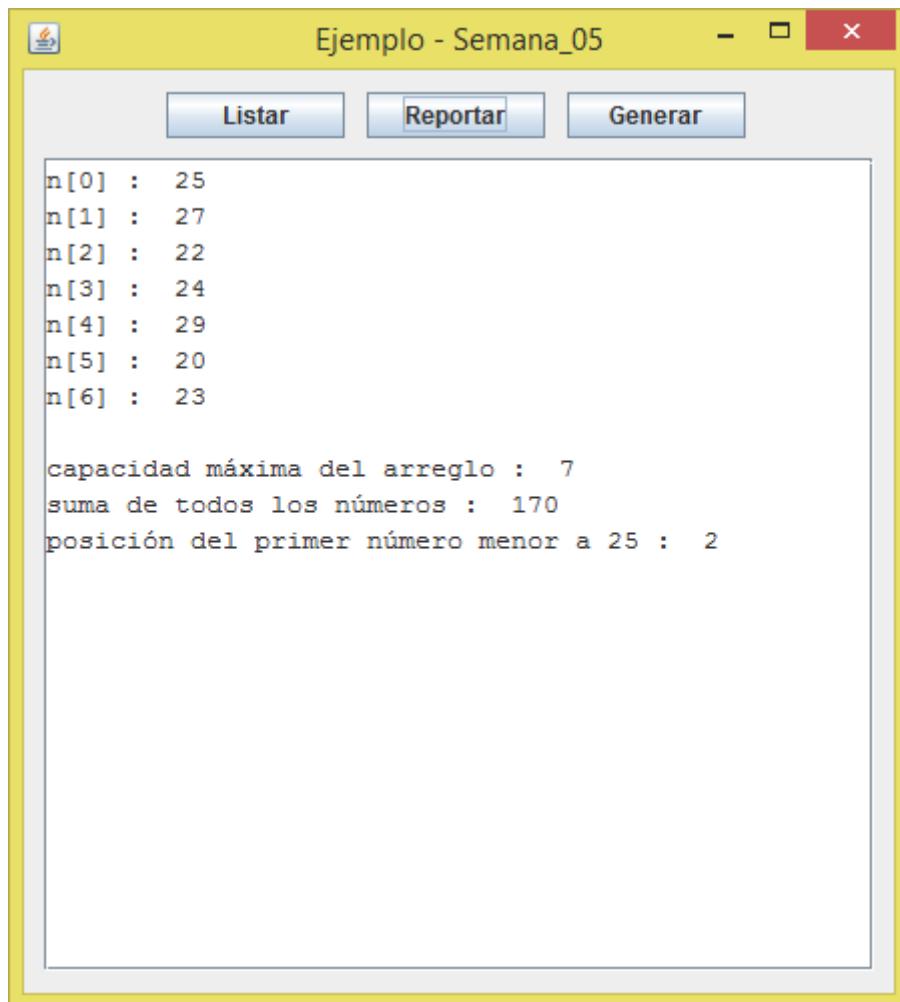
import semana_05.Arreglo;
...
public class Ejemplo extends JFrame implements ActionListener {
    // Declaración global
    Arreglo a = new Arreglo();

    ...
    protected void actionPerformedBtnListar(ActionEvent arg0) {
        /*
         * Visualiza los números del arreglo
         */
        txtS.setText("");
        for (int i=0; i<a.tamaño(); i++)
            imprimir("n[" + i + "] : " + a.obtener(i));
    }

    protected void actionPerformedBtnReportar(ActionEvent arg0) {
        /*
         * Muestra un análisis del arreglo
         */
        txtS.setText("");
        imprimir("capacidad máxima : " + a.tamaño());
        imprimir("suma de los números : " + a.sumaNumeros());
        imprimir("posición del primer número menor a 25 : " +
a.posPrimerNumeroMenorA25());
    }

    protected void actionPerformedBtnGenerar(ActionEvent arg0) {
    /*
     * Reemplaza los números por otros aleatorios de tres cifras
     */
        a.generar();
        txtS.setText("Los números del arreglo han sido cambiados.");
    }
}

```



Ejercicios

Ejercicio Propuesto 1

Diseñe la clase **ArregloEdades** en el paquete **semana_05** con el atributo privado **edad** (*int*) de tipo arreglo lineal y con los valores de inicialización:

27, 22, 13, 12, 25, 11, 29, 70, 15, 20

Implemente además:

- Un Constructor que no haga nada.
- Un método **tamaño** que retorne la cantidad de elementos del arreglo **edad**.
- Un método **obtener** que reciba una posición y retorne la edad registrada en dicha posición.
- Un método **edadPromedio** que retorne el promedio de todas las edades.
- Un método **edadMayor** que retorne la mayor de todas las edades.
- Un método **edadMenor** que retorne la menor de todas las edades.
- Un método **cantMayoresEdad** que retorne la cantidad de personas mayores de edad.
- Un método **cantMenoresEdad** que retorne la cantidad de personas menores de edad.
- Un método **buscarPrimeraEdadAdolescente** que retorne la posición de la primera edad encontrada en el rango de 12 a 20 años. En caso no exista retorne -1.
- Un método **buscarUltimaEdadAdolescente** que retorne la posición de la última edad encontrada en el rango de 12 a 20 años. En caso no exista retorne -1.
- Un método **generarEdades** que reemplace las edades actuales por otras aleatorias comprendidas en el rango de 10 a 90 años.
- Haga uso del método:

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

En la clase principal:

- Declare y cree el objeto global **ae** de tipo **ArregloEdades**.
- A la pulsación del botón **Listar** visualice las edades del arreglo.
- A la pulsación del botón **Reportar** visualice: cantidad de edades, edad promedio, edad mayor, edad menor, cantidad de personas mayores de edad, cantidad de personas menores de edad, posición de la primera edad adolescente y posición de la última edad adolescente.
- A la pulsación del botón **Generar** invoque al método **generarEdades**.

Ejercicio Propuesto 2

Diseñe la clase **ArregloNotas** en el paquete **semana_05** con el atributo privado **nota** (*int*) de tipo arreglo lineal y con los valores de inicialización:

11, 10, 16, 18, 15, 13, 20, 12, 19, 17

Implemente además:

- Un Constructor que no haga nada.
- Un método **tamaño** que retorne la cantidad de elementos del arreglo **nota**.
- Un método **obtener** que reciba una posición y retorne la nota registrada en dicha posición.
- Un método **notaPromedio** que retorne el promedio de todas las notas.
- Un método **notaMayor** que retorne la mayor de todas las notas.
- Un método **notaMenor** que retorne la menor de todas las notas.

- Un método **cantNotasAprobatorias** que retorne la cantidad de alumnos que obtuvieron de 13 a más.
- Un método **cantNotasDesaprobatorias** que retorne la cantidad de alumnos que obtuvieron menos de 13.
- Un método **cantNotasMayoresA15** que retorne la cantidad de alumnos que obtuvieron más de 15.
- Un método **buscarPrimeraNotaAprobatoria** que retorne la posición de la primera nota mayor o igual a 13. En caso no exista retorne -1.
- Un método **buscarUltimaNotaDesaprobatoria** que retorne la posición de la última nota menor a 13. En caso no exista retorne -1.
- Un método **generarNotas** que reemplace las notas actuales por otras aleatorias comprendidas en el rango de 0 a 20.

En la clase principal:

- Declare y cree el objeto global **an** de tipo ArregloNotas.
- A la pulsación del botón **Listar** visualice las notas del arreglo.
- A la pulsación del botón **Reportar** visualice: cantidad de notas, nota promedio, nota mayor, nota menor, cantidad de notas aprobatorias, cantidad de notas desaprobatorias, cantidad de notas mayores a 15, posición de la primera nota mayor o igual a 13 y posición de la última nota menor a 13.
- A la pulsación del botón **Generar** invoque al método **generarNotas**.

Ejercicio Propuesto 3

Diseñe la clase **ArregloSueldos** en el paquete **semana_05** con el atributo privado **sueldo** (*double*) de tipo arreglo lineal y con los valores de inicialización:

2400.5, 1500.2, 800.4, 1000.3, 4700.1, 600.0, 3300.8, 2600.6, 5100.9, 2000.7

Implemente además:

- Un Constructor que no haga nada.
- Un método **tamaño** que retorne la cantidad de elementos del arreglo **sueldo**.
- Un método **obtener** que reciba una posición y retorne el sueldo registrado en dicha posición.
- Un método **sueldoPromedio** que retorne el promedio de todos los sueldos.
- Un método **sueldoMayor** que retorne el mayor de todos los sueldos.
- Un método **sueldoMenor** que retorne el menor de todos los sueldos.
- Un método **cantMayoresSueldoPromedio** que retorne la cantidad de empleados cuyo sueldo es mayor o igual al sueldo promedio.
- Un método **cantMenoresSueldoPromedio** que retorne la cantidad de empleados cuyo sueldo es menor al sueldo promedio.
- Un método **buscarPrimerSueldoMayorAlMinimo** que retorne la posición del primer sueldo mayor a 850.0. En caso no exista retorne -1.
- Un método **buscarUltimoSueldoMenorAlMinimo** que retorne la posición del último sueldo menor a 850.0. En caso no exista retorne -1.

En la clase principal:

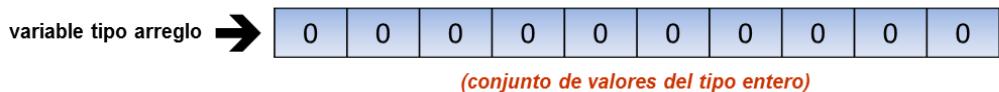
- Declare y cree el objeto global **as** de tipo ArregloSueldos.
- A la pulsación del botón **Listar** visualice los sueldos del arreglo.
- A la pulsación del botón **Reportar** visualice: cantidad de sueldos, sueldo promedio, sueldo mayor, sueldo menor, cantidad de empleados cuyo sueldo es mayor o igual al sueldo promedio, cantidad de empleados cuyo sueldo es menor al sueldo promedio, posición del primer sueldo mayor a 850.0 y posición del último sueldo menor a 850.0.

2.2. Artificios y operaciones variadas

2.2.1. Descripción

Es posible también crear un arreglo reservando una cierta cantidad de espacios antes de ingresar valores. Se usa para tener la facilidad de adicionar uno por uno los valores y en diferentes tiempos. Además, hace posible definir la dimensión del arreglo en tiempo de ejecución del programa.

Cuando se crea un arreglo con reserva Java inicializa por defecto cada valor: con **0** si es de tipo entero, **0.0** si es de tipo real, **null** si es de tipo cadena, **vacío** si es de tipo carácter y **false** si es de tipo booleano.



Observación: para efectos visuales y con la finalidad de evitar confusión no colocaremos los valores de inicialización.

2.2.2. Declaración, creación y reserva

Con la reserva 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.

Forma 1:

```
tipo[] arreglo = new tipo[longitud];
```

Ejemplo:

```
int[] n = new int[10];
```

Forma 2:

```
tipo arreglo[] = new tipo[longitud];
```

Ejemplo:

```
int n[] = new int[10];
```

Forma 3:

```
tipo[] arreglo;
arreglo = new tipo[longitud];
```

Ejemplo:

```
int[] n;
n = new int[10];
```

Forma 4:

```
tipo arreglo[];
arreglo = new tipo[longitud];
```

Ejemplo:

```
int n[];
n = new int[10];
```

2.2.3. Declaración privada, creación y reserva

Reserva de un arreglo privado con capacidad para diez números.

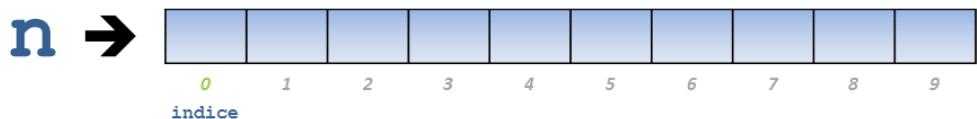
```
private int[] n = new int[10];
```



2.2.4. Ingreso personalizado

Para ingresar valores personalizados es necesario considerar un contador (*índice*) de tal manera que sirva como guía para saber dónde ubicar al nuevo valor.

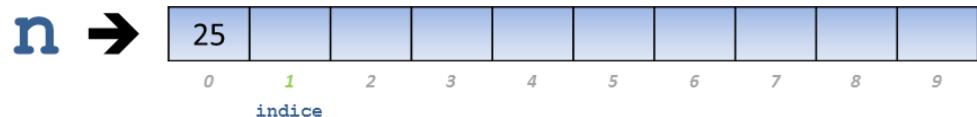
```
private int[] n = new int[10];
private int indice = 0;
```



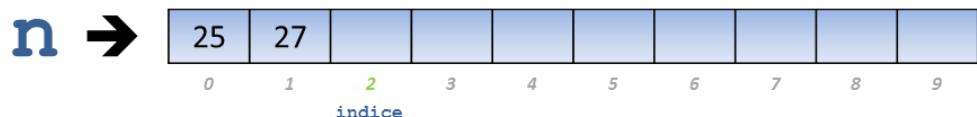
Artificio:

```
n[indice] = numero;
indice ++;
```

Al ingresar el número 25



Al ingresar el número 27



Al ingresar el número 22

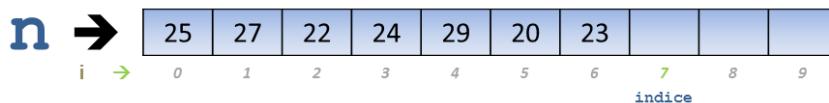


El contador **índice** nos dice que hay tres números ingresados: **n[0]**, **n[1]** y **n[2]**

También nos indica la posición donde se almacenará el siguiente ingreso.

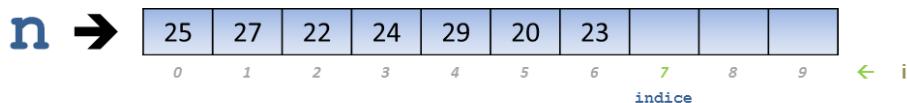
2.2.5. Recorrido

Consiste en contar las posiciones ingresadas de izquierda a derecha o viceversa.



EN ASCENSO (de izquierda a derecha)

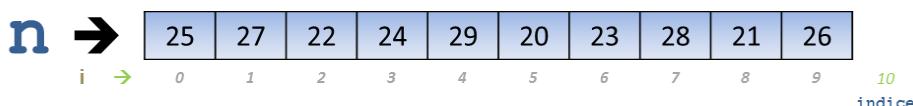
```
for (int i=0; i<indice; i++) {  
    ...  
}
```



EN DESCENSO (de derecha a izquierda)

```
for (int i=indice-1; i>=0; i--) {  
    ...  
}
```

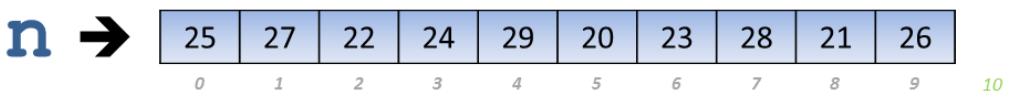
2.2.6. Redimensionamiento



Es posible redimensionar un arreglo en tiempo de ejecución del programa con la finalidad de seguir ingresando valores en forma ilimitada. El problema de redimensionar es que la data se pierde. En consecuencia debemos establecer una especie de back up. Para ello, debemos detectar cuando el arreglo se encuentre lleno.

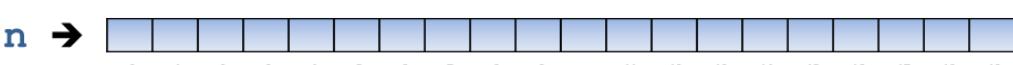
1º Declaramos un arreglo aux y lo direccionamos hacia el arreglo n

```
int[] aux = n;
```



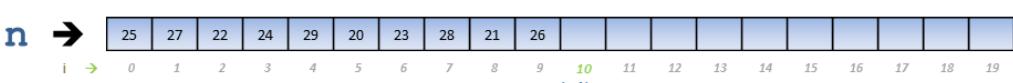
2º Redimensionamos el arreglo n a una capacidad mayor

```
n = new int[indice + 10];
```

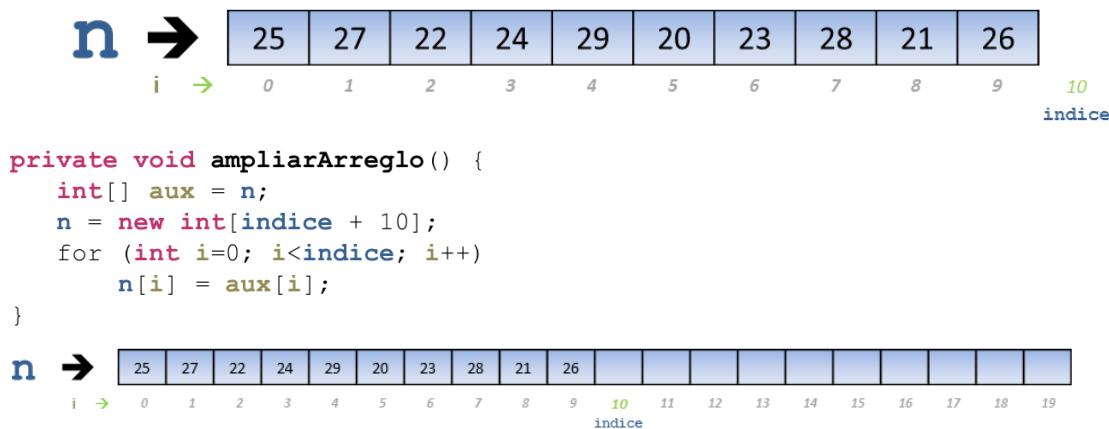


3º Recuperamos en el nuevo arreglo n cada uno de los elementos de aux

```
for (int i=0; i<indice; i++)  
    n[i] = aux[i];
```



2.2.7. Método privado ampliarArreglo



2.2.8. Operaciones públicas básicas

Son aquellos métodos que permiten acceder a los elementos registrados desde el exterior de la clase que lo contiene.

Ejemplo: método que retorna la cantidad de valores ingresados hasta ese momento

```
public int tamaño() {
    return indice;
}
```

Ejemplo: método que recibe una posición y retorna una copia del número que allí se ubica

```
public int obtener(int i) {
    return n[i];
}
```

Ejemplo: método que recibe un número y lo adiciona al arreglo

```
public void adicionar(int numero) {
    if (indice == n.length)
        ampliarArreglo();
    n[indice] = numero;
    indice++;
}
```

2.2.9. Operaciones públicas complementarias

Son los métodos que realizan operaciones adicionales.

Ejemplo: método que elimina de manera lógica el último valor ingresado

```
public void eliminarAlFinal() {
    indice--;
}
```

Ejemplo: método que elimina de manera lógica todos los valores

```
public void eliminarTodo() {
    indice = 0;
}
```

Ejercicio 13: Diseñe un algoritmo que retorne el último número del arreglo.

```
public int ultimoNumero() {  
    return n[indice-1];  
}
```

Ejercicio 14: Diseñe un algoritmo que retorne la suma de los números ingresados hasta ese momento.

```
public int sumaNumeros() {  
    int suma = 0;  
    for (int i=0; i<indice; i++)  
        suma += n[i];  
  
    return suma;  
}
```

Ejercicio 15: Diseñe un algoritmo que retorne el promedio de los números ingresados hasta ese momento.

```
public double promedio() {  
    return sumaNumeros()*1.0/indice;  
}
```

Ejercicios

Ejercicio Propuesto 1

Diseñe la clase **ArregloEdades** en el paquete **semana_06** con el atributo privado **edad** (*int*) de tipo arreglo lineal y el atributo privado **índice** (*int*).

Implemente además:

- Un Constructor sin parámetros que reserve 10 espacios en **edad** e inicialice con 0 al **índice**.
- Un método **tamaño** que retorne la cantidad de datos ingresados hasta ese momento.
- Un método **obtener** que reciba una posición y retorne la edad registrada en dicha posición.
- Un método privado **ampliarArreglo** que extienda el arreglo en diez espacios más.
- Un método **adicionar** que reciba una edad y la registre en la posición que corresponda. Verifique primero si el arreglo está lleno para invocar al método **ampliarArreglo**.
- Un método **eliminarAlFinal** que elimina lógicamente la última edad del arreglo.
- Un método **eliminarTodo** que elimina lógicamente todas las edades.
- Un método **edadMayor** que retorne la mayor de todas las edades.
- Un método **buscarPrimeraEdadAdulta** que retorne la posición de la primera edad encontrada en el rango de 20 a 59 años. En caso no exista retorne -1.
- Un método **incrementarPrimeraEdadAdulta** que aumente la primera edad adulta en 5 años.
- Un método **reemplazarPrimeraEdadAdulta** que cambie la primera edad adulta por la edad mayor.

En la clase principal:

- Declare y cree el objeto global **ae** de tipo **ArregloEdades**.
- Implemente un método **listar** que visualice las edades registradas hasta ese momento.
- A la pulsación del botón **Adicionar** lea una edad por GUI y adiciónela al arreglo. Invoque luego al método **listar**.
- A la pulsación del botón **Eliminar al final** invoque al método **eliminarAlFinal** e invoque al método **listar**. En caso que el arreglo esté vacío muestre el mensaje respectivo.
- A la pulsación del botón **Eliminar todo** invoque al método **eliminarTodo**. En caso que el arreglo esté vacío muestre el mensaje respectivo.
- A la pulsación del botón **Incrementar primera edad adulta** invoque al método **incrementarPrimeraEdadAdulta**. En caso que no exista ninguna edad adulta visualice un mensaje al respecto.
- A la pulsación del botón **Reemplazar primera edad adulta** invoque al método **reemplazarPrimeraEdadAdulta**. En caso que no exista ninguna edad adulta visualice un mensaje al respecto.

Ejercicio Propuesto 2

Diseñe la clase **ArregloNotas** en el paquete **semana_06** con el atributo privado **nota** (*int*) de tipo arreglo lineal y el atributo privado **índice** (*int*).

Implemente además:

- Un Constructor sin parámetros que reserve 10 espacios en **nota** e inicialice con 0 al **índice**.
- Un método **tamaño** que retorne la cantidad de datos ingresados hasta ese momento.

- Un método **obtener** que reciba una posición y retorne la nota registrada en dicha posición.
- Un método privado **ampliarArreglo** que extienda el arreglo en diez espacios más.
- Un método **adicionar** que reciba una nota y la registre en la posición que corresponda. Verifique primero si el arreglo está lleno para invocar al método **ampliarArreglo**.
- Un método **eliminarAlFinal** que elimina lógicamente la última nota del arreglo.
- Un método **eliminarTodo** que elimina lógicamente todas las notas.
- Un método **buscarUltimaNotaDesaprobatoria** que retorne la posición de la última nota menor que 13. En caso no exista retorne -1.
- Un método **decrementarUltimaNotaDesaprobatoria** que disminuya la última nota menor que 13 en dos puntos. En caso que al disminuir resulte una nota negativa, fije la nota en cero.
- Un método **reemplazarUltimaNotaDesaprobatoria** que cambie la última nota desaprobatoria por la última nota del arreglo.

En la clase principal:

- Declare y cree el objeto global **an** de tipo ArregloNotas.
- Implemente un método **listar** que visualice las notas registradas hasta ese momento.
- A la pulsación del botón **Adicionar** lea una nota por GUI y adicionela al arreglo. Invoca luego al método **listar**.
- A la pulsación del botón **Eliminar al final** invoque al método **eliminarAlFinal** e invoque al método **listar**. En caso que el arreglo esté vacío muestre el mensaje respectivo.
- A la pulsación del botón **Eliminar todo** invoque al método **eliminarTodo**. En caso que el arreglo esté vacío muestre el mensaje respectivo.
- A la pulsación del botón **Decrementar última nota desaprobatoria** invoque al método **decrementarUltimaNotaDesaprobatoria**. En caso que no exista ninguna nota desaprobatoria visualice un mensaje al respecto.
- A la pulsación del botón **Reemplazar última nota desaprobatoria** invoque al método **reemplazarUltimaNotaDesaprobatoria**. En caso que no exista nota desaprobatoria visualice un mensaje al respecto.

Ejercicio Propuesto 3

Diseñe la clase **ArregloTemperaturas** en el paquete **semana_06** con el atributo privado **temperatura** (*double*) de tipo arreglo lineal y el atributo privado **índice** (*int*).

Implemente además:

- Un Constructor sin parámetros que reserve 10 espacios en **temperatura** e inicialice con 0 al **índice**.
- Un método **tamaño** que retorne la cantidad de datos ingresados hasta ese momento.
- Un método **obtener** que reciba una posición y retorne la temperatura registrada en dicha posición.
- Un método privado **ampliarArreglo** que extienda el arreglo en diez espacios más.
- Un método **adicionar** que reciba una temperatura y la registre en la posición que corresponda. Verifique primero si el arreglo está lleno para invocar al método **ampliarArreglo**.
- Un método **eliminarAlFinal** que elimina lógicamente la última temperatura del arreglo.
- Un método **eliminarTodo** que elimina lógicamente todas las temperaturas.
- Un método **temperaturaMenor** que retorne la temperatura más baja.
- Un método **buscarPrimeraTemperaturaNormal** que retorne la posición de la primera temperatura encontrada en el rango de 36.1°C a 37.2°C.

- Un método **reemplazarPrimeraTemperaturaNormal** que cambie la primera temperatura normal por la menor temperatura del arreglo. Para el efecto, invoque a los métodos **buscarPrimeraTemperaturaNormal** y **temperaturaMenor**.
- Un método **incrementarTemperaturas** que aumente todas las temperaturas en 0.2°C.

En la clase principal:

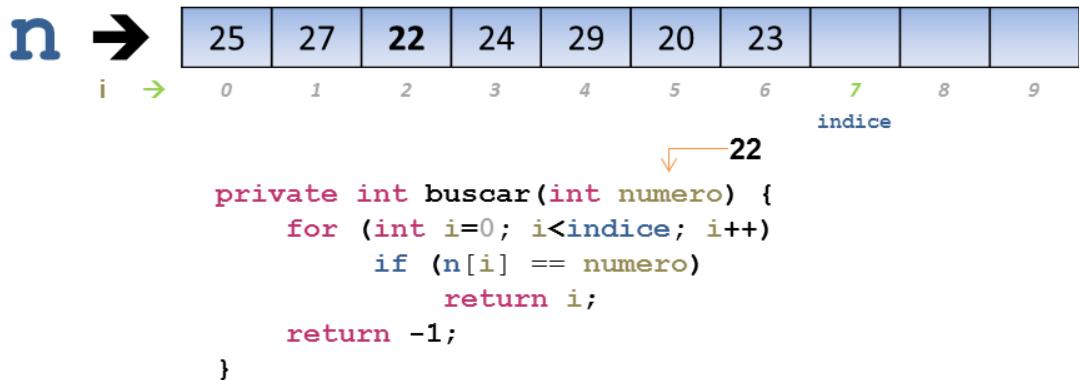
- Declare y cree el objeto global **at** de tipo **ArregloTemperaturas**.
- Implemente un método **listar** que visualice las temperaturas registradas hasta ese momento.
- A la pulsación del botón **Adicionar** lea una temperatura por GUI y adicionela al arreglo. Invoque luego al método **listar**.
- A la pulsación del botón **Eliminar al final** invoque al método **eliminarAlFinal** e invoque al método **listar**. En caso que el arreglo esté vacío muestre el mensaje respectivo.
- A la pulsación del botón **Eliminar todo** invoque al método **eliminarTodo**. En caso que el arreglo esté vacío muestre el mensaje respectivo.
- A la pulsación del botón **Reemplazar primera temperatura normal** invoque al método **reemplazarPrimeraTemperaturaNormal**. Visualice los cambios. En caso que no exista ninguna temperatura normal visualice un mensaje al respecto.
- A la pulsación del botón **Incrementar temperaturas** invoque al método **incrementarTemperaturas**. Visualice los cambios.

2.3. Artificios y operaciones especiales

2.3.1. Método privado Buscar

Consiste en buscar un número en el arreglo.

Recibe un número y retorna la posición. Si no lo encuentra retorna -1.



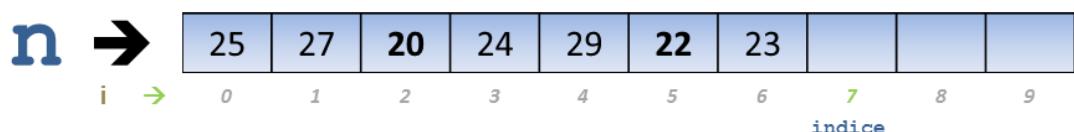
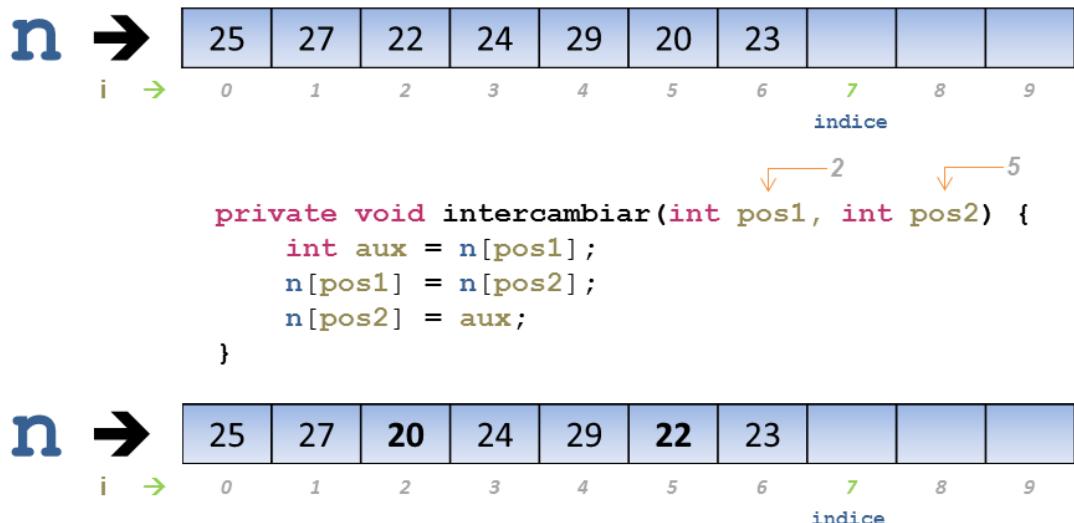
Ejemplo:

```
int pos1 = buscar(22); // pos1 ← 2
int pos2 = buscar(28); // pos2 ← -1
```

2.3.2. Método privado Intercambiar

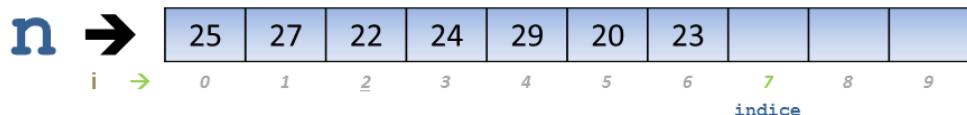
Consiste en cambiar de lugar a dos contenidos del arreglo. Para ello debemos definir las posiciones y apoyarnos en una variable auxiliar.

Recibe dos posiciones e intercambia sus contenidos.



2.3.3. Operaciones públicas complementarias

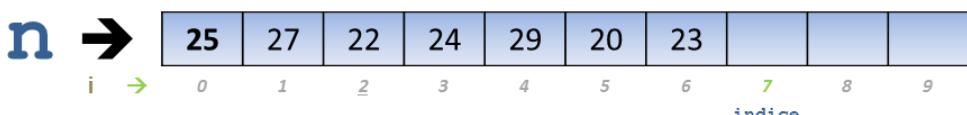
Ejercicio 16: Busque un número (recibido), y si no está registrado adiciónelo.



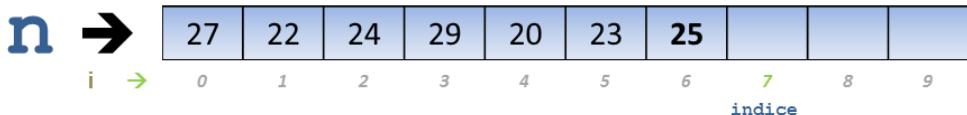
```
public void buscarAdicionar(int numero) {
    int pos = buscar(numero);
    if (pos == -1)
        adicionar(numero);
}
```



Ejercicio 17: Traslade el primer número al final.



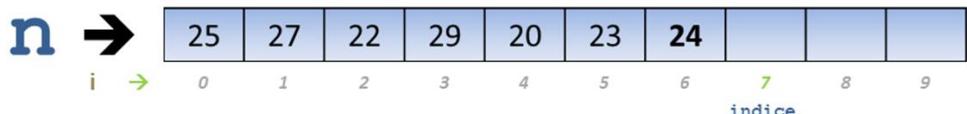
```
public void primeroAlFinal() {
    for (int i=0; i<indice-1; i++)
        intercambiar(i, i+1);
}
```



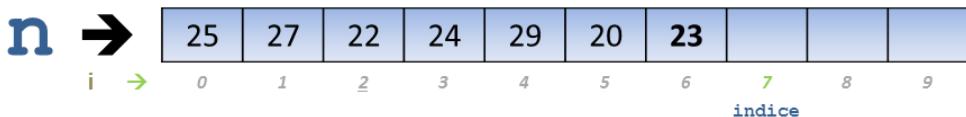
Ejercicio 18: Busque un número (recibido) y si está registrado trasladarlo al final.



```
public void buscarRezagar(int numero) {
    int pos = buscar(numero);
    if (pos != -1)
        for (int i=pos; i<indice-1; i++)
            intercambiar(i, i+1);
}
```



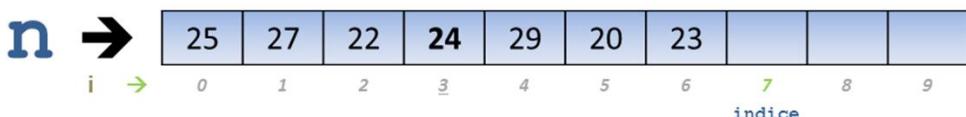
Ejercicio 19: Traslade el último número al inicio.



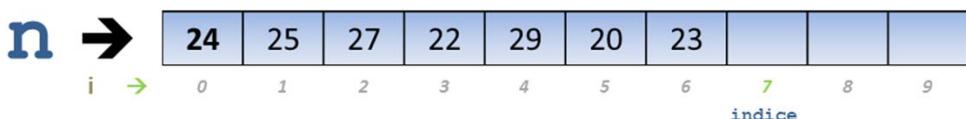
```
public void ultimoAlInicio() {
    for (int i=indice-1; i>0; i--)
        intercambiar(i, i-1);
}
```



Ejercicio 20: Busque un número (recibido) y si está registrado trasladarlo al inicio.



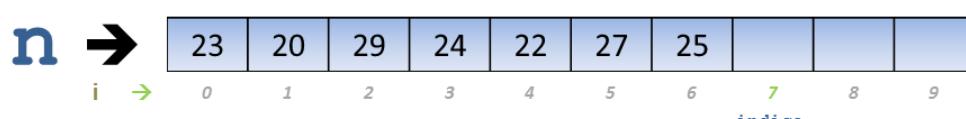
```
public void buscarPriorizar(int numero) {
    int pos = buscar(numero);
    if (pos != -1)
        for (int i=pos; i>0; i--)
            intercambiar(i, i-1);
}
```



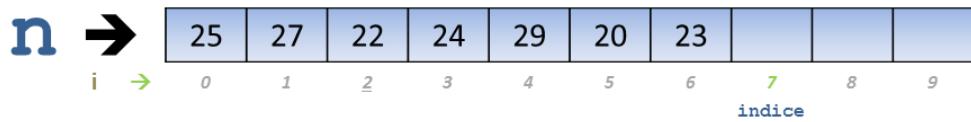
Ejercicio 21: Invierta el contenido del arreglo.



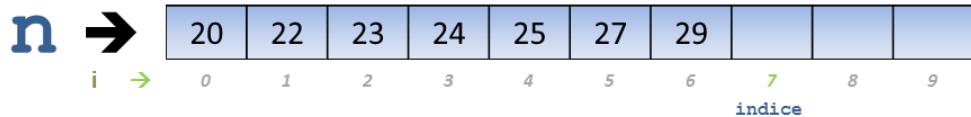
```
public void invertirArreglo() {
    int i = 0, j = indice-1;
    while (i < j) {
        intercambiar(i, j);
        i++;
        j--;
    }
}
```



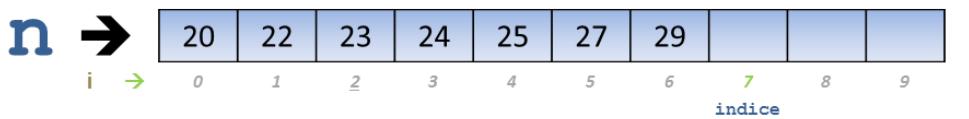
Ejercicio 22: Ordene de menor a mayor el contenido del arreglo



```
public void ordenarArreglo() {
    for (int i=0; i<indice-1; i++)
        for (int j=i+1; j<indice; j++)
            if (n[i] > n[j])
                intercambiar(i, j);
}
```



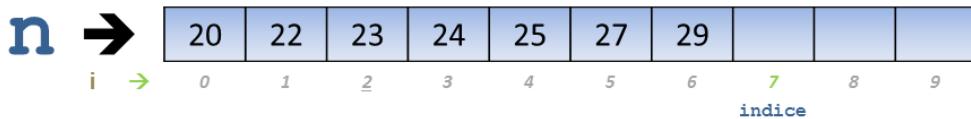
Ejercicio 23: Adicione un número (recibido) de tal manera que el arreglo siempre quede ordenado de menor a mayor.



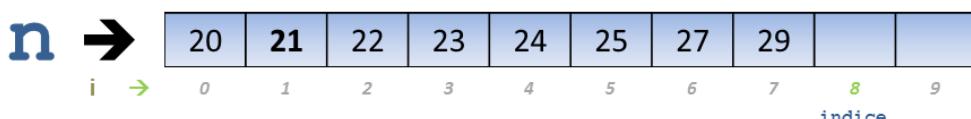
```
public void adicionarOrdenar(int numero) {
    adicionar(numero);
    for (int i=indice-1; i>0 && n[i] < n[i-1]; i--)
        intercambiar(i, i-1);
}
```



Ejercicio 24: Busque un número (recibido), y si no está registrado adicónelo de tal manera que el arreglo siempre quede ordenado de menor a mayor.



```
public void buscarAdicionarOrdenar(int numero) {
    int pos = buscar(numero);
    if (pos == -1)
        adicionarOrdenar(numero);
}
```

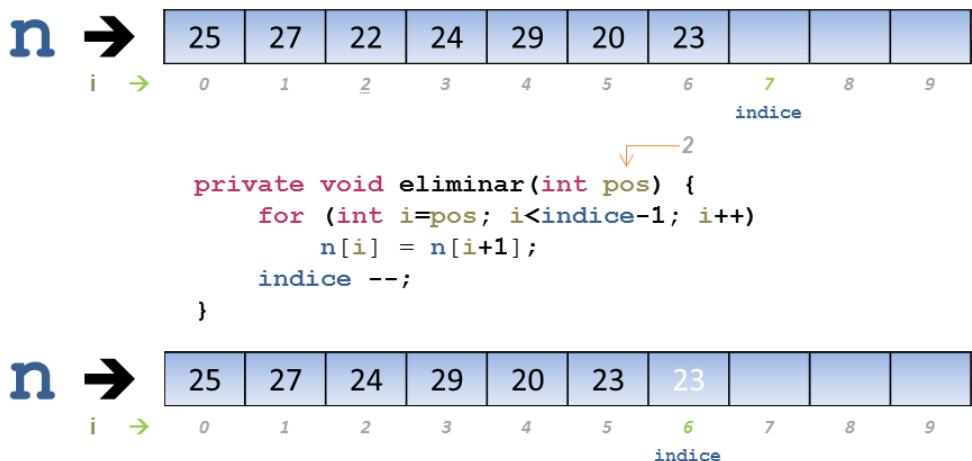


2.4. Artificios y operaciones complementarias

2.4.1. Método privado Eliminar

Consiste en anular un número ingresado.

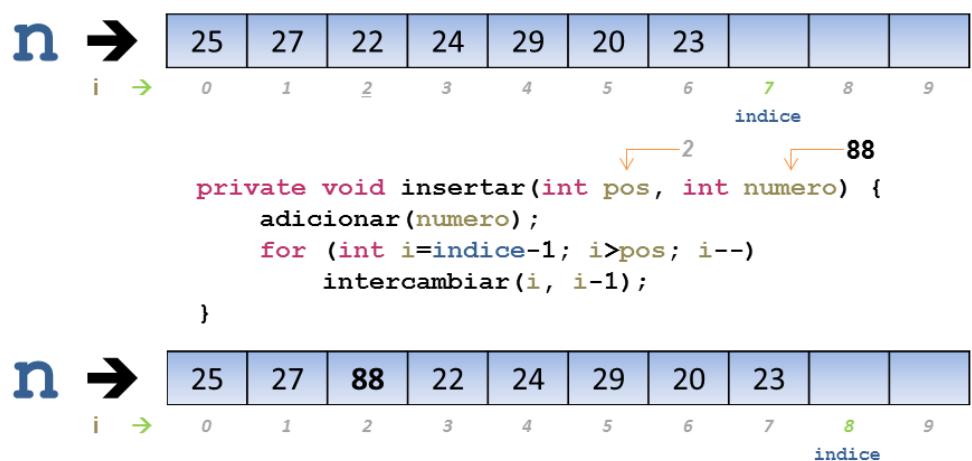
Recibe una posición y elimina del arreglo el valor ubicado en dicha posición.



2.4.2. Método privado Insertar

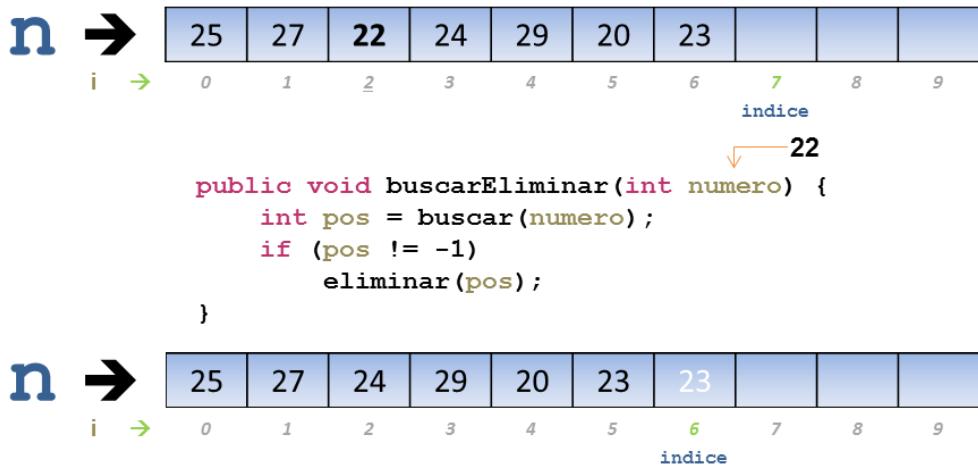
Consiste en registrar un número en una determinada ubicación.

Recibe una posición, un número e inserta el valor en dicha posición

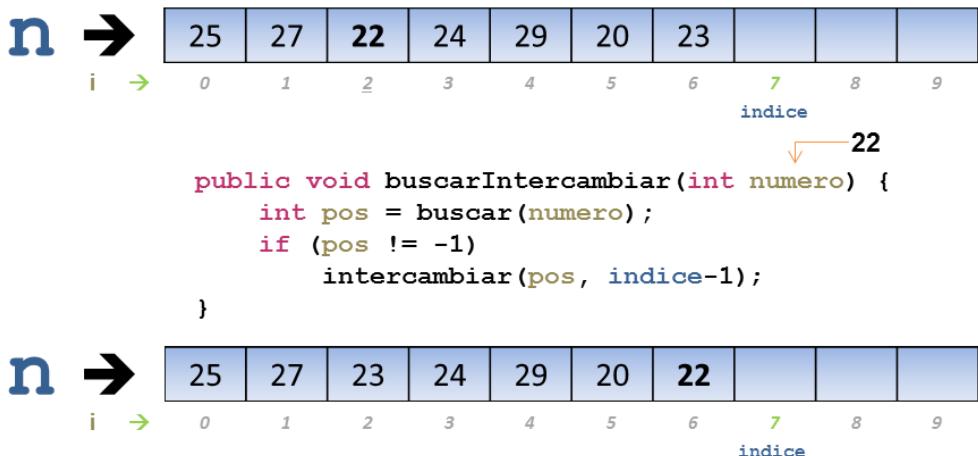


2.4.3. Operaciones públicas complementarias

Ejercicio 25: Busque un número (recibido) y si está registrado elimínelo.



Ejercicio 26: Busque un número (recibido) y si está registrado intercambielo con el último número del arreglo.



Ejercicios

Ejercicio Propuesto 1

Dada la implementación de la clase **ArregloEdades** en el paquete **semana_08** adicione:

- Un método **buscarPrimeraEdadAdolescente** que retorne la posición de la primera edad adolescente (edad en el rango de 12 a 20 años). En caso que no exista, retorne el valor -1.
- Un método **buscarUltimaEdadAdolescente** que retorne la posición de la última edad adolescente (edad en el rango de 12 a 20 años). En caso que no exista, retorne el valor -1.
- Un método **reemplazarPrimeraEdadAdolescente** que reemplace la primera edad adolescente por la última edad.
- Un método **intercambiarEdadesAdolescentesExtremas** que intercambie la primera edad adolescente con la última edad adolescente.
- Un método **eliminarPrimeraEdadAdolescente** que elimine la primera edad adolescente.

En la clase principal:

- A la pulsación del botón **Remplazar primera edad adolescente**, invoque al método **reemplazarPrimeraEdadAdolescente** y visualice un listado actualizado de edades. En caso que el reemplazo no sea posible, muestre un mensaje al respecto.
- A la pulsación del botón **Intercambiar edades adolescentes extremas** invoque al método **intercambiarEdadesAdolescentesExtremas** y visualice un listado actualizado de edades. En caso que el intercambio no sea posible, muestre un mensaje al respecto.
- A la pulsación del botón **Eliminar primera edad adolescente** invoque al método **eliminarPrimeraEdadAdolescente** y visualice un listado actualizado de edades. En caso que la eliminación no sea posible, muestre un mensaje al respecto.

Ejercicio Propuesto 2

Dada la implementación de la clase **ArregloNotas** en el paquete **semana_08** adicione:

- Un método **notaMenor** que retorne la menor de todas las notas.
- Un método **buscarPrimeraNotaAprobatoria** que retorne la posición de la primera nota aprobatoria (nota en el rango de 13 a 20). En caso que no exista, retorne -1.
- Un método **buscarUltimaNotaAprobatoria** que retorne la posición de la última nota aprobatoria (nota en el rango de 13 a 20). En caso que no exista, retorne -1.
- Un método **reemplazarUltimaNotaAprobatoria** que reemplace la última nota aprobatoria por la nota menor.
- Un método **eliminarPrimeraNotaAprobatoria** que elimine la primera nota aprobatoria.

En la clase principal:

- A la pulsación del botón **Remplazar última nota aprobatoria**, invoque al método **reemplazarUltimaNotaAprobatoria** y visualice un listado actualizado de notas. En caso que el reemplazo no sea posible, muestre un mensaje al respecto.
- A la pulsación del botón **Eliminar primera nota aprobatoria** invoque al método **eliminarPrimeraNotaAprobatoria** y visualice un listado actualizado de notas. En caso que la eliminación no sea posible, muestre un mensaje al respecto.

Ejercicio Propuesto 3

Dada la implementación de la clase **ArregloTemperaturas** en el paquete **semana_08** adicione:

- Un método **temperaturaPromedio** que retorne el promedio de todas las temperaturas.
- Un método **buscarPrimeraTemperaturaFebril** que retorne la posición de la primera temperatura febril (temperatura mayor que 37.2 °C). En caso que no exista, retorne el valor -1.

- Un método **buscarUltimaTemperaturaFebril** que retorne la posición de la última temperatura febril (temperatura mayor que 37.2 °C). En caso que no exista, retorne el valor -1.
- Un método **reemplazarPrimeraTemperaturaFebril** que reemplace la primera temperatura febril por la última temperatura febril.
- Un método **reemplazarUltimaTemperaturaFebril** que reemplace la última temperatura febril por la temperatura promedio.
- Un método **eliminarPrimeraTemperaturaFebril** que elimine la primera temperatura febril.

En la clase principal:

- A la pulsación del botón **Remplazar primera temperatura febril**, invoque al método **reemplazarPrimeraTemperaturaFebril** y visualice un listado actualizado de temperaturas. En caso que el remplazo no sea posible, muestre un mensaje al respecto.
- A la pulsación del botón **Reemplazar última temperatura febril**, invoque al método **reemplazarUltimaTemperaturaFebril** y visualice un listado actualizado de temperaturas. En caso que el remplazo no sea posible, muestre un mensaje al respecto.
- A la pulsación del botón **Eliminar primera temperatura febril** invoque al método **eliminarPrimeraTemperaturaFebril** y visualice un listado actualizado de temperaturas. En caso que la eliminación no sea posible, muestre un mensaje al respecto.

Ejercicio Propuesto 4

Diseñe la clase **ArregloCódigos** en el paquete **semana_08** con el atributo privado **codigo** (*int*) de tipo arreglo lineal y el atributo privado **índice** (*int*).

Implemente además:

- Un Constructor sin parámetros que reserve 10 espacios en **codigo** e inicialice con 0 al **índice**.
- Un método **tamaño** que retorne la cantidad de datos ingresados hasta ese momento.
- Un método **obtener** que reciba una posición y retorne el código registrado en dicha posición.
- Un método privado **ampliarArreglo** que extienda el arreglo en diez espacios más.
- Un método **adicionar** que reciba un código y lo registre en la posición que corresponda. Verifique primero si el arreglo está lleno para invocar al método **ampliarArreglo**.
- Un método **intercambiarSegPen** que cambie de lugar al segundo y penúltimo código del arreglo.
- Un método **eliminarPrimero** que retire el primer código del arreglo.
- Un método **buscarCodigo** que retorne la posición del último código que se encuentre en el rango de 1000 a 1111. En caso no exista retorne -1.
- Un método **intercambiarCodigo** que cambie de lugar al último código que se encuentre en el rango de 1000 a 1111 con el tercer código del arreglo.
- Un método **eliminarCodigo** que retire del arreglo al último código que se encuentre en el rango de 1000 a 1111.

En la clase principal:

- Declare y cree el objeto global **ac** de tipo **ArregloCódigos**.
- Implemente un método **listar** que visualice los códigos registrados hasta ese momento.
- A la pulsación del botón **Adicionar** lea un código por GUI y adicionelo al arreglo. Invoque luego al método **listar**.
- A la pulsación del botón **Intercambiar 1** invoque al método **intercambiarSegPen** e invoque al método **listar**. En caso de que no sea posible muestre el mensaje respectivo.
- A la pulsación del botón **Eliminar 1** invoque al método **eliminarPrimero**. En caso que el arreglo esté vacío muestre el mensaje respectivo.

- A la pulsación del botón **Intercambiar 2** invoque al método **intercambiarCodigo**. En caso de que no sea posible visualice un mensaje al respecto.
- A la pulsación del botón **Eliminar 2** invoque al método **eliminarCodigo**. En caso de que no sea posible visualice un mensaje al respecto.



CLASE ARRAYLIST

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).

TEMARIO

3.1. Tema 9 : Conceptos y operaciones simples

- 3.1.1 : Descripción
- 3.1.2 : Coleccionista de objetos distintos
- 3.1.3 : Coleccionista de objetos iguales
- 3.1.4 : Declaración privada y creación
- 3.1.5 : Métodos básicos de la clase *ArrayList*
- 3.1.6 : Operaciones públicas básicas
- 3.1.7 : Operaciones públicas complementarias

3.2. Tema 10 : Operaciones variadas

- 3.2.1 : Métodos adicionales de la clase *ArrayList*
- 3.2.2 : Operaciones públicas básicas

3.3. Tema 11 : Mantenimiento

- 3.3.1 : Diseño básica de un proyecto

ACTIVIDADES PROPUESTAS

- Emplear los métodos de la clase ***ArrayList*** para manipular un arreglo de objetos.
- Crear un mantenimiento.
- Manipular arreglo de objetos utilizando los métodos del ***ArrayList***.

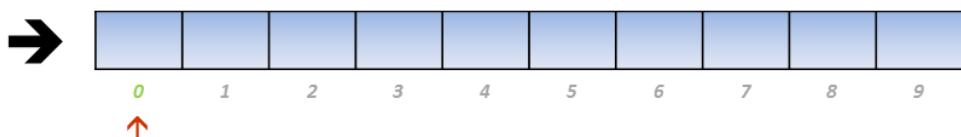
3.1. Conceptos y operaciones simples

3.1.1. Descripción

La clase **ArrayList** implementada por Java permite manipular una colección de objetos. Su diseño híbrido le permite comportarse como arreglo (*Array*) o como lista (*List*). Internamente todo es dinámico.

La Clase **ArrayList** inicializa por defecto un arreglo con capacidad inicial para diez direcciones de memoria. Al comienzo todas las posiciones apuntan a **null**.

Define también un guía (*int*) que apunta a la posición cero. Cuando requiera mayor capacidad ampliará el arreglo en diez espaciamientos más, y así sucesivamente.



Para tener acceso a la clase **ArrayList** es necesario colocar:

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

3.1.2. Coleccionista de objetos distintos

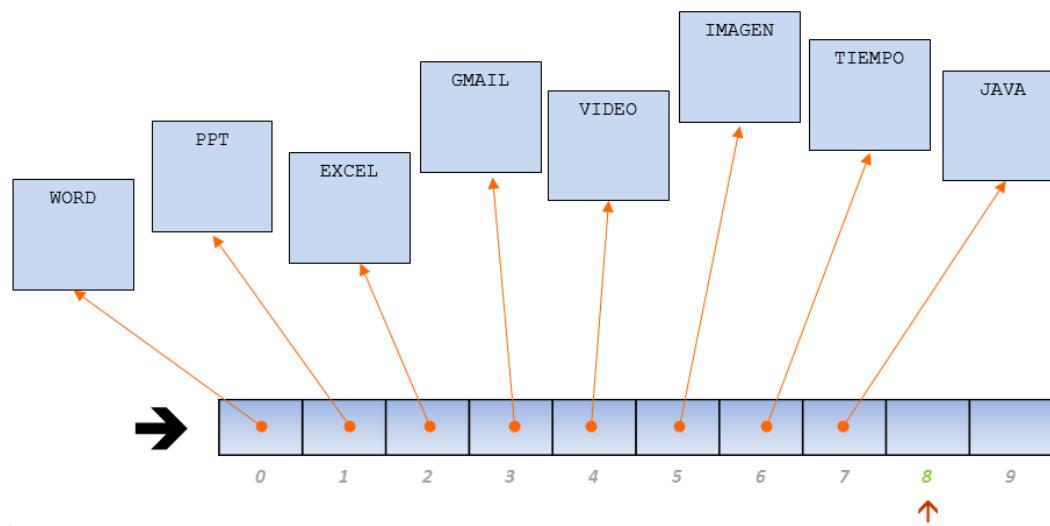
El **ArrayList** almacena por defecto direcciones de memoria de objetos diferentes.

Forma 1

```
ArrayList colecciónista;  
...  
colecciónista = new ArrayList();
```

Forma 2

```
ArrayList colecciónista = new ArrayList();
```



3.1.3. Coleccionista de objetos iguales

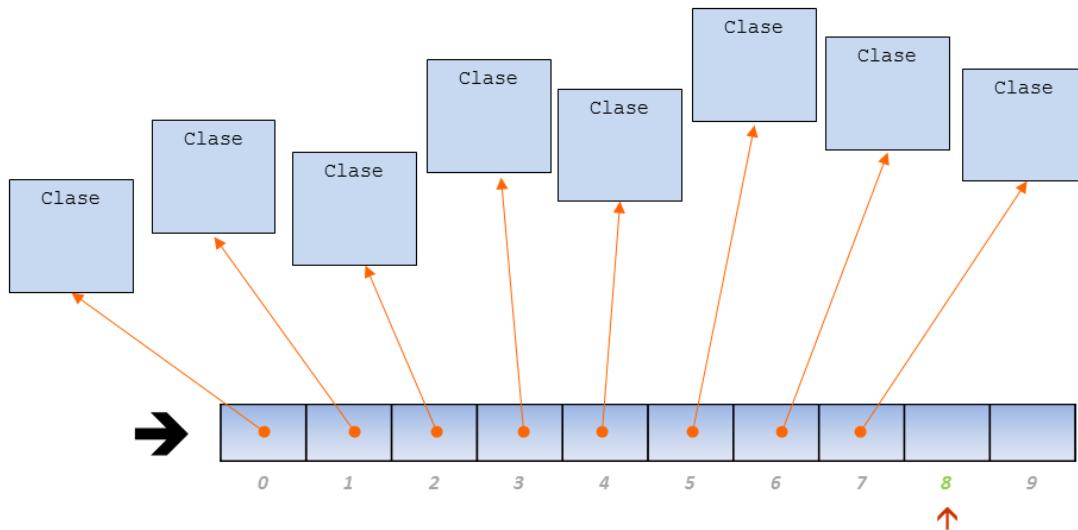
La clase **ArrayList** puede comportarse como un coleccionista de objetos iguales. En ese caso debemos colocar el nombre de una **Clase** entre los signos < >

Forma 1

```
ArrayList <Clase> coleccionista;
...
colecciónista = new ArrayList <Clase> ();
```

Forma 2

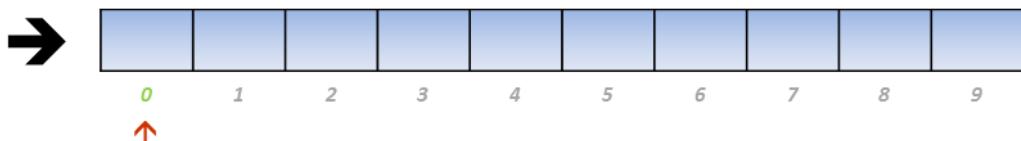
```
ArrayList <Clase> coleccionista = new ArrayList <Clase> ();
```



El coleccionista ahora sólo puede almacenar objetos del mismo tipo.

3.1.4. Declaración privada y creación

```
private ArrayList <Alumno> alu = new ArrayList <Alumno> ();
```



El coleccionista **alu** sólo almacenará direcciones de memoria de tipo **Alumno**.

3.1.4.1. Clase Alumno

```
package clase;

public class Alumno {
    // Atributos privados
    private int codigo, nota1, nota2;
    private String nombre;

    // Constructor
    public Alumno(int codigo, String nombre, int nota1, int nota2) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.nota1 = nota1;
        this.nota2 = nota2;
    }

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

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

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

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

    public int getCodigo() {
        return codigo;
    }

    public String getNombre() {
        return nombre;
    }

    public int getNota1() {
        return nota1;
    }

    public int getNota2() {
        return nota2;
    }

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

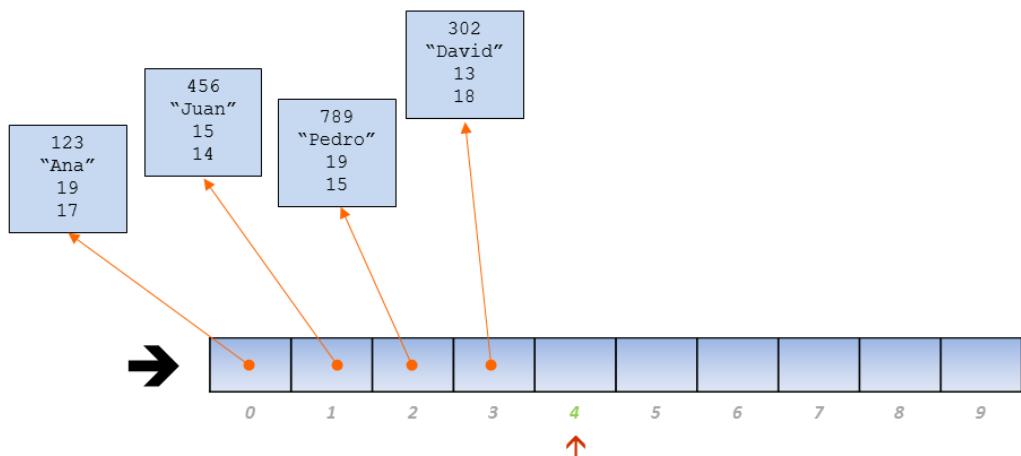
3.1.5. Métodos básicos de la clase ArrayList

1) `public void add(Object) {
}`

Adiciona una dirección de memoria en la posición que sigue.

Ej:

```
alu.add(new Alumno(123, "Ana", 19, 17));  
alu.add(new Alumno(456, "Juan", 15, 14));  
alu.add(new Alumno(789, "Pedro", 19, 15));  
alu.add(new Alumno(302, "David", 13, 18));
```

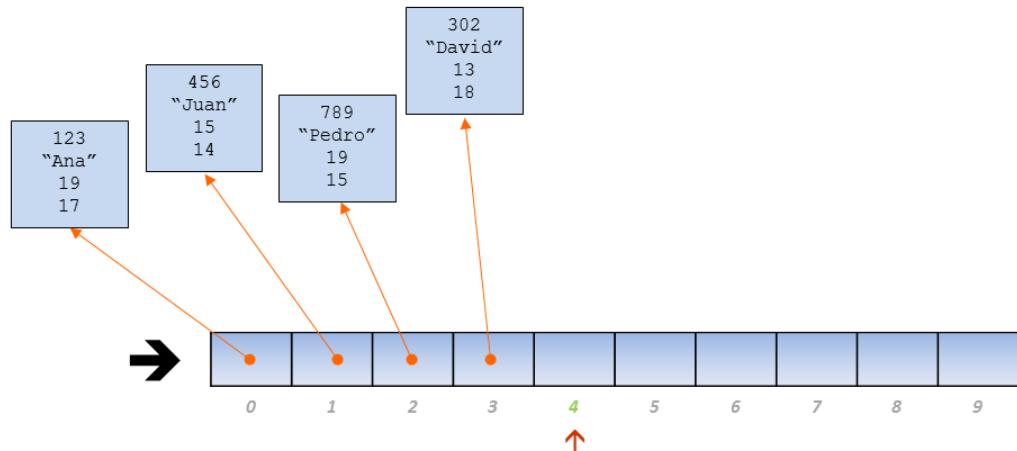


2) `public int size() {
}`

Retorna la cantidad de direcciones de memoria almacenadas.

Ej:

```
int cantidad = alu.size(); // cantidad es igual a 4
```

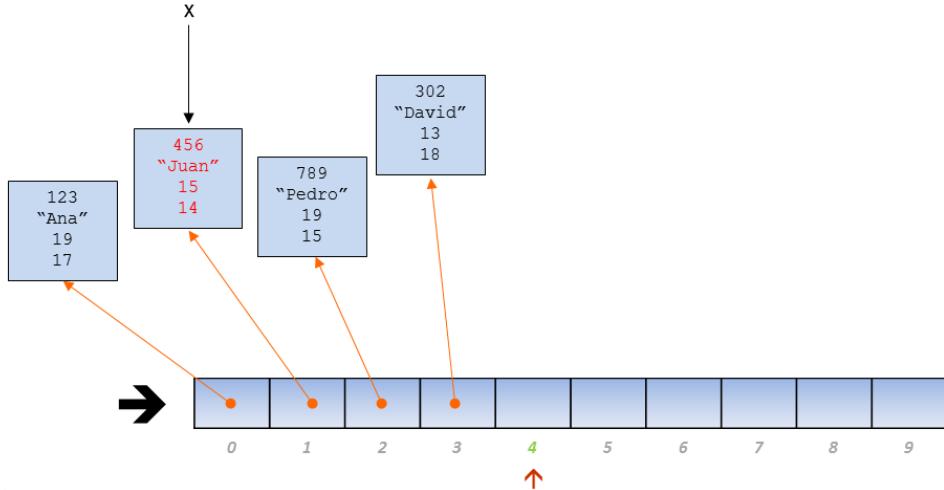


```
3) public Object get(int) {
}
```

Retorna la dirección de memoria registrada en la posición que se indica.

Ej:

```
Alumno x = alu.get(1); // x tiene la dirección de memoria del segundo alumno
```



3.1.6. Operaciones públicas básicas

Son los métodos que permiten acceder al **ArrayList** desde el exterior de la clase que lo implementa.

Ejemplo: método que retorna la cantidad de elementos registrados en el **ArrayList**

```
public int tamaño() {
    return alu.size();
}
```

Ejemplo: método que recibe una posición y retorna la referencia del alumno respectivo

```
public Alumno obtener(int i) {
    return alu.get(i);
}
```

3.1.7. Operaciones públicas complementarias

Son aquellos métodos que realizan operaciones complementarias.

Ejemplo: método que retorna el promedio de promedios de todos los alumnos

```
public double promedioGeneral() {
    double suma = 0.0;
    for (int i=0; i<tamaño(); i++)
        suma += obtener(i).promedio();
    return suma / tamaño();
}
```

Ejercicio 27: Implemente la clase ArregloAlumnos en el paquete arreglo con el atributo privado:

- ArrayList **alu** de tipo Alumno.

Implemente como públicos:

- Un constructor que cree el *ArrayList* y autogeneré cuatro objetos.
- Un método **tamaño** que retorne la cantidad de elementos registrados hasta ese momento.
- Un método **obtener** que reciba la posición y retorne la dirección de memoria del alumno respectivo.
- Un método **promedioGeneral** que retorne el promedio de promedios de todos los alumnos.

```
package arreglo;

import clase.Alumno;
import java.util.ArrayList;

public class ArregloAlumnos {
    // Atributo privado
    private ArrayList <Alumno> alu;

    // Constructor
    public ArregloAlumnos() {
        alu = new ArrayList <Alumno> ();
        alu.add(new Alumno(123, "Ana", 19, 17));
        alu.add(new Alumno(456, "Juan", 15, 14));
        alu.add(new Alumno(789, "Pedro", 19, 15));
        alu.add(new Alumno(302, "David", 13, 18));
    }

    // Operaciones públicas básicas
    public int tamaño() {
        return alu.size();
    }

    public Alumno obtener(int i) {
        return alu.get(i);
    }

    // Operaciones públicas complementarias
    public double promedioGeneral() {
        double suma = 0.0;
        for (int i=0; i<tamaño(); i++)
            suma += obtener(i).promedio();

        return suma / tamaño();
    }
}
```

Ejercicio 28: En el programa principal declare y cree como variable global un objeto de tipo ArregloAlumnos y a la pulsación del botón respectivo:

- Visualice los datos completos de los alumnos.
- Muestre la cantidad de alumnos y el promedio general de todos los alumnos.

```
package gui;

import arreglo.ArregloAlumnos;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import java.awt.Font;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

public class Ejemplo extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;

    private JPanel contentPane;
    private JButton btnListar;
    private JButton btnReportar;
    private JScrollPane scrollPaneA;
    private JScrollPane scrollPaneB;
    private JTextArea txtS;
    private JTable tblTabla;
    private DefaultTableModel modelo;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Ejemplo frame = new Ejemplo();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
/**  
 * Create the frame.  
 */  
public Ejemplo() {  
    setTitle("Ejemplo - Semana_09");  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    setBounds(100, 100, 430, 500);  
  
    contentPane = new JPanel();  
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
    setContentPane(contentPane);  
    contentPane.setLayout(null);  
  
    btnListar = new JButton("Listar");  
    btnListar.addActionListener(this);  
    btnListar.setBounds(116, 11, 89, 23);  
    contentPane.add(btnListar);  
  
    btnReportar = new JButton("Reportar");  
    btnReportar.addActionListener(this);  
    btnReportar.setBounds(209, 11, 89, 23);  
    contentPane.add(btnReportar);  
  
    scrollPaneA = new JScrollPane();  
    scrollPaneA.setBounds(10, 48, 394, 200);  
    contentPane.add(scrollPaneA);  
  
    tblTabla = new JTable();  
    tblTabla.setFillsViewportHeight(true);  
    scrollPaneA.setViewportView(tblTabla);  
  
    modelo = new DefaultTableModel();  
    modelo.addColumn("código");  
    modelo.addColumn("nombre");  
    modelo.addColumn("nota 1");  
    modelo.addColumn("nota 2");  
    modelo.addColumn("promedio");  
    tblTabla.setModel(modelo);  
  
    scrollPaneB = new JScrollPane();  
    scrollPaneB.setBounds(10, 252, 394, 200);  
    contentPane.add(scrollPaneB);  
  
    txtS = new JTextArea();  
    txtS.setFont(new Font("Monospaced", Font.PLAIN, 13));  
    scrollPaneB.setViewportView(txtS);  
}  
  
public void actionPerformed(ActionEvent arg0) {  
    if (arg0.getSource() == btnReportar) {  
        actionPerformedBtnReportar(arg0);  
    }  
  
    if (arg0.getSource() == btnListar) {  
        actionPerformedBtnListar(arg0);  
    }  
}
```

```
// Declaración global
ArregloAlumnos aa = new ArregloAlumnos();

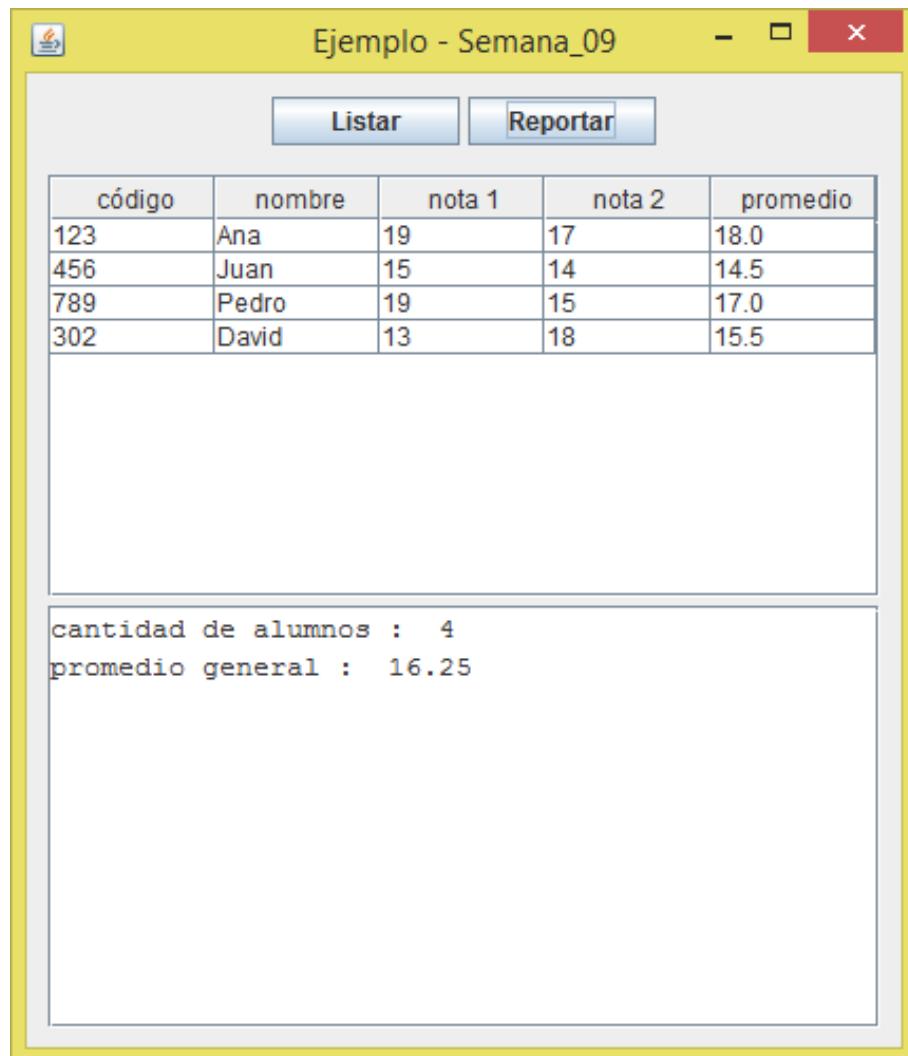
protected void actionPerformedBtnListar(ActionEvent arg0) {
    /**
     * Visualiza el contenido del ArrayList
     */
    modelo.setRowCount(0);
    for (int i=0; i<aa.tamaño(); i++) {
        Object[] fila = { aa.obtener(i).getCodigo(),
                          aa.obtener(i).getNombre(),
                          aa.obtener(i).getNota1(),
                          aa.obtener(i).getNota2(),
                          aa.obtener(i).promedio() };

        modelo.addRow(fila);
    }
}

protected void actionPerformedBtnReportar(ActionEvent arg0) {
    /**
     * Muestra un reporte del ArrayList
     */
    txtS.setText("");
    imprimir("cantidad de alumnos : " + aa.tamaño());
    imprimir("promedio general : " + aa.promedioGeneral());
}

// Métodos tipo void sin parámetros
void imprimir() {
    imprimir("");
}

// Métodos tipo void con parámetros
void imprimir(String s) {
    txtS.append(s + "\n");
}
```



The screenshot shows a Windows application window titled "Ejemplo - Semana_09". The window has a yellow header bar with the title and standard window controls (minimize, maximize, close). Below the header is a toolbar with two buttons: "Listar" and "Reportar". The main area contains a table with student data:

código	nombre	nota 1	nota 2	promedio
123	Ana	19	17	18.0
456	Juan	15	14	14.5
789	Pedro	19	15	17.0
302	David	13	18	15.5

Below the table, the window displays the following text output:

```
cantidad de alumnos : 4
promedio general : 16.25
```

Ejercicios

Ejercicio Propuesto 1

Dada la implementación de la clase **ArregloAlumnos** en el paquete **arreglo**

```
package arreglo;

import clase.Alumno;
import java.util.ArrayList;

public class ArregloAlumnos {
    // Atributo privado
    private ArrayList <Alumno> alu;

    // Constructor
    public ArregloAlumnos() {
        alu = new ArrayList <Alumno> ();
        alu.add(new Alumno(123, "Ana", 19, 17));
        alu.add(new Alumno(456, "Juan", 15, 14));
        alu.add(new Alumno(789, "Pedro", 19, 15));
        alu.add(new Alumno(302, "David", 13, 18));
        alu.add(new Alumno(208, "Carlos", 20, 19));
        alu.add(new Alumno(417, "Jorge", 12, 13));
        alu.add(new Alumno(208, "María", 15, 17));
        alu.add(new Alumno(820, "José", 11, 10));
    }

    // Operaciones públicas básicas
    public int tamaño() {
        return alu.size();
    }

    public Alumno obtener(int i) {
        return alu.get(i);
    }

    // Operaciones públicas complementarias
    public int cantAprobados() {
        return 0;
    }
}
```

Agregue en esta última la operación complementaria que retorne:

- Cantidad de alumnos aprobados (promedio mayor o igual a 13).
- Cantidad de alumnos desaprobados (promedio menor a 13).
- Promedio mayor y menor.
- Nombre del primer alumno desaprobado. En caso no exista retorne null.
- Nombre del último alumno aprobado. En caso no exista retorne null.

En la clase principal:

- Considere la declaración global **ArregloAlumnos aa = new ArregloAlumnos();**
- Implemente la pulsación del botón **Listar** que muestre los datos completos de cada alumno.
- Implemente la pulsación del botón **Reportar** que muestre los retornos de los métodos complementarios.

Ejercicio Propuesto 2

Implemente la clase **Docente** en el paquete **clase** con los atributos privados: código (*int*), nombre (*String*), horas (*int*) y tarifa (*double*); un constructor que inicialice los atributos privados, los métodos de acceso público set/get y el método sueldo (horas * tarifa).

Implemente la clase **ArregloDocentes** en el paquete **arreglo** con el atributo privado **doc** (ArrayList de tipo Docente) e implemente:

Métodos básicos

- Un constructor que cree el ArrayList y adicione las DirMem de ocho objetos Docente.
- Método tamaño que retorne la cantidad de objetos Docente registrados hasta ese momento.
- Método obtener que reciba una posición y retorne la DirMem del objeto Docente respectivo.

Métodos complementarios

- Retorne el sueldo promedio.
- Retorne el sueldo mayor.
- Retorne el sueldo menor.
- Retorne la tarifa mayor.
- Retorne la tarifa menor.

En la clase principal:

- Considere la declaración global **ArregloDocentes ad = new ArregloDocentes();**
- Implemente la pulsación del botón **Listar** que muestre los datos completos de cada docente.
- Implemente la pulsación del botón **Reportar** que muestre los retornos de los métodos complementarios.

Ejercicio Propuesto 3

Implemente la clase **Factura** en el paquete **clase** con los atributos privados: ruc (*String*), empresa (*String*), unidades (*int*) y precio unitario (*double*); un constructor que inicialice los atributos privados, los métodos de acceso público set/get y el método importeFacturado (unidades * precio unitario).

Implemente la clase **ArregloFacturas** en el paquete **arreglo** con el atributo privado **fac** (ArrayList de tipo Factura) e implemente:

Métodos básicos

- Un constructor que cree el ArrayList y adicione las DirMem de ocho objetos Factura.
- Método tamaño que retorne la cantidad de objetos Factura registrados hasta ese momento.
- Método obtener que reciba una posición y retorne la DirMem del objeto Factura respectivo.

Métodos complementarios

- Retorne suma de todos los importes facturados.
- Retorne importe promedio facturado.
- Retorne menor importe facturado.
- Retorne mayor importe facturado.

- Retorne nombre de la primera empresa cuyo importe facturado sea menor al importe promedio facturado.

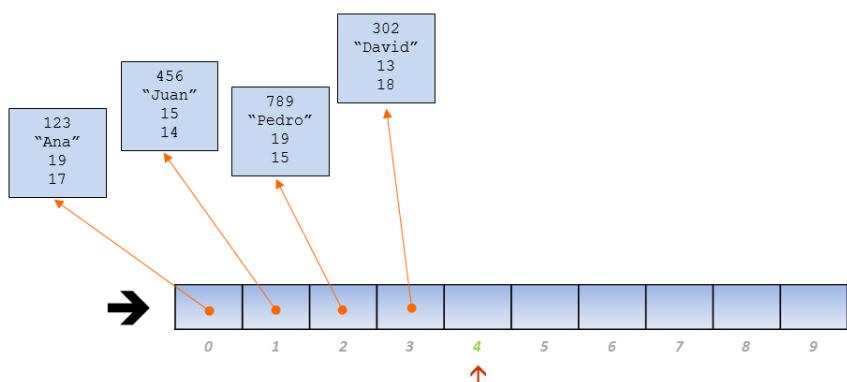
En la clase principal:

- Considere la declaración global ArregloFacturas **af** = new ArregloFacturas();
- Implemente la pulsación del botón **Listar** que muestre los datos completos de cada factura.
- Implemente la pulsación del botón **Reportar** que muestre los retornos de los métodos complementarios.

3.2. Operaciones variadas

```
private ArrayList <Alumno> alu = new ArrayList <Alumno> ();
```

```
alu.add(new Alumno(123, "Ana", 19, 17));
alu.add(new Alumno(456, "Juan", 15, 14));
alu.add(new Alumno(789, "Pedro", 19, 15));
alu.add(new Alumno(302, "David", 13, 18));
```

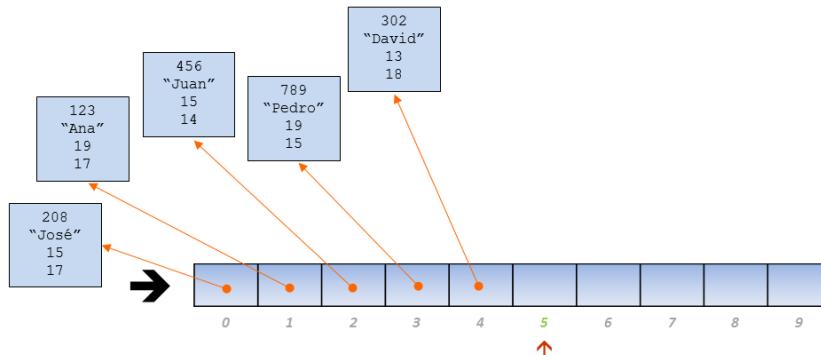


3.2.1. Métodos adicionales de la clase ArrayList

1) `public void add(int, Object) {`
 `}`

Inserta la dirección de memoria en la posición que se indica.

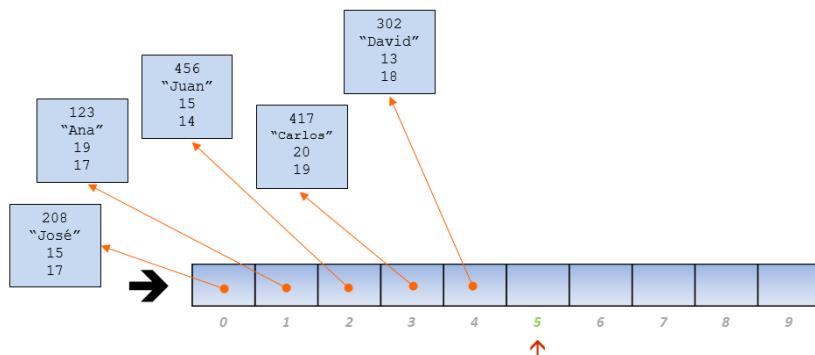
Ej:
`alu.add(0, new Alumno(208, "José", 15, 17));`



2) `public void set(int, Object) {`
 `}`

Impone una dirección de memoria en la posición que se indica.

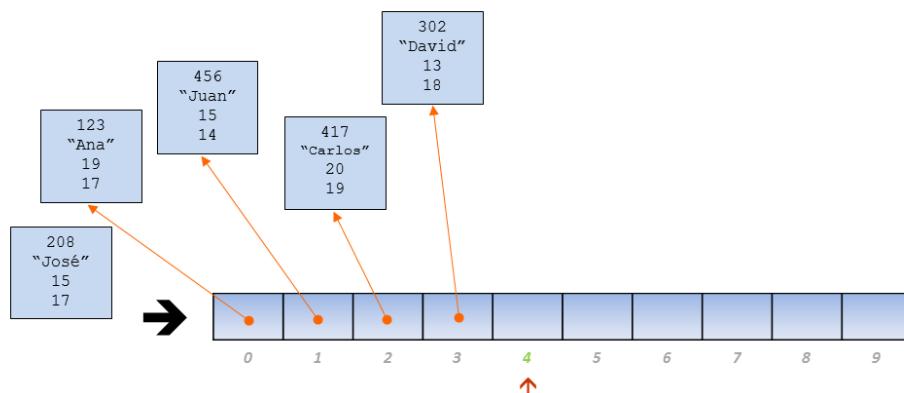
Ej:
`alu.set(3, new Alumno(417, "Carlos", 20, 19));`



```
3) public void remove(int) {  
}
```

Retira del arreglo la dirección de memoria de la posición indicada.

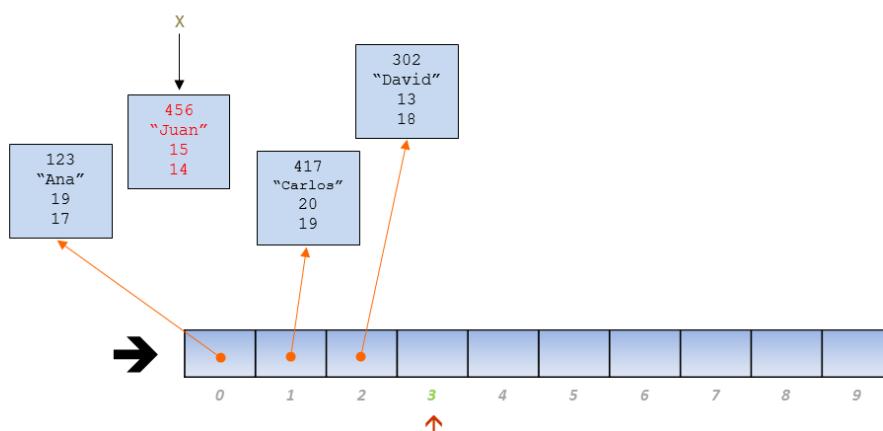
Ej:
alu.remove(0);



```
4) public void remove(Object) {  
}
```

Retira del arreglo la dirección de memoria del objeto referenciado.

Ej:
Alumno x = alu.get(1); // x tiene la DirMem del segundo alumno
alu.remove(x); // el segundo alumno ha sido retirado del arreglo

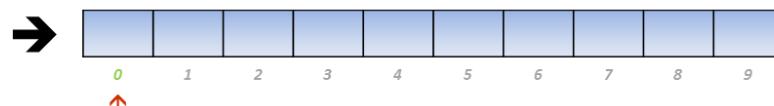


```
5) public void clear() {  
}
```

Retira del arreglo todas las direcciones de memoria.

Reinicializa el ArrayList (capacidad inicial: diez alumnos).

Ej:
alu.clear();



3.2.2. Operaciones públicas básicas

Ejemplo: método que adiciona una dirección de memoria al arreglo

```
public void adicionar(Alumno x) {
    alu.add(x);
}
```

Ejemplo: método que retira del arreglo la ultima dirección de memoria.

```
public void eliminarAlFinal() {
    alu.remove(tamaño()-1);
}
```

Ejemplo: método que retira del arreglo todas las direcciones de memoria.

```
public void eliminarTodo() {
    alu.clear();
}
```

Ejemplo: método que busca un código y retorna la dirección de memoria del objeto que lo contiene. En caso no exista retorna null.

```
public Alumno buscar(int codigo) {
    for (int i=0; i<tamaño(); i++)
        if (obtener(i).getCodigo() == codigo)
            return obtener(i);
    return null;
}
```

Ejemplo: método que recibe la dirección de memoria de un objeto Alumno y lo retira del ArrayList.

```
public void eliminar(Alumno x) {
    alu.remove(x);
}
```

Ejercicio 29: Implemente la clase ArregloAlumnos en el paquete arreglo con el atributo privado:

- ArrayList **alu** de tipo Alumno.

Implemente como públicos:

- Un constructor que cree el *ArrayList* y autogenere ocho objetos.
- Un método **tamaño** que retorne la cantidad de elementos registrados hasta ese momento.
- Un método **obtener** que reciba la posición y retorne la dirección de memoria del alumno respectivo.
- Un método **adicionar** que reciba la dirección de memoria de un nuevo alumno y lo adicione al *ArrayList*.
- Un método **eliminarAlFinal** que retire del *ArrayList* la última dirección de memoria.
- Un método **eliminarTodo** que retire del *ArrayList* todas las direcciones de memoria.
- Un método **buscar** que busque un código y retorne la dirección de memoria del objeto que lo contiene. En caso no exista retorne null.
- Un método **eliminar** que reciba la dirección de memoria de un objeto Alumno y lo retire del *ArrayList*.

```
package arreglo;

import clase.Alumno;

import java.util.ArrayList;

public class ArregloAlumnos {
    // Atributo privado
    private ArrayList <Alumno> alu;

    // Constructor
    public ArregloAlumnos() {
        alu = new ArrayList <Alumno> ();
        alu.add(new Alumno(123, "Ana", 19, 17));
        alu.add(new Alumno(456, "Juan", 15, 14));
        alu.add(new Alumno(789, "Pedro", 19, 15));
        alu.add(new Alumno(302, "David", 13, 18));
        alu.add(new Alumno(208, "Carlos", 20, 19));
        alu.add(new Alumno(417, "Jorge", 12, 13));
        alu.add(new Alumno(208, "María", 15, 17));
        alu.add(new Alumno(820, "José", 11, 10));
    }

    // Operaciones públicas básicas
    public int tamaño() {
        return alu.size();
    }

    public Alumno obtener(int i) {
        return alu.get(i);
    }

    public void adicionar(Alumno x) {
        alu.add(x);
    }

    public void eliminarAlFinal() {
        alu.remove(tamaño()-1);
    }

    public void eliminarTodo() {
        alu.clear();
    }

    // Operaciones públicas complementarias
    public Alumno buscar(int codigo) {
        for (int i=0; i<tamaño(); i++)
            if (obtener(i).getCodigo() == codigo)
                return obtener(i);

        return null;
    }

    public void eliminar(Alumno x) {
        alu.remove(x);
    }
}
```

Ejercicio 30: En el programa principal declare y cree como variable global un objeto de tipo ArregloAlumnos e implementa el método listar que visualice todos los alumnos ingresados.

Implemente la pulsación de los botones:

- **Adicionar** : envíe al método adicionar un nuevo alumno creado, validando que el código no se repita.
- **Eliminar al final** : elimine al último alumno.
- **Eliminar todo** : elimine a todos los alumnos.
- **Eliminar por código** : busque un código y si existe retire el objeto del arreglo.

```
package gui;

import clase.Arreglo;
import arreglo.ArregloAlumnos;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JScrollPane;
import javax.swing.JOptionPane;
import javax.swing.SwingConstants;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Ejemplo extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;

    private JPanel contentPane;
    private JLabel lblCodigo;
    private JLabel lblNombre;
    private JLabel lblNota1;
    private JLabel lblNota2;
    private JTextField txtCodigo;
    private JTextField txtNombre;
    private JTextField txtNota1;
    private JTextField txtNota2;
    private JButton btnAdicionar;
    private JButton btnEliminarAlFinal;
    private JButton btnEliminarTodo;
    private JButton btnEliminarPorCodigo;
    private JScrollPane scrollPane;
    private JTable tblTabla;
    private DefaultTableModel modelo;
    /**
```

```
* Launch the application.
*/
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Ejemplo frame = new Ejemplo();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
* Create the frame.
*/
public Ejemplo() {
    setTitle("Ejemplo - Semana_10");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 630, 370);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    lblCodigo = new JLabel(" Código");
    lblCodigo.setBounds(10, 11, 40, 28);
    contentPane.add(lblCodigo);

    txtCodigo = new JTextField();
    txtCodigo.setBounds(54, 11, 40, 28);
    contentPane.add(txtCodigo);
    txtCodigo.setColumns(10);

    lblNombre = new JLabel("Nombre");
    lblNombre.setBounds(120, 11, 50, 28);
    contentPane.add(lblNombre);

    txtNombre = new JTextField();
    txtNombre.setBounds(170, 11, 60, 28);
    contentPane.add(txtNombre);
    txtNombre.setColumns(10);

    lblNota1 = new JLabel("Nota 1");
    lblNota1.setBounds(255, 11, 40, 28);
    contentPane.add(lblNota1);

    txtNota1 = new JTextField();
    txtNota1.setBounds(295, 11, 40, 28);
    contentPane.add(txtNota1);
    txtNota1.setColumns(10);

    lblNota2 = new JLabel("Nota 2");
    lblNota2.setHorizontalAlignment(SwingConstants.RIGHT);
    lblNota2.setBounds(358, 11, 40, 28);
    contentPane.add(lblNota2);
```

```
txtNota2 = new JTextField();
txtNota2.setBounds(402, 11, 40, 28);
contentPane.add(txtNota2);
txtNota2.setColumns(10);

btnAdicionar = new JButton("Adicionar");
btnAdicionar.addActionListener(this);
btnAdicionar.setBounds(450, 50, 150, 23);
contentPane.add(btnAdicionar);

btnEliminarAlFinal = new JButton("Eliminar al final");
btnEliminarAlFinal.setBounds(450, 75, 150, 23);
btnEliminarAlFinal.addActionListener(this);
contentPane.add(btnEliminarAlFinal);

btnEliminarTodo = new JButton("Eliminar todo");
btnEliminarTodo.setBounds(450, 100, 150, 23);
btnEliminarTodo.addActionListener(this);
contentPane.add(btnEliminarTodo);

btnEliminarPorCodigo = new JButton("Eliminar por código");
btnEliminarPorCodigo.addActionListener(this);
btnEliminarPorCodigo.setBounds(450, 125, 150, 23);
contentPane.add(btnEliminarPorCodigo);

scrollPane = new JScrollPane();
scrollPane.setBounds(10, 50, 432, 274);
contentPane.add(scrollPane);

tblTabla = new JTable();
tblTabla.setFillsViewportHeight(true);
scrollPane.setViewportView(tblTabla);

modelo = new DefaultTableModel();
modelo.addColumn("código");
modelo.addColumn("nombre");
modelo.addColumn("nota 1");
modelo.addColumn("nota 2");
modelo.addColumn("promedio");
tblTabla.setModel(modelo);

listar();
}

public void actionPerformed(ActionEvent arg0) {
    if (arg0.getSource() == btnEliminarPorCodigo) {
        actionPerformedBtnEliminarPorCodigo(arg0);
    }
    if (arg0.getSource() == btnEliminarTodo) {
        actionPerformedBtnEliminarTodo(arg0);
    }
    if (arg0.getSource() == btnEliminarAlFinal) {
        actionPerformedBtnEliminarAlFinal(arg0);
    }
    if (arg0.getSource() == btnAdicionar) {
        actionPerformedBtnAdicionar(arg0);
    }
}
```

```
// Declaración global
ArregloAlumnos aa = new ArregloAlumnos();

protected void actionPerformedBtnAdicionar(ActionEvent arg0) {
    /**
     * Adiciona un nuevo alumno creado validando
     * que el código no se repita */
    try {
        int codigo = leerCodigo();
        if (aa.buscar(codigo) == null) {
            String nombre = leerNombre();
            if (nombre.length() > 0)
                try {
                    int nota1 = leerNota1();
                    try {
                        int nota2 = leerNota2();
                        Alumno nuevo = new Alumno(codigo, nombre,
                            nota1, nota2);
                        aa.adicionar(nuevo);
                        listar();
                        limpiza();
                    } catch (Exception e) {
                        mensaje("ingrese NOTA 2 correcta");
                        txtNota2.setText("");
                        txtNota2.requestFocus();
                    }
                } catch (Exception e) {
                    mensaje("ingrese NOTA 1 correcta");
                    txtNota1.setText("");
                    txtNota1.requestFocus();
                } else {
                    mensaje("ingrese NOMBRE correcto");
                    txtNombre.setText("");
                    txtNombre.requestFocus();
                }
        } else {
            mensaje("el CODIGO ya existe");
            txtCodigo.setText("");
            txtCodigo.requestFocus();
        }
    } catch (Exception e) {
        mensaje("ingrese CODIGO correcto");
        txtCodigo.setText("");
        txtCodigo.requestFocus();
    }
}

protected void actionPerformedBtnEliminarAlFinal(ActionEvent arg0) {
    /**
     * Elimina al último alumno */
    if (aa.tamaño() > 0) {
        aa.eliminarAlFinal();
        listar();
    }
    else
        mensaje("el ArrayList de alumnos está vacío");
    limpiza();
}

protected void actionPerformedBtnEliminarTodo(ActionEvent arg0) {
```

```
/** Elimina a todos los alumnos */
if (aa.tamaño() > 0) {
    aa.eliminarTodo();
    listar();
}
else
    mensaje("el ArrayList de alumnos está vacío");
limpieza();
}

protected void actionPerformedBtnEliminarPorCodigo(ActionEvent arg0){
    /** Busca un código y si existe retira el objeto del ArrayList */
    try {
        int codigo = leerCodigo();
        Alumno a = aa.buscar(codigo);
        if (a == null)
            mensaje("el CODIGO no existe");
        else {
            aa.eliminar(a);
            listar();
        }
        txtCodigo.setText("");
        txtCodigo.requestFocus();
    }
    catch (Exception e) {
        mensaje("ingrese CODIGO correcto");
        txtCodigo.setText("");
        txtCodigo.requestFocus();
    }
}

// Métodos tipo void sin parámetros
void limpieza() {
    txtCodigo.setText("");
    txtNombre.setText("");
    txtNota1.setText("");
    txtNota2.setText("");
    txtCodigo.requestFocus();
}

void listar() {
    modelo.setRowCount(0);
    for (int i=0; i<aa.tamaño(); i++) {
        Object fila[] = { aa.obtener(i).getCodigo(),
                          aa.obtener(i).getNombre(),
                          aa.obtener(i).getNota1(),
                          aa.obtener(i).getNota2(),
                          aa.obtener(i).promedio() };
        modelo.addRow(fila);
    }
}

// Métodos tipo void con parámetros
void mensaje(String s) {
    JOptionPane.showMessageDialog(this, s);
}

// Métodos que retornan valor sin parámetros
```

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

String leerNombre() {
    return txtNombre.getText().trim();
}

int leerNota1() {
    return Integer.parseInt(txtNota1.getText().trim());
}

int leerNota2() {
    return Integer.parseInt(txtNota2.getText().trim());
}

}
```

Ejemplo - Semana_10

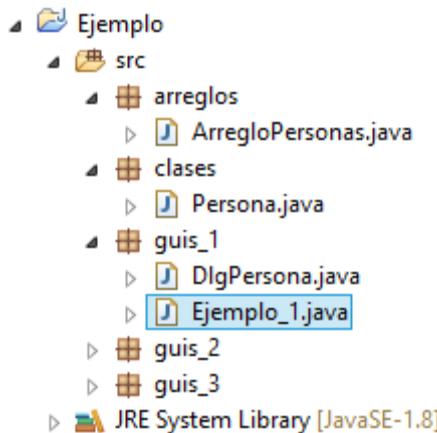
Código	Nombre	Nota 1	Nota 2	Promedio
123	Ana	19	17	18.0
456	Juan	15	14	14.5
789	Pedro	19	15	17.0
302	David	13	18	15.5
417	Carlos	20	19	19.5
641	Jorge	12	13	12.5
208	Maria	15	17	16.0
820	José	11	10	10.5

Operaciones:

- Adicionar
- Eliminar al final
- Eliminar todo
- Eliminar por código

3.3. Mantenimiento

3.3.1. Diseño básico de un Proyecto



Ejercicio 31: Diseñe la clase Persona en el paquete `clases` con los atributos privados: `codigo` (int), `nombre` (String), `dni` (String), `peso` (double), `estatura` (double) y `estado` (int).

Implemente además

- Un constructor que inicialice a todos los atributos.
- Métodos de acceso público `set` para todos los atributos privados. Use la referencia `this`.
- Métodos de acceso público `get` para todos los atributos privados.
- Método público que retorne el índice de masa corporal:
 $imc = peso / (estatura * estatura)$

```
package clases;

public class Persona {
    // Atributos privados
    private int codigo, estado;
    private String nombre, dni;
    private double peso, estatura;

    // Constructor
    public Persona(int codigo, String nombre, String dni,
                  double peso, double estatura, int estado) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.dni = dni;
        this.peso = peso;
        this.estatura = estatura;
        this.estado = estado;
    }

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

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

```
public void setDni(String dni) {
    this.dni = dni;
}

public void setPeso(double peso) {
    this.peso = peso;
}

public void setEstatura(double estatura) {
    this.estatura = estatura;
}

public void setEstado(int estado) {
    this.estado = estado;
}

public int getCodigo() {
    return codigo;
}

public String getNombre() {
    return nombre;
}

public String getDni() {
    return dni;
}

public double getPeso() {
    return peso;
}

public double getEstatura() {
    return estatura;
}

public int getEstado() {
    return estado;
}

// Operaciones públicas complementarias
public double imc() {
    return peso / (estatura * estatura);
}

}
```

Ejercicio 32: Implemente la clase ArregloPersonas en el paquete arreglos con el atributo privado:

- *ArrayList* **per** de tipo Persona.

Implemente como públicos:

- Un constructor que cree el *ArrayList* y autogeneré cinco objetos.
- Un método **adicionar** que reciba la dirección de memoria de una nueva persona y la adicione al *ArrayList*.
- Un método **tamaño** que retorne la cantidad de elementos registrados hasta ese momento.

- Un método **obtener** que reciba la posición y retorne la dirección de memoria de la persona respectiva.
- Un método **eliminar** que reciba la dirección de memoria de un objeto Persona y lo retire del *ArrayList*.
- Un método **buscar** que busque un código y retorne la dirección de memoria del objeto que lo contiene. En caso no exista retorne null.
- Un método **buscar** que busque un dni y retorne la dirección de memoria del objeto que lo contiene. En caso no exista retorne null.
- Un método **codigoCorrelativo** que retorne un número entero autogenerado y correlativo empezando a partir de 10001.

```
package arreglos;

import clases.Persona;

import java.util.ArrayList;

public class ArregloPersonas {
    // Atributo privado
    private ArrayList <Persona> per;

    // Constructor
    public ArregloPersonas () {
        per = new ArrayList <Persona> ();
        adicionar(new Persona(1001, "Juan Prado Salazar", "07557853", 82.3, 1.75, 3));
        adicionar(new Persona(1002, "Pedro Romero Soto", "11002348", 79.5, 1.58, 1));
        adicionar(new Persona(1003, "Luis Pinto Garza", "62279345", 82.7, 1.83, 0));
        adicionar(new Persona(1004, "Daniel Rojas Saenz", "20977241", 80.2, 1.72, 2));
        adicionar(new Persona(1005, "Jorge Espinal Vega", "06377845", 75.9, 1.88, 1));
    }

    // Operaciones públicas básicas
    public void adicionar(Persona p) {
        per.add(p);
    }

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

    public Persona obtener(int pos) {
        return per.get(pos);
    }

    public void eliminar(Persona x) {
        per.remove(x);
    }

    public Persona buscar(int codigo) {
        Persona x;
        for (int i=0; i<tamaño(); i++) {
            x = obtener(i);
            if (x.getCodigo() == codigo)
                return x;
        }
        return null;
    }
}
```

```

public Persona buscar(String dni) {
    Persona x;
    for (int i=0; i<tamaño(); i++) {
        x = obtener(i);
        if (x.getDni().equals(dni))
            return x;
    }

    return null;
}

// Operaciones públicas complementarias
public int codigoCorrelativo() {
    if (tamaño() == 0)
        return 10001;
    else
        return obtener(tamaño()-1).getCodigo() + 1;
}

}

```

Ejercicio 33: Diseñe en la clase principal DlgPersona una GUI adecuada para realizar las operaciones básicas de todo proyecto: adicionar, consultar, modificar y eliminar. Realice las validaciones de ingreso respectivas. Considere la declaración global ArregloPersonas ap = new ArregloPersonas();

```

package guis_1;

import clases.Persona;
import arreglos.ArregloPersonas;

import java.awt.EventQueue;

import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumnModel;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class DlgPersona extends JDialog implements ActionListener {

    private JLabel lblCodigo;
    private JLabel lblNombre;
    private JLabel lblDni;
    private JLabel lblPeso;
    private JLabel lblEstatura;
    private JLabel lblEstadoCivil;
    private JTextField txtCodigo;

```

```
private JTextField txtNombre;
private JTextField txtDni;
private JTextField txtPeso;
private JTextField txtEstatura;
private JComboBox <String> cboEstadoCivil;
private JScrollPane scrollPane;
private JButton btnBuscar;
private JButton btnAdicionar;
private JButton btnConsultar;
private JButton btnModificar;
private JButton btnEliminar;
private JButton btnAceptar;
private JButton btnVolver;
private JTable tblPersona;
private DefaultTableModel modelo;

// Tipo de operación a procesar: Adicionar, Consultar,
// Modificar o Eliminar
private int tipoOperacion;

// Constantes para los tipos de operaciones
public final static int ADICIONAR = 0;
public final static int CONSULTAR = 1;
public final static int MODIFICAR = 2;
public final static int ELIMINAR = 3;

/**
 * Launch the application.
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                DlgPersona dialog = new DlgPersona();
                dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
                dialog.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
 * Create the dialog.
 */
public DlgPersona() {
    setResizable(false);
    setTitle("Mantenimiento | Persona");
    setBounds(100, 100, 800, 600);
    getContentPane().setLayout(null);

    lblCodigo = new JLabel("C\u00f3digo");
    lblCodigo.setBounds(10, 10, 110, 23);
    getContentPane().add(lblCodigo);

    lblNombre = new JLabel("Nombre");
    lblNombre.setBounds(10, 35, 70, 23);
    getContentPane().add(lblNombre);
```

```
lblDni = new JLabel("DNI");
lblDni.setBounds(10, 60, 70, 23);
getContentPane().add(lblDni);

lblPeso = new JLabel("Peso");
lblPeso.setBounds(10, 85, 70, 23);
getContentPane().add(lblPeso);

lblEstatura = new JLabel("Estatura");
lblEstatura.setBounds(10, 110, 70, 23);
getContentPane().add(lblEstatura);

lblEstadoCivil = new JLabel("Estado civil");
lblEstadoCivil.setBounds(10, 135, 86, 23);
getContentPane().add(lblEstadoCivil);

txtCodigo = new JTextField();
txtCodigo.setBounds(90, 10, 86, 23);
getContentPane().add(txtCodigo);
txtCodigo.setEditable(false);
txtCodigo.setColumns(10);

txtNombre = new JTextField();
txtNombre.setBounds(90, 35, 251, 23);
getContentPane().add(txtNombre);
txtNombre.setEditable(false);
txtNombre.setColumns(10);

txtDni = new JTextField();
txtDni.setBounds(90, 60, 86, 23);
getContentPane().add(txtDni);
txtDni.setEditable(false);
txtDni.setColumns(10);

txtPeso = new JTextField();
txtPeso.setBounds(90, 85, 50, 23);
getContentPane().add(txtPeso);
txtPeso.setEditable(false);
txtPeso.setColumns(10);

txtEstatura = new JTextField();
txtEstatura.setBounds(90, 110, 50, 23);
getContentPane().add(txtEstatura);
txtEstatura.setEditable(false);
txtEstatura.setColumns(10);

cboEstadoCivil = new JComboBox <String> ();
cboEstadoCivil.setModel(new DefaultComboBoxModel <String>
    (new String[] {"Soltero", "Casado",
        "Viudo", "Divorciado"}));
cboEstadoCivil.setBounds(90, 135, 86, 23);
getContentPane().add(cboEstadoCivil);
cboEstadoCivil.setEnabled(false);

scrollPane = new JScrollPane();
scrollPane.setBounds(10, 170, 775, 360);
getContentPane().add(scrollPane);
```

```
tblPersona = new JTable();
tblPersona.setFillsViewportHeight(true);
scrollPane.setViewportView(tblPersona);

modelo = new DefaultTableModel();
modelo.addColumn("CÓDIGO");
modelo.addColumn("NOMBRE");
modelo.addColumn("DNI");
modelo.addColumn("PESO (kg)");
modelo.addColumn("ESTATURA (mts)");
modelo.addColumn("ESTADO CIVIL");
modelo.addColumn("IMC = peso/estatura2");
tblPersona.setModel(modelo);

btnBuscar = new JButton("Buscar");
btnBuscar.addActionListener(this);
btnBuscar.setEnabled(false);
btnBuscar.setBounds(240, 10, 101, 23);
getContentPane().add(btnBuscar);

btnAdicionar = new JButton("Adicionar");
btnAdicionar.addActionListener(this);
btnAdicionar.setBounds(10, 540, 120, 23);
getContentPane().add(btnAdicionar);

btnConsultar = new JButton("Consultar");
btnConsultar.addActionListener(this);
btnConsultar.setBounds(135, 540, 120, 23);
getContentPane().add(btnConsultar);

btnModificar = new JButton("Modificar");
btnModificar.addActionListener(this);
btnModificar.setBounds(260, 540, 120, 23);
getContentPane().add(btnModificar);

btnEliminar = new JButton("Eliminar");
btnEliminar.addActionListener(this);
btnEliminar.setBounds(385, 540, 120, 23);
getContentPane().add(btnEliminar);

btnAceptar = new JButton("Aceptar");
btnAceptar.addActionListener(this);
btnAceptar.setEnabled(false);
btnAceptar.setBounds(510, 540, 120, 23);
getContentPane().add(btnAceptar);

btnVolver = new JButton("Volver");
btnVolver.addActionListener(this);
btnVolver.setEnabled(false);
btnVolver.setBounds(664, 540, 120, 23);
getContentPane().add(btnVolver);

ajustarAnchoColumnas();
listar();
}

// Declaración global
ArregloPersonas ap = new ArregloPersonas();
```

```
public void actionPerformed(ActionEvent arg0) {  
    if (arg0.getSource() == btnBuscar) {  
        actionPerformedBtnBuscar(arg0);  
    }  
  
    if (arg0.getSource() == btnVolver) {  
        actionPerformedBtnVolver(arg0);  
    }  
  
    if (arg0.getSource() == btnAceptar) {  
        actionPerformedBtnAceptar(arg0);  
    }  
  
    if (arg0.getSource() == btnEliminar) {  
        actionPerformedBtnEliminar(arg0);  
    }  
  
    if (arg0.getSource() == btnModificar) {  
        actionPerformedBtnModificar(arg0);  
    }  
  
    if (arg0.getSource() == btnConsultar) {  
        actionPerformedBtnConsultar(arg0);  
    }  
  
    if (arg0.getSource() == btnAdicionar) {  
        actionPerformedBtnAdicionar(arg0);  
    }  
}  
  
protected void actionPerformedBtnAdicionar(ActionEvent arg0) {  
    tipoOperacion = ADICIONAR;  
  
    txtCodigo.setText(" " + ap.codigoCorrelativo());  
    txtNombre.setText("");  
    txtDni.setText("");  
    txtPeso.setText("");  
    txtEstatura.setText("");  
    habilitarEntradas(true);  
    habilitarBotones(false);  
    txtNombre.requestFocus();  
}  
  
protected void actionPerformedBtnConsultar(ActionEvent arg0) {  
    tipoOperacion = CONSULTAR;  
  
    txtCodigo.setEditable(true);  
    habilitarBotones(false);  
    txtCodigo.requestFocus();  
}  
  
protected void actionPerformedBtnModificar(ActionEvent arg0) {  
    tipoOperacion = MODIFICAR;  
  
    txtCodigo.setEditable(true);  
    habilitarEntradas(true);  
    habilitarBotones(false);  
    txtCodigo.requestFocus();  
}
```

```

protected void actionPerformedBtnEliminar(ActionEvent arg0) {
    tipoOperacion = ELIMINAR;
    txtCodigo.setEditable(true);
    habilitarBotones(false);
}

protected void actionPerformedBtnAceptar(ActionEvent arg0) {
    switch (tipoOperacion) {
        case ADICIONAR:
            adicionarPersona(); break;
        case CONSULTAR:
            consultarPersona(); break;
        case MODIFICAR:
            modificarPersona(); break;
        case ELIMINAR:
            eliminarPersona();
    }
}

protected void actionPerformedBtnBuscar(ActionEvent arg0) {
    consultarPersona();
}

protected void actionPerformedBtnVolver(ActionEvent arg0) {
    txtCodigo.setText("");
    txtNombre.setText("");
    txtDni.setText("");
    txtPeso.setText("");
    txtEstatura.setText("");
    txtCodigo.setEditable(false);
    habilitarEntradas(false);
    habilitarBotones(true);
}

// Métodos tipo void (sin parámetros)
void ajustarAnchoColumnas() {
    TableColumnModel tcm = tblPersona.getColumnModel();
    tcm.getColumn(0).setPreferredWidth(anchoColumna(10)); // código
    tcm.getColumn(1).setPreferredWidth(anchoColumna(20)); // nombre
    tcm.getColumn(2).setPreferredWidth(anchoColumna(10)); // dni
    tcm.getColumn(3).setPreferredWidth(anchoColumna(10)); // peso
    tcm.getColumn(4).setPreferredWidth(anchoColumna(15)); // estatura
    tcm.getColumn(5).setPreferredWidth(anchoColumna(15)); // estadoCivil
    tcm.getColumn(6).setPreferredWidth(anchoColumna(20)); // imc
}

void listar() {
    Persona x;
    modelo.setRowCount(0);
    for (int i=0; i<ap.tamaño(); i++) {
        x = ap.obtener(i);
        Object[] fila = {x.getCodigo(), x.getNombre(),
                        x.getDni(), x.getPeso(),
                        x.getEstatura(),
                        enTextoEstadoCivil(x.getEstado()),
                        x.getImc() };
        modelo.addRow(fila);
    }
}

```

```
void consultarPersona() {
    try {
        Persona x = ap.buscar(leerCodigo());

        if (x != null) {
            txtNombre.setText(x.getNombre());
            txtDni.setText(x.getDni());
            txtPeso.setText("" + x.getPeso());
            txtEstatura.setText("" + x.getEstatura());
            cboEstadoCivil.setSelectedIndex(x.getEstado());
            txtCodigo.requestFocus();
        } else
            error("El código " + leerCodigo() + " no existe", txtCodigo);
    } catch (Exception e) {
        error("Ingrese CÓDIGO correcto", txtCodigo);
    }
}

void adicionarPersona() {
    int codigo = leerCodigo();
    String nombre = leerNombre();

    if (nombre.length() > 0) {
        String dni = leerDni();

        if (ap.buscar(dni) == null)
            try {
                double peso = leerPeso();

                try {
                    double estatura = leerEstatura();
                    int estado = leerPosEstado();

                    Persona nueva = new Persona(codigo, nombre,
                        dni, peso,
                        estatura, estado);
                    ap.adicionar(nueva);
                    listar();

                    txtCodigo.setText("" + ap.codigoCorrelativo());
                    txtNombre.setText("");
                    txtDni.setText("");
                    txtPeso.setText("");
                    txtEstatura.setText("");
                    txtNombre.requestFocus();
                } catch (Exception e) {
                    error("Ingrese ESTATURA correcta", txtEstatura);
                }
            } catch (Exception e) {
                error("Ingrese PESO correcto", txtPeso);
            }
        else
            error("Ingrese DNI correcto", txtDni);
    } else
        error("Ingrese NOMBRE correcto", txtNombre);
}
```

```
void eliminarPersona() {
    try {
        int codigo = leerCodigo();
        Persona x = ap.buscar(codigo);

        if (x != null) {
            ap.eliminar(x);
            listar();
            txtCodigo.setText("");
            txtNombre.setText("");
            txtDni.setText("");
            txtPeso.setText("");
            txtEstatura.setText("");
            txtCodigo.requestFocus();
        } else
            error("El código " + codigo + " no existe", txtCodigo);
    } catch (Exception e) {
        error("Ingrese CÓDIGO correcto", txtCodigo);
    }
}

void modificarPersona() {
    try {
        Persona x = ap.buscar(leerCodigo());
        String nombre = leerNombre();

        if (nombre.length() > 0) {
            String dni = leerDni();
            Persona y = ap.buscar(dni);

            if (y == null || x.equals(y))
                try {
                    double peso = leerPeso();
                    try {
                        double estatura = leerEstatura();
                        int estado = leerPosEstado();

                        x.setNombre(nombre);
                        x.setDni(dni);
                        x.setPeso(peso);
                        x.setEstatura(estatura);
                        x.setEstado(estado);

                        listar();
                        txtCodigo.requestFocus();
                    } catch (Exception e) {
                        error("Ingrese ESTATURA correcta", txtEstatura);
                    }
                } catch (Exception e) {
                    error("Ingrese PESO correcto", txtPeso);
                }
            else
                error("Ingrese DNI correcto", txtDni);
        } else
            error("Ingrese NOMBRE correcto", txtNombre);
    } catch (Exception e) {
        error("Ingrese CÓDIGO correcto", txtCodigo);
    }
}
```

```
// Métodos tipo void (con parámetros)
void error(String s, JTextField txt) {
    mensaje(s);
    txt.setText("");
    txt.requestFocus();
}

void habilitarBotones(boolean sino) {
    if (tipoOperacion != ADICIONAR)
        btnBuscar.setEnabled(!sino);

    btnAdicionar.setEnabled(sino);
    btnConsultar.setEnabled(sino);
    btnModificar.setEnabled(sino);
    btnEliminar.setEnabled(sino);

    if (tipoOperacion == CONSULTAR)
        btnAceptar.setEnabled(false);
    else
        btnAceptar.setEnabled(!sino);

    btnVolver.setEnabled(!sino);
}

void habilitarEntradas(boolean sino) {
    txtNombre.setEditable(sino);
    txtDni.setEditable(sino);
    txtPeso.setEditable(sino);
    txtEstatura.setEditable(sino);
    cboEstadoCivil.setEnabled(sino);
}

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

// Métodos que retornan valor (sin parámetros)
int leerCodigo() {
    return Integer.parseInt(txtCodigo.getText().trim());
}

String leerNombre() {
    return txtNombre.getText().trim();
}

String leerDni() {
    return txtDni.getText().trim();
}

double leerPeso() {
    return Double.parseDouble(txtPeso.getText().trim());
}

double leerEstatura() {
    return Double.parseDouble(txtEstatura.getText().trim());
}
```

```

int leerPosEstado() {
    return cboEstadoCivil.getSelectedIndex();
}

// Métodos que retornan valor (con parámetros)
int anchoColumna(int porcentaje) {
    return porcentaje * scrollPane.getWidth() / 100;
}

String enTextoEstadoCivil(int i) {
    return cboEstadoCivil.getItemAt(i);
}
}

```

Mantenimiento | Persona

Código	<input type="text"/>	<input type="button" value="Buscar"/>				
Nombre	<input type="text"/>					
DNI	<input type="text"/>					
Peso	<input type="text"/>					
Estatura	<input type="text"/>					
Estado civil	<input type="button" value="Soltero"/>	<input type="button" value="▼"/>				
CÓDIGO	NOMBRE	DNI	PESO (kg)	ESTATURA (mts)	ESTADO CIVIL	IMC = peso/estatura ²
1001	Juan Prado Salazar	07557853	82.3	1.75	Divorciado	26.8734693877551
1002	Pedro Romero Soto	11002348	79.5	1.58	Casado	31.84585803557122
1003	Luis Pinto Garza	62279345	82.7	1.83	Soltero	24.69467586371644
1004	Daniel Rojas Saenz	20977241	80.2	1.72	Viudo	27.109248242293134
1005	Jorge Espinal Vega	06377845	75.9	1.88	Casado	21.47464916251698



TÉCNICAS AVANZADAS DE POO

LOGRO DE LA UNIDAD DE APRENDIZAJE

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

TEMARIO

4.1. Tema 12 : Herencia y Polimorfismos

- 4.1.1 : Generalización / especialización
- 4.1.2 : Herencia
- 4.1.3 : Relación *es-un* o *es-una*
- 4.1.4 : Uso de *super*
- 4.1.5 : Sobrescritura de métodos
- 4.1.6 : Clases abstractas y métodos abstractos
- 4.1.7 : Técnicas de casting
- 4.1.8 : Polimorfismo y uso de “instanceof”

4.2. Tema 13 : Interfaces

- 4.2.1 : Definición
- 4.2.2 : Herencia múltiple

ACTIVIDADES PROPUESTAS

- Entender el concepto de Herencia (relación es-un).
- Emplear el modificador *protected*.
- Implementar clases abstractas
- Aplicar técnicas de casting

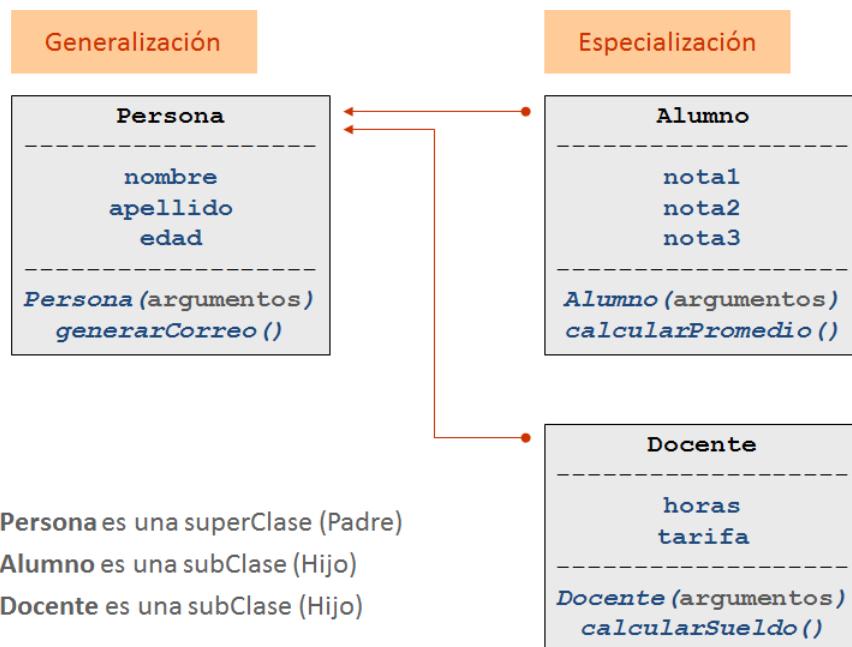
4.1. Herencia y Polimorfismo

4.1.1. 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 *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 clases Hijo heredan características de la clase Padre y añaden características específicas que las diferencian.

4.1.2. Herencia

Es el mecanismo mediante el cual se puede definir una clase (subclase) en base a 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 encuentre en el mismo paquete a la vez que impide el acceso al miembro desde el exterior de la superclase.

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

```
public class Hijo extends Padre {
    // Cuerpo de la subclase
    ...
}

public class Padre {
    // Cuerpo de la superclase
    ...
}
```

4.1.3. Relación “es-un” o “es-una”

La herencia permite establecer una jerarquía de especialización mediante la relación “**es-un**” o “**es-una**”. Ejemplo:

un Mamífero **es un** Animal
 un Ave **es un** Animal
 una Vaca **es un** Mamífero
 un Pato **es un** Ave

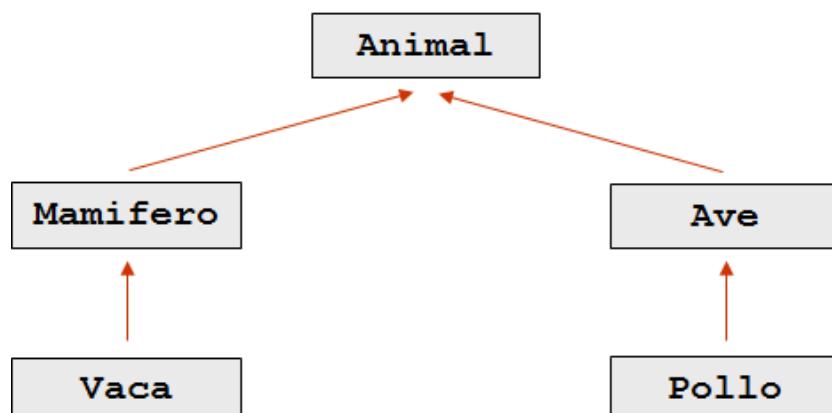
```
public class Animal {
    ...
}

public class Mamifero extends Animal {
    ...
}

public class Ave extends Animal {
    ...
}

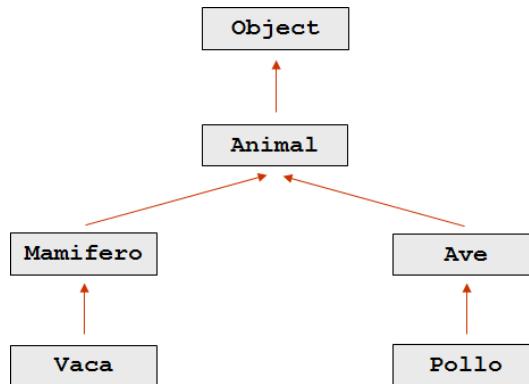
public class Vaca extends Mamifero {
    ...
}

public class Pollo extends Ave {
    ...
}
```



Si una clase no tiene una clase **Padre** explícita, entonces implícitamente su clase **Padre** es la clase **Object**. Así, en el caso de la clase **Animal**, implícitamente figura como:

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



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

4.1.4. Uso de **super**

El Constructor de la clase **Padre** puede invocarse desde la clase **Hijo** utilizando la palabra **super** de la siguiente forma:

```
super(argumentos);
```

Esta instrucción tiene que ser la primera sentencia dentro del constructor de la clase **Hijo**.

Ejercicio 34: Cree la clase **Persona** en el package padre con los atributos protegidos: nombre (cadena), apellido (cadena) y edad (entero). A través de un constructor inicialice sus atributos. Implemente un método que autogenera un correo electrónico y un método que retorne en una cadena los atributos protegidos.

```
package padre;

public class Persona {

    // Atributos protegidos
    protected String nombre, apellido;
    protected int edad;

    // Constructor
    public Persona(String nombre, String apellido, int edad) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
    }
}
```

```

// Operaciones
public String generarCorreo() {
    return nombre + "." + apellido + "@cibertec.edu.pe";
}

public String datosDeLaPersona() {
    return "Nombre : " + nombre + "\n" +
           "Apellido : " + apellido + "\n" +
           "Edad : " + edad;
}
}

```

Ejercicio 35: Cree la clase Alumno en el package hijo que herede de la clase Persona. Considere como atributos privados tres notas enteras. A través de su constructor reciba la data del alumno y derive a la clase Persona la información que corresponda. Implemente un método que retorne el promedio del alumno y un método que retorne los datos completos del alumno (heredando los datos de la clase Persona).

```

package hijo;

import padre.Persona;

public class Alumno extends Persona {
    // Atributos privados
    private int nota1, nota2, nota3;

    // Constructor
    public Alumno(String nombre, String apellido, int edad,
                  int nota1, int nota2, int nota3) {
        super(nombre, apellido, edad);
        this.nota1 = nota1;
        this.nota2 = nota2;
        this.nota3 = nota3;
    }

    // Operaciones
    public double calcularPromedio() {
        return (nota1 + nota2 + nota3) / 3.0;
    }

    public String datosCompletos() {
        return datosDeLaPersona() + "\n" +
               "Nota1 : " + nota1 + "\n" +
               "Nota2 : " + nota2 + "\n" +
               "Nota3 : " + nota3;
    }
}

```

Ejercicio 36: Cree la clase Docente en el package hijo que herede de la clase Persona. Considere como atributos privados horas trabajadas (entero) y tarifa por hora (real). A través de su constructor reciba la data del docente y derive a la clase Persona la información respectiva. Implemente un método que retorne el sueldo del docente y un método que retorne los datos completos del docente (heredando los datos de la clase Persona).

```
package hijo;

import padre.Persona;

public class Docente extends Persona {
    // Atributos privados
    private int horas;
    private double tarifa;

    // Constructor
    public Docente(String nombre, String apellido, int edad,
                   int horas, double tarifa) {
        super(nombre, apellido, edad);
        this.horas = horas;
        this.tarifa = tarifa;
    }

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

    public String datosCompletos() {
        return datosDeLaPersona() + "\n" +
            "Horas : " + horas + "\n" +
            "Tarifa : " + tarifa;
    }
}
```

Ejercicio 37: En el programa principal declare y cree un objeto de cada clase con datos fijos y aplicando sobrecarga de métodos, implemente tres métodos listado, uno para cada tipo de objeto.

```
package cibertec;

import padre.Persona;
import hijo.*;
...
public class Programa extends JFrame implements ActionListener {
    ...
    protected void actionPerformedBtnProcesar(ActionEvent arg0) {
        Persona p = new Persona("Juan", "Matos", 18);
        listado(p);

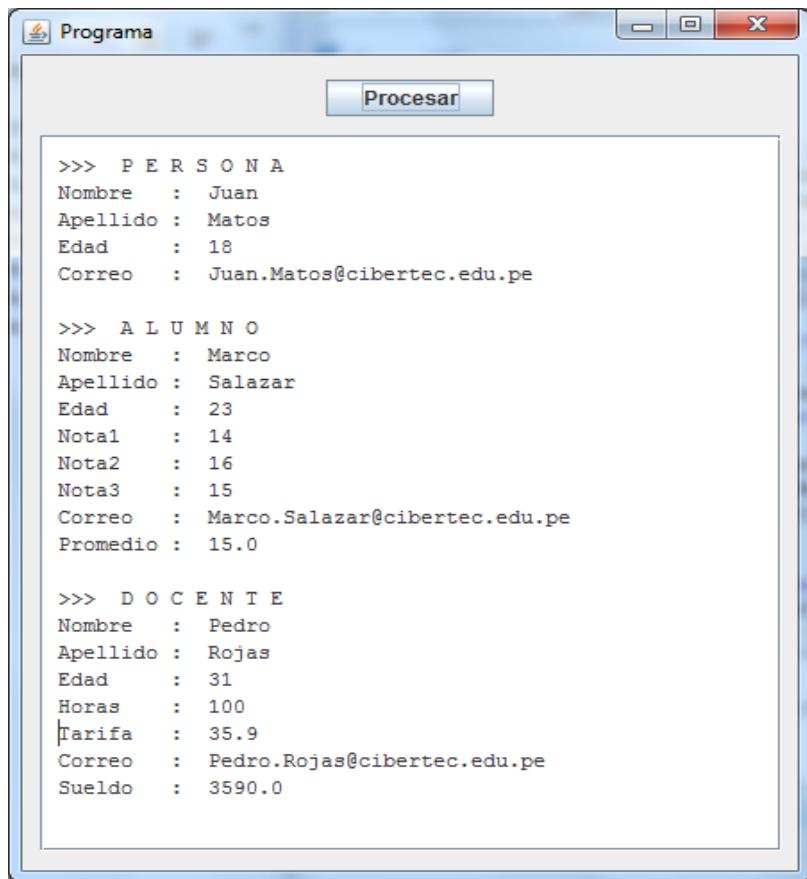
        Alumno a = new Alumno("Marco", "Salazar", 23, 14, 16, 15);
        listado(a);

        Docente d = new Docente("Pedro", "Rojas", 31, 100, 35.9);
        listado(d);
    }

    // Métodos tipo void (con parámetros)
    void listado(Persona x) {
        imprimir("">>>> P E R S O N A");
        imprimir(x.datosDeLaPersona());
        imprimir("Correo : " + x.generarCorreo());
        imprimir("");
    }

    void listado(Alumno x) {
        imprimir("">>>> A L U M N O");
        imprimir(x.datosCompletos());
        imprimir("Correo : " + x.generarCorreo());
        imprimir("Promedio : " + x.calcularPromedio());
        imprimir("");
    }

    void listado(Docente x) {
        imprimir("">>>> D O C E N T E");
        imprimir(x.datosCompletos());
        imprimir("Correo : " + x.generarCorreo());
        imprimir("Sueldo : " + x.calcularSueldo());
    }
    ...
}
```



4.1.5. Sobrescritura de métodos

Una clase **Hijo** puede añadir nuevos miembros e incluso puede redefinir miembros de la clase **Padre**. 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**.

Se utiliza **super** también para acceder desde la Clase **Hijo** a un método de la Clase **Padre**, en caso que la Clase **Hijo** cuente con un método igual en sintaxis.

Ejercicio 38: Cambie de nombre al método **datosDeLaPersona** de la clase **Persona** por el de **datosCompletos**.

```

package padre;

public class Persona {
    ...
    public String datosCompletos() {
        return "Nombre : " + nombre + "\n" +
            "Apellido : " + apellido + "\n" +
            "Edad : " + edad;
    }
}

```

Ejercicio 39: Utilice super en la clase Alumno para hacer que el método ***datosCompletos*** de dicha clase ejecute el método ***datosCompletos*** de la clase Persona.

```
package hijo;

import padre.Persona;

public class Alumno extends Persona {
    ...
    public String datosCompletos() {
        return super.datosCompletos() + "\n" +
            "Nota1 : " + nota1 + "\n" +
            "Nota2 : " + nota2 + "\n" +
            "Nota3 : " + nota3;
    }
}
```

Ejercicio 40: Utilice super en la clase Docente para hacer que el método ***datosCompletos*** de dicha clase ejecute el método ***datosCompletos*** de la clase Persona.

```
package hijo;

import padre.Persona;

public class Docente extends Persona {
    ...
    public String datosCompletos() {
        return super.datosCompletos() + "\n" +
            "Nota1 : " + nota1 + "\n" +
            "Nota2 : " + nota2 + "\n" +
            "Nota3 : " + nota3;
    }
}
```

Ejercicio 41: En el programa principal ajuste el método listado que recibe un objeto de tipo Persona dado el cambio de nombre del método ***datosDeLaPersona*** por el de ***datosCompletos***.

```
package cibertec;

import padre.Persona;
import hijo.*;

...
public class Programa extends JFrame implements ActionListener {
    ...
    protected void actionPerformedBtnProcesar(ActionEvent arg0) {
        Persona p = new Persona("Juan", "Matos", 18);
        listado(p);

        Alumno a = new Alumno("Marco", "Salazar", 23, 14, 16, 15);
        listado(a);

        Docente d = new Docente("Pedro", "Rojas", 31, 100, 35.9);
        listado(d);
    }
}
```

```

// Mètodes tipo void (con paràmetros)
void listado(Persona x) {
    imprimir("">>>> P E R S O N A");
    imprimir(x.datosCompletos());
    imprimir("Correo : " + x.generarCorreo());
    imprimir("");
}

void listado(Alumno x) {
    imprimir("">>>> A L U M N O");
    imprimir(x.datosCompletos());
    imprimir("Correo : " + x.generarCorreo());
    imprimir("Promedio : " + x.calcularPromedio());
    imprimir("");
}

void listado(Docente x) {
    imprimir("">>>> D O C E N T E");
    imprimir(x.datosCompletos());
    imprimir("Correo : " + x.generarCorreo());
    imprimir("Sueldo : " + x.calcularSueldo());
}
...
}

```

4.1.6. Clases abstractas y métodos abstractos

Una Clase abstracta es aquella que no permite crear objetos de manera directa. Se usa únicamente para crear subClases. Una clase abstracta se etiqueta con la palabra reservada **abstract**.

Ejercicio 42: Convierta a la clase Persona en clase abstracta.

```

package padre;

public abstract class Persona {
    ...
}

➔ La clase Persona ya no puede crear objetos directamente.

```

Un método es abstracto cuando no tiene implementación y solamente se define con el objetivo de obligar a que en cada clase Hijo que deriva de la clase abstracta se realice la correspondiente implementación.

Un método abstracto se etiqueta con la palabra reservada **abstract**.

Si una Clase tiene por lo menos un método abstracto, entonces la clase tiene que ser abstracta, de lo contrario el compilador mostrará un mensaje de error.

Ejercicio 43: Anuncie en la clase abstracta Persona un método abstracto.

```
package padre;

public abstract class Persona {
    ...
    // Método abstracto
    public abstract String identificacion();
}
```

Ejercicio 44: En la clase Alumno implemente por obligación el método público *identificacion*.

```
package hijo;

import padre.Persona;

public class Alumno extends Persona {
    ...
    public String identificacion() {
        return ">>> A L U M N O";
    }
}
```

Ejercicio 45: En la clase Docente implemente por obligación el método público *identificacion*.

```
package hijo;

import padre.Persona;

public class Docente extends Persona {
    ...
    public String identificacion() {
        return ">>> D O C E N T E";
    }
}
```

Ejercicio 46: En los métodos listado del programa principal muestre la identificación del objeto recibido respectivamente.

```
package cibertec;

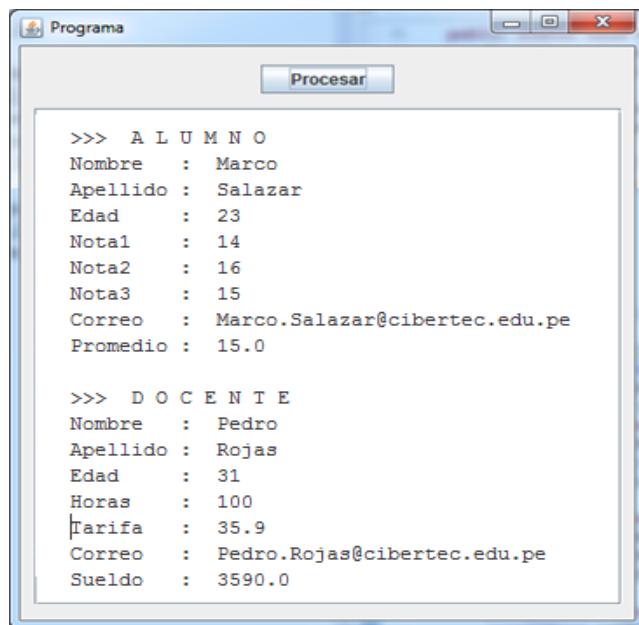
import padre.Persona;
import hijo.*;

...
public class Programa extends JFrame implements ActionListener {
    ...
    protected void actionPerformedBtnProcesar(ActionEvent arg0) {
        Alumno a = new Alumno("Marco", "Salazar", 23, 14, 16, 15);
        listado(a);

        Docente d = new Docente("Pedro", "Rojas", 31, 100, 35.9);
        listado(d);
    }
}
```

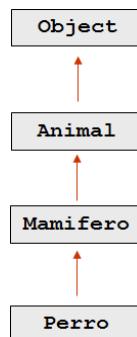
```
// Mètodos tipo void (con parámetros)
void listado(Alumno x) {
    imprimir(x.identificacion());
    imprimir(x.datosCompletos());
    imprimir("Correo : " + x.generarCorreo());
    imprimir("Promedio : " + x.calcularPromedio());
    imprimir("");
}

void listado(Docente x) {
    imprimir(x.identificacion());
    imprimir(x.datosCompletos());
    imprimir("Correo : " + x.generarCorreo());
    imprimir("Sueldo : " + x.calcularSueldo());
}
}
```



4.1.7. Técnicas de casting

El casting consiste en realizar conversiones de tipo. El casting no modifica al objeto sólo su tipo. Tiene sentido únicamente por la existencia de la herencia.



4.1.7.1. 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. Ejemplo:

```
Object oa = new Animal();
Object om = new Mamifero();
Object op = new Perro();
Animal am = new Mamifero();
Animal ap = new Perro();
Mamifero mp = new Perro();
```

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

public class Mamifero extends Animal {
    public String mensaje() {
        return "soy mamífero";
    }
}

public class Perro extends Mamifero {
    public String mensaje() {
        return "soy perro";
    }

    public String hacerRuido() {
        return "guau";
    }
}
```

oa, om, op de tipo *Object* no reconocen los métodos *mensaje()* ni *hacerRuido()* porque no aparecen en la Clase *Object*.

am, ap de tipo *Animal* reconocen al método *hacerRuido()*. El primero lo busca en la Clase *Mamifero*, pero al no encontrarlo lo ejecuta en la Clase *Animal*. El segundo lo busca y ejecuta en la Clase *Perro*.

mp de tipo *Mamifero* identifica a los métodos *mensaje()* y *hacerRuido()*, ambos se ejecutan en la Clase *Perro*.

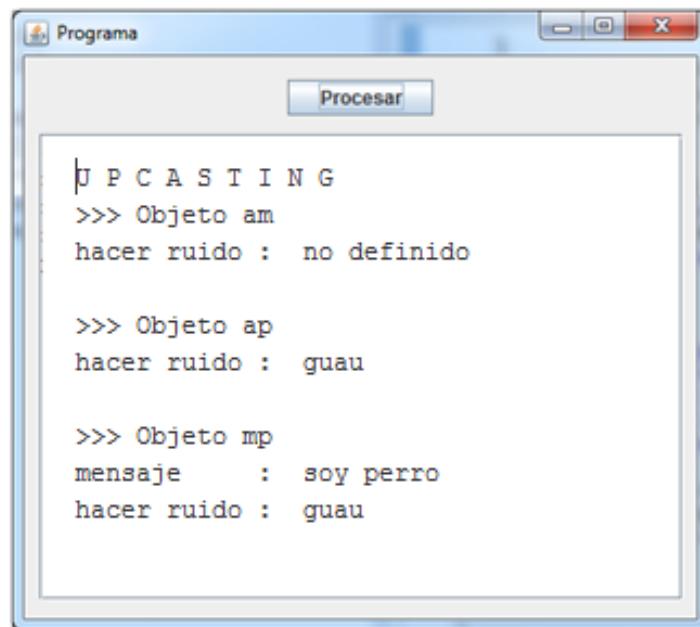
```
protected void actionPerformedBtnProcesar(ActionEvent arg0) {

    // Upcasting
    Object oa = new Animal();
    Object om = new Mamifero();
    Object op = new Perro();
    Animal am = new Mamifero();
    Animal ap = new Perro();
    Mamifero mp = new Perro();
```

```

imprimir("U P C A S T I N G");
imprimir(">>> Objeto am");
imprimir("hacer ruido : " + am.hacerRuido());
imprimir("");
imprimir(">>> Objeto ap");
imprimir("hacer ruido : " + ap.hacerRuido());
imprimir("");
imprimir(">>> Objeto mp");
imprimir("mensaje : " + mp.mensaje());
imprimir("hacer ruido : " + mp.hacerRuido());
}

```



4.1.7.2. 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. Ejemplo:

```

Animal    ao = (Animal) oa;
Mamifero  mo = (Mamifero) om;
Perro     po = (Perro) op;
Mamifero  ma = (Mamifero) am;
Perro     pa = (Perro) ap;
Perro     pm = (Perro) mp;

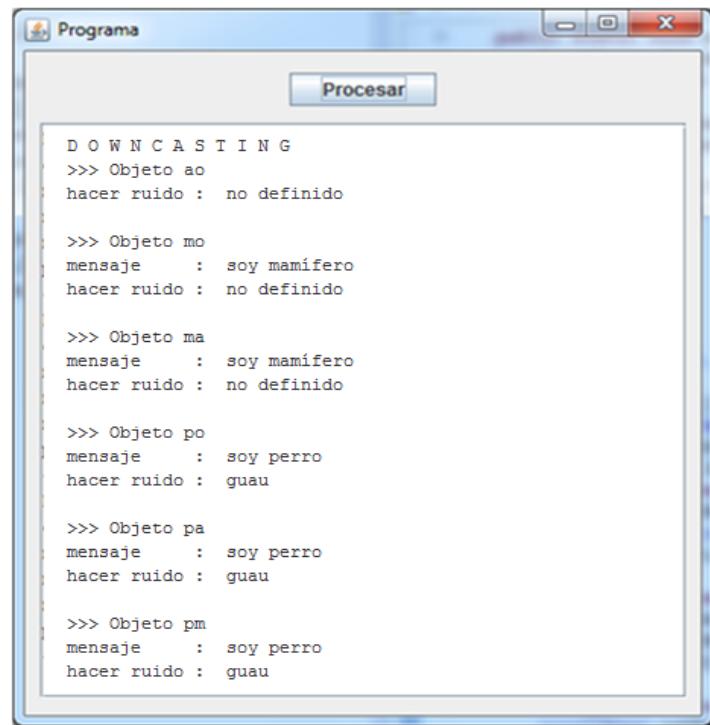
```

ao de tipo *Animal* reconoce al método *hacerRuido()*, lo busca en dicha Clase y lo ejecuta.

mo, ma de tipo *Mamifero* reconocen los métodos *mensaje()* y *hacerRuido()*, el primero lo ejecuta en la Clase *Mamífero* y el segundo en la Clase *Animal*.

po, pa, pm de tipo *Perro* reconocen los métodos *mensaje()* y *hacerRuido()*, ambos se ejecutan en la Clase *Perro*.

```
protected void actionPerformedBtnProcesar(ActionEvent arg0) {  
    // Downcasting  
    Animal ao = (Animal) oa;  
    Mamifero mo = (Mamifero) om;  
    Perro po = (Perro) op;  
    Mamifero ma = (Mamifero) am;  
    Perro pa = (Perro) ap;  
    Perro pm = (Perro) mp;  
  
    imprimir("D O W N C A S T I N G");  
    imprimir(">>> Objeto ao");  
    imprimir("hacer ruido : " + ao.hacerRuido());  
    imprimir("");  
    imprimir(">>> Objeto mo");  
    imprimir("mensaje : " + mo.mensaje());  
    imprimir("hacer ruido : " + mo.hacerRuido());  
    imprimir("");  
    imprimir(">>> Objeto ma");  
    imprimir("mensaje : " + ma.mensaje());  
    imprimir("hacer ruido : " + ma.hacerRuido());  
    imprimir("");  
    imprimir(">>> Objeto po");  
    imprimir("mensaje : " + po.mensaje());  
    imprimir("hacer ruido : " + po.hacerRuido());  
    imprimir("");  
    imprimir(">>> Objeto pa");  
    imprimir("mensaje : " + pa.mensaje());  
    imprimir("hacer ruido : " + pa.hacerRuido());  
    imprimir("");  
    imprimir(">>> Objeto pm");  
    imprimir("mensaje : " + pm.mensaje());  
    imprimir("hacer ruido : " + pm.hacerRuido());  
}
```

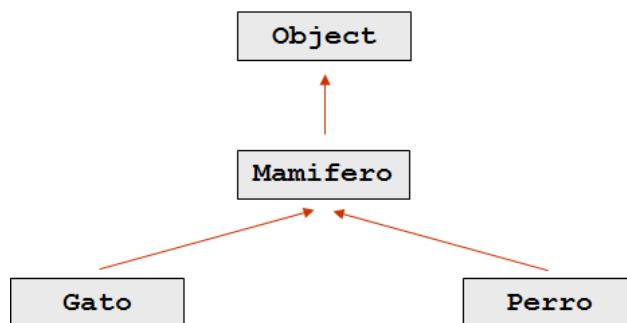


4.1.8. Polimorfismo y uso de `instanceof`

4.1.8.1. Polimorfismo

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 sobrescritos, 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 polimórfico cuando actúa de diferentes formas dependiendo del objeto que reciba.



```

public abstract class Mamifero {
    public String mensaje() {
        return "soy mamifero";
    }

    public abstract String hacerRuido();
}

public class Gato extends Mamifero {
    public String mensaje() {
        return "soy gato";
    }

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

public class Perro extends Mamifero {
    public String hacerRuido() {
        return "guau";
    }
}

protected void actionPerformedBtnProcesar(ActionEvent arg0) {
    Gato g = new Gato();
    listado(g);

    Perro p = new Perro();
    listado(p);
}
  
```

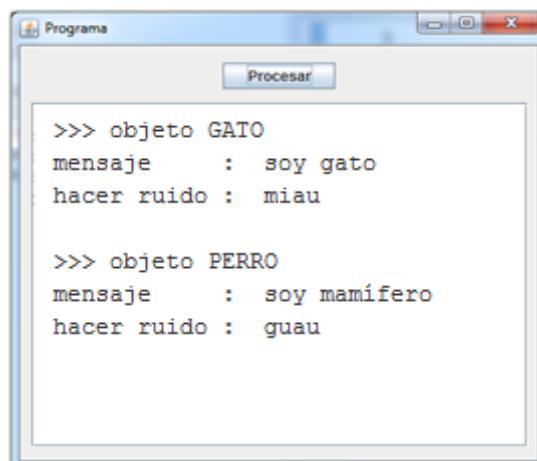
```
void listado(Mamifero x) {  
    imprimir("mensaje : " + x.mensaje());  
    imprimir("hacer ruido : " + x.hacerRuido());  
}
```

En el ejemplo, el método listado del programa principal es un método polimórfico, ya que cuando es llamado por primera vez, lista el comportamiento de un Gato y la segunda vez, lista el comportamiento de un Perro.

4.1.8.2. Operador instanceof

Se utiliza para determinar si el objeto es de la Clase esperada antes de realizar el casting.

```
void listado(Mamifero x) {  
    if (x instanceof Gato)  
        imprimir(">>> objeto GATO");  
    else  
        imprimir(">>> objeto PERRO");  
  
    imprimir("mensaje : " + x.mensaje());  
    imprimir("mensaje : " + x.hacerRuido());  
}
```



Ejercicios

Ejercicio Propuesto 1

Diseñe la clase **Figura** en el paquete padre que contenga:

- 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 Circulo** 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)

Circulo 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 ($\pi * radio * radio$)

Por último, implemente el método Procesar en el programa principal que contiene y cree 2 objetos: uno de tipo Cuadrado y el otro de tipo Circulo e imprima su ubicación y área de cada objeto.

Ejercicio Propuesto 2

Diseñe la clase **Trabajador** en el paquete padre con los siguientes elementos:

- Atributos protegidos: nombre, apellido y telefono de tipo String
- Constructor que inicializa los atributos de la clase.
- Un método codigoGenerado() que retorna el código formado por el primer carácter del nombre, el último carácter del apellido y el teléfono del trabajador.
- Un método datosDelTrabajador() que retorne en una cadena los datos completos 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 ($sbasic * porcentajeBonificacion / 100$)
- Un método sbruto() que retorna el sueldo bruto del empleado ($sbasic + bonificacion$ en soles)
- Un método datosCompletos() que retorne una cadena conteniendo: nombre, apellido, teléfono, 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 datosCompletos() que retorne una cadena conteniendo: nombre, apellido, teléfono y el sueldo bruto.

Por último, a la pulsación del botón Procesar de la clase Principal (donde está la GUI), cree 2 objetos con datos fijos: uno de tipo Empleado y el otro de tipo Consultor e imprima sus datos invocando al método mostrarDatos() y su código generado.

4.2. Interfaces

4.2.1. Definición

Una interfaz es una Clase completamente abstracta, es decir no tiene implementación. 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. Ejemplo:

```
package cibertec;
...
public class Programa extends JFrame implements ActionListener, ItemListener{
    ...
    public void actionPerformed(ActionEvent arg0) {
        ...
    }

    public void itemStateChanged(ItemEvent arg0) {
        ...
    }
    ...
}
```

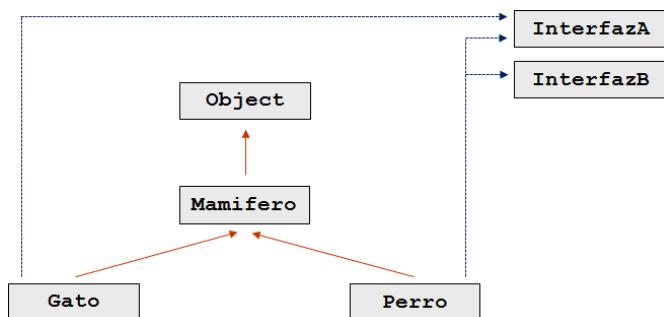
Una interfaz podrá verse simplemente como una forma, es como un molde; solamente permite declarar nombres de métodos. En este caso, no es necesario definirlos como abstractos, puesto que lo son implícitamente; y si adicionalmente tiene miembros datos, éstos serán constantes, es decir, **static** y **final**. Al utilizar **implements** para el interfaz es como si se hiciese una acción de copiar y pegar el código de la 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 interfaz y clase abstracta es que la interfaz posee un mecanismo de encapsulamiento sin forzar al usuario a utilizar la herencia.

4.2.2. Herencia Múltiple

En Java realmente no existe la herencia múltiple, lo que se puede hacer es crear una clase que implemente (**implements**) más de un interfaz, pero sólo puede extender a una clase (**extends**).

Ejemplo



```
package interfaz;

public interface InterfazA {
    public double vacunaA = 42.75;
    public String cuidado();
}
```

```
package interfaz;

public interface InterfazB {
    public double vacunaB = 96.28;
    public String peligro();
}
```

```
package padre;

public abstract class Mamifero {
    public String mensaje() {
        return "soy mamifero";
    }
    public abstract String hacerRuido();
}
```

```
package hijo;

import padre.Mamifero;
import interfaz.InterfazA;

public class Gato extends Mamifero implements InterfazA {
    public String mensaje() {
        return "soy gato";
    }
    public String hacerRuido() {
        return "miau";
    }
    public String cuidado() {
        return "el gato puede tener rabia";
    }
}
```

```
package hijo;

import padre.Mamifero;
import interfaz.*;

public class Perro extends Mamifero implements InterfazA, InterfazB {
    public String hacerRuido() {
        return "guau";
    }
    public String cuidado() {
        return "el perro puede tener rabia";
    }
    public String peligro() {
        return "el perro muerde";
    }
}
```

```

package cibertec;

import padre.Mamifero;
import hijo.*;

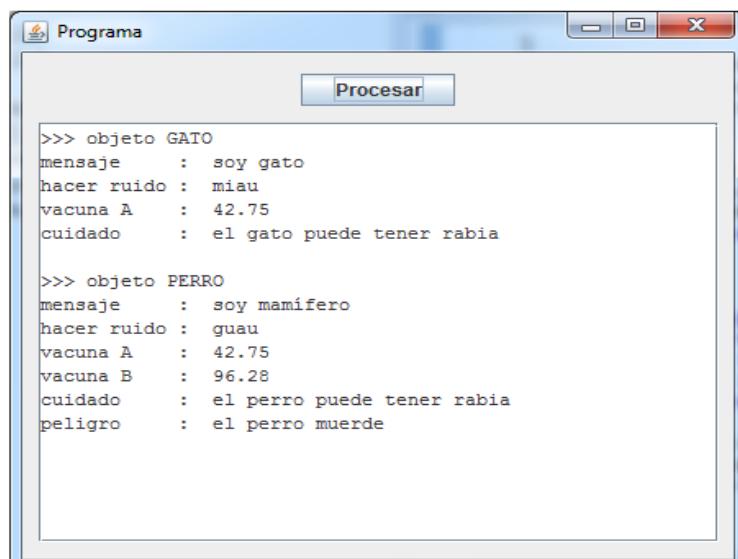
...
public class Programa extends JFrame implements ActionListener {
    ...
    protected void actionPerformedBtnProcesar(ActionEvent arg0) {
        Gato g = new Gato();
        listado(g);

        Perro p = new Perro();
        listado(p);
    }
    ...
    void listado(Mamifero x) {
        if (x instanceof Gato)
            imprimir(">>> objeto GATO");
        else
            imprimir(">>> objeto PERRO");

        imprimir("mensaje : " + x.mensaje());
        imprimir("hacer ruido : " + x.hacerRuido());

        if (x instanceof Gato) {
            imprimir("vacuna A : " + ((Gato)x).vacunaA);
            imprimir("cuidado : " + ((Gato)x).cuidado());
        } else {
            imprimir("vacuna A : " + ((Perro)x).vacunaA);
            imprimir("vacuna B : " + ((Perro)x).vacunaB);
            imprimir("cuidado : " + ((Perro)x).cuidado());
            imprimir("peligro : " + ((Perro)x).peligro());
        }
    }
    ...
}

```



Ejemplo

Diseño de una aplicación que muestra las operaciones **avanzar**, **detener**, **retroceder**, **subir** y **bajar** que pueden hacer los medios de transporte **auto**, **bicicleta**, **moto**, **avión** y **helicóptero**. Consideramos que algunos son terrestres y otros son aereos. Por lo tanto, hay operaciones que por ejemplo un avión puede hacer pero no un auto. En la solución empleamos interfaces, clases abstractas y subclases.

```
package interfaz;

public interface Movimiento {
    public String avanzar();
    public String detener();
    public String retroceder();
}
```

```
package interfaz;

public interface Vuelo {
    public String subir();
    public String bajar();
}
```

```
package parente;

import interfaz.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 hijo;

import padre.Transporte;

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 está avanzando";
    }
}
```

```
package hijo;

import padre.Transporte;

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 está avanzando";
    }
    public String detener() {
        return "la bicicleta se detuvo";
    }
}
```

```
package hijo;

import padre.Transporte;

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 está avanzando";
    }
    public String detener() {
        return "la moto se detuvo";
    }
    public String retroceder() {
        return "la moto está retrocediendo";
    }
}
```

```
package hijo;

import padre.Transporte;
import interfaz.Vuelo;

public class Avion extends Transporte implements Vuelo {
    public Avion(int capacidad) {
        super(capacidad);
    }

    public String mostrarCapacidad() {
        return "Capacidad de pasajeros del Avión : " + capacidad;
    }

    public String avanzar() {
        return "el avión está avanzando";
    }

    public String detener() {
        return "el avión se detuvo";
    }

    public String retroceder() {
        return "el avión está retrocediendo";
    }

    public String subir() {
        return "el avión está subiendo";
    }

    public String bajar() {
        return "el avión está bajando";
    }
}
```

```
package hijo;

import padre.Transporte;
import interfaz.Vuelo;

public class Helicoptero extends Transporte implements Vuelo {
    public Helicoptero(int capacidad) {
        super(capacidad);
    }

    public String mostrarCapacidad() {
        return "Capacidad de pasajeros del Helicóptero : " + capacidad;
    }

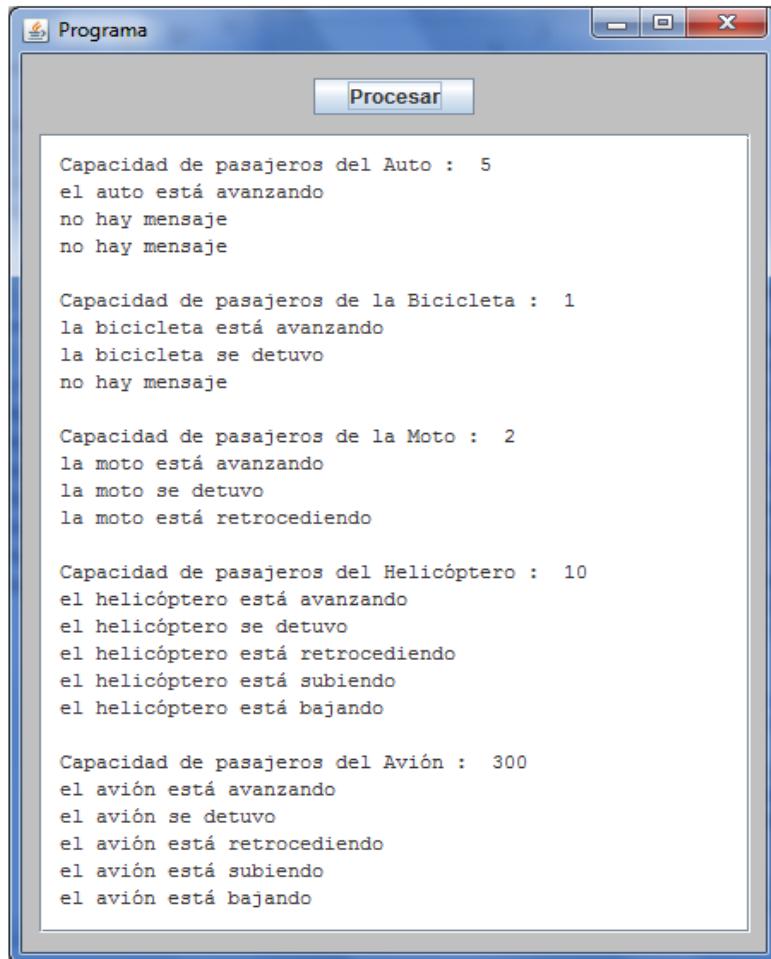
    public String avanzar() {
        return "el helicóptero está avanzando";
    }

    public String detener() {
        return "el helicóptero se detuvo";
    }

    public String retroceder() {
        return "el helicóptero está retrocediendo";
    }
}
```

```
public String subir() {  
    return "el helicóptero está subiendo";  
}  
  
public String bajar() {  
    return "el helicóptero está bajando";  
}  
}
```

```
package cibertec;  
  
import padre.Transporte;  
import hijo.*;  
...  
public class Programa extends JFrame implements ActionListener {  
    protected void actionPerformedBtnProcesar(ActionEvent arg0) {  
        Auto aut = new Auto(5);  
        listado(aut);  
        imprimir("");  
  
        Bicicleta bic = new Bicicleta(1);  
        listado(bic);  
        imprimir("");  
  
        Moto mot = new Moto(2);  
        listado(mot);  
        imprimir("");  
  
        Helicoptero hel = new Helicoptero(10);  
        listado(hel);  
        imprimir(hel.subir());  
        imprimir(hel.bajar());  
        imprimir("");  
  
        Avion avi = new Avion(300);  
        listado(avi);  
        imprimir(avi.subir());  
        imprimir(avi.bajar());  
    }  
  
    // Mètodes tipo void (con paràmetres)  
    void listado(Transporte x) {  
        imprimir(x.mostrarCapacidad());  
        imprimir(x.avanzar());  
        imprimir(x.detener());  
        imprimir(x.retroceder());  
    }  
    ...  
}
```





ARCHIVOS

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 `PrintWriter`, `FileWriter`, `BufferedReader` y `FileReader` para almacenar la data en archivos de texto.

TEMARIO

5.1. Tema 14 : Manejo de archivos de texto

- 5.1.1 : Descripción
- 5.1.2 : Clases `PrintWriter` y `FileWriter`
- 5.1.3 : Escritura en un archivo de texto
- 5.1.4 : Clases `BufferedReader` y `FileReader`
- 5.1.5 : Método `split` de la clase `String`
- 5.1.6 : Lectura de un archivo de texto

5.2 Tema 15 : Mantenimiento

- 5.2.1 : Proyecto con archivos de texto: Caso Inmobiliaria

ACTIVIDADES PROPUESTAS

- Emplear los métodos de la clase `ArrayList` para manipular un arreglo de objetos.
- Crear un mantenimiento.
- Emplear las clases `PrintWriter`, `FileWriter`, `BufferedReader`, `FileReader` y `File` para almacenar un arreglo de objetos en archivos de texto.
- Manipular arreglo de objetos utilizando los métodos de la clase `ArrayList`.

5.1. Manejo de Archivos de Texto

5.1.1. Descripcion

Para tener acceso a las clases que permiten manejar archivos de texto es necesario colocar:

```
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.

```
try {
    ...
} catch(Exception e) {
    ...
}
```

5.1.2. Clases PrintWriter y FileWriter

Permite escribir los datos de la memoria hacia un archivo (output).

A través de su constructor debemos enviar un objeto creado del tipo **FileWriter** con el nombre del archivo a crear. Ejemplo:

```
PrintWriter pw = new PrintWriter(new FileWriter("Docentes.txt"));
```

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.

Por defecto el archivo es creado dentro de la carpeta classes.

Una vez abierto el archivo, **pw** apunta a la primera línea.



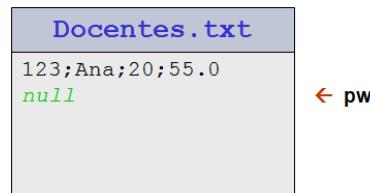
El método **println(String)** asociado a **pw** graba la cadena enviada y genera un salto de línea. Ejemplo:

```
String linea = "123;Ana;20;55.0";
pw.println(linea);
```

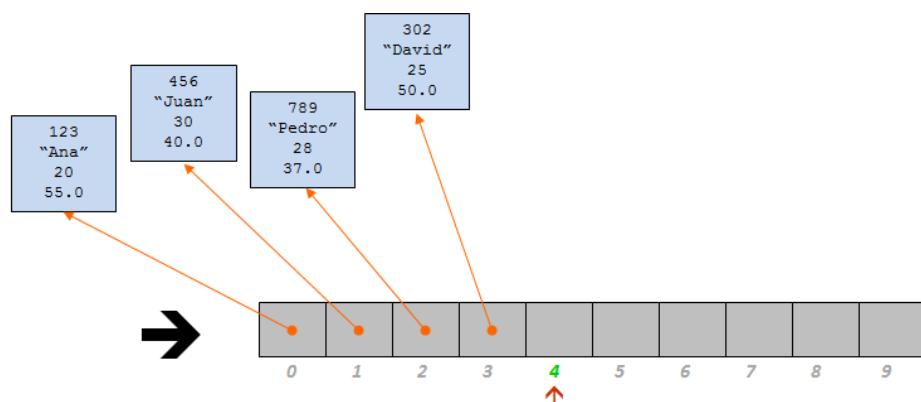


El método **close()** asociado a **pw** cierra el acceso de escritura, colocando **null** en la línea señalada por **pw**. Ejemplo:

```
pw.close();
```



Asumiendo el colecciónista **ArrayList** de tipo **Docente** representado por **doc** luego de cuatro ingresos.



Ejercicio 47: Añada un método público a la clase **ArregloDocentes** que grabe el **ArrayList** de docentes en el archivo de texto **Docentes.txt**.

```
public void grabarDocentes() {
    try {
        PrintWriter pw;
        Docente x;
        String linea;

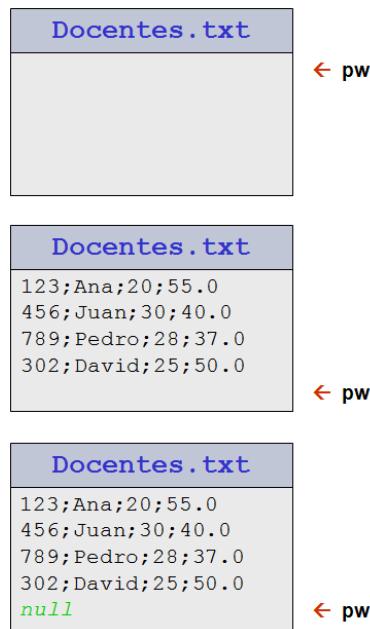
        pw = new PrintWriter(new FileWriter("Docentes.txt"));

        for (int i=0; i<doc.size(); i++) {
            x = doc.get(i);

            linea = x.getCodigo() + ";" +
                    x.getNombre() + ";" +
                    x.getHoras() + ";" +
                    x.getTarifa();

            pw.println(linea);
        }

        pw.close();
    } catch(Exception e) {
    }
}
```



5.1.3. Clases BufferedReader y FileReader

Permite leer los datos de un archivo a la memoria (input).

A través de su constructor debemos enviar un objeto creado del tipo **FileReader** con el nombre del archivo a leer. Ejemplo:

```
BufferedReader br = new BufferedReader(new FileReader("Docentes.txt"));
```

La Clase **FileReader** se encarga de abrir el archivo en modo de lectura.

Por defecto el archivo es creado dentro de la carpeta classes.

Una vez abierto el archivo, **br** apunta a la primera línea.



El método **readLine()** asociado a **br** retorna una cadena y genera un salto de línea. Ejemplo:

```
String linea = br.readLine();
```

El resultado será: linea = "123;Ana;20;55.0";

Docentes.txt
123;Ana;20;55.0
456;Juan;30;40.0
789;Pedro;28;37.0
302;David;25;50.0
null

← br

El método **close()** asociado a **br** cierra el acceso de lectura. Ejemplo:

```
br.close();
```

5.1.4. Método **split** de la clase **String**

Busca un tope en una cadena y distribuye una copia de las subcadenas en un arreglo lineal de cadenas. Ejemplo:

```
String linea = "123;Ana;20;55.0", s[];
s = linea.split(";;");
```

El resultado será:

```
s[0] ← "123"
s[1] ← "Ana"
s[2] ← "20"
s[3] ← "55.0"
```

Ejercicio 48: Añada un método público a la clase **ArregloDocentes** que cargue el archivo de texto Docentes.txt en el ArrayList de docentes.

```
public void cargarDocentes() {
    try {
        BufferedReader br;
        String linea, s[], nombre;
        int codigo, horas;
        double tarifa;

        br = new BufferedReader(new FileReader("Docentes.txt"));

        while ((linea = br.readLine()) != null) {
            s = linea.split(";;");
            codigo = Integer.parseInt(s[0]);
            nombre = s[1];
            horas = Integer.parseInt(s[2]);
            tarifa = Double.parseDouble(s[3]);

            doc.add(new Docente(codigo, nombre, horas, tarifa));
        }

        br.close();
    } catch (Exception e) {
    }
}
```

5.1.5. Clase File

Permite averiguar acerca de un archivo antes de realizar alguna actividad sobre él. El método `exists()` devuelve `true` o `false` en caso exista o no exista el archivo. Ejemplo:

```
File f;
f = new File("Docentes.txt");
if (f.exists()) {
    ...
} else {
    ...
}
```

Ejercicio 49: Añada un atributo privado de tipo cadena en `ArregloDocentes` que guarde el nombre del archivo. Dicho atributo se inicializará a través del constructor. Asimismo dentro del constructor invoque al método cargarDocentes. Implemente los métodos de acceso público `set/get` respectivos.

```
package semana_11;

import java.util.ArrayList;

public class ArregloDocentes {
    // Atributos privados
    private ArrayList <Docente> doc;
    private String archivo;

    // Constructor
    public ArregloDocentes(String archivo) {
        doc = new ArrayList <Docente> ();
        this.archivo = archivo;
        cargarDocentes();
    }

    // Métodos de acceso: set/get
    public void setArchivo(String archivo) {
        this.archivo = archivo;
    }

    public String getArchivo() {
        return archivo;
    }
}
```

Ejercicio 50: En el programa principal envíe el nombre del archivo de texto al momento de crear el objeto `ad`.

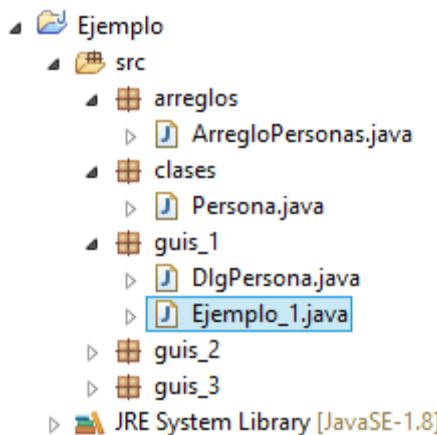
```
package cibertec;

import semana_11.*;

public class Programa extends JFrame implements ActionListener {
    // Declaración de variables globales
    ArregloDocentes ad = new ArregloDocentes("Docentes.txt");
}
```

5.2. Mantenimiento

5.2.1. Diseño básico de un Proyecto con archivos de texto



Ejercicio 51: Diseñe la clase Persona en el paquete `clases` con los atributos privados: `codigo` (int), `nombre` (String), `dni` (String), `peso` (double), `estatura` (double) y `estado` (int).

Implemente además

- Un constructor que inicialice a todos los atributos.
- Métodos de acceso público `set` para todos los atributos privados. Use la referencia `this`.
- Métodos de acceso público `get` para todos los atributos privados.
- Método público que retorne el índice de masa corporal:
 $imc = peso / (estatura * estatura)$

```
package clases;

public class Persona {
    // Atributos privados
    private int codigo, estado;
    private String nombre, dni;
    private double peso, estatura;

    // Constructor
    public Persona(int codigo, String nombre, String dni,
                  double peso, double estatura, int estado) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.dni = dni;
        this.peso = peso;
        this.estatura = estatura;
        this.estado = estado;
    }

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

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

```
public void setDni(String dni) {
    this.dni = dni;
}

public void setPeso(double peso) {
    this.peso = peso;
}

public void setEstatura(double estatura) {
    this.estatura = estatura;
}

public void setEstado(int estado) {
    this.estado = estado;
}

public int getCodigo() {
    return codigo;
}

public String getNombre() {
    return nombre;
}

public String getDni() {
    return dni;
}

public double getPeso() {
    return peso;
}

public double getEstatura() {
    return estatura;
}

public int getEstado() {
    return estado;
}

// Operaciones públicas complementarias
public double imc() {
    return peso / (estatura * estatura);
}

}
```

Ejercicio 52: Implemente la clase **ArregloPersonas** en el paquete arreglos con el atributo privado **ArrayList per** de tipo Persona.

Implemente como públicos:

- Un constructor que cree el *ArrayList* y autogeneré cinco objetos.
- Un método **adicionar** que reciba la dirección de memoria de una nueva persona y la adicione al *ArrayList*.
- Un método **tamaño** que retorne la cantidad de elementos registrados hasta ese momento.
- Un método **obtener** que reciba la posición y retorne la dirección de memoria de la persona respectiva.

- Un método **eliminar** que reciba la dirección de memoria de un objeto Persona y lo retire del *ArrayList*.
- Un método **buscar** que busque un código y retorne la dirección de memoria del objeto que lo contiene. En caso no exista retorne null.
- Un método **buscar** que busque un dni y retorne la dirección de memoria del objeto que lo contiene. En caso no exista retorne null.
- Un método **codigoCorrelativo** que retorne un número entero autogenerado y correlativo empezando a partir de 10001.
- Un método **grabarPersonas** que guarde los datos de las Personas en un archivo de texto.
- Un método **cargarPersonas** que lea los datos de las Personas de un archivo de texto.

```
package arreglos;

import clases.Persona;
import java.util.ArrayList;

public class ArregloPersonas {
    // Atributo privado
    private ArrayList <Persona> per;

    // Constructor
    public ArregloPersonas () {
        per = new ArrayList <Persona> ();
        adicionar(new Persona(1001, "Juan Prado Salazar", "07557853", 82.3, 1.75, 3));
        adicionar(new Persona(1002, "Pedro Romero Soto", "11002348", 79.5, 1.58, 1));
        adicionar(new Persona(1003, "Luis Pinto Garza", "62279345", 82.7, 1.83, 0));
        adicionar(new Persona(1004, "Daniel Rojas Saenz", "20977241", 80.2, 1.72, 2));
        adicionar(new Persona(1005, "Jorge Espinal Vega", "06377845", 75.9, 1.88, 1));
    }

    // Operaciones públicas básicas
    public void adicionar(Persona p) {
        per.add(p);
    }

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

    public Persona obtener(int pos) {
        return per.get(pos);
    }

    public void eliminar(Persona x) {
        per.remove(x);
    }

    public Persona buscar(int codigo) {
        Persona x;
        for (int i=0; i<tamaño(); i++) {
            x = obtener(i);
            if (x.getCodigo() == codigo)
                return x;
        }
        return null;
    }
    public Persona buscar(String dni) {
        Persona x;
```

```

        for (int i=0; i<tamaño(); i++) {
            x = obtener(i);
            if (x.getDni().equals(dni))
                return x;
        }

        return null;
    }

    // Operaciones públicas complementarias
    public int codigoCorrelativo() {
        if (tamaño() == 0)
            return 10001;
        else
            return obtener(tamaño()-1).getCodigo() + 1;
    }
    public void grabarPersonas() {
        try {
            PrintWriter pw;
            String linea;
            Persona x;
            pw = new PrintWriter(new FileWriter(archivo));
            for (int i=0; i<tamaño(); i++) {
                x = obtener(i);
                linea = x.getCodigo() + ";" +
                    x.getNombre() + ";" +
                    x.getPeso() + ";" +
                    x.getEstatura();
                pw.println(linea);
            }
            pw.close();
        } catch (Exception e) {
        }
    }
    public void cargarPersonas() {
        try {
            BufferedReader br;
            String linea, nombre;
            String s[];
            int codigo;
            double peso, estatura;
            br = new BufferedReader(new FileReader(archivo));
            while ((linea = br.readLine()) != null) {
                s = linea.split(";");
                codigo = Integer.parseInt(s[0].trim());
                nombre = s[1].trim();
                peso = Double.parseDouble(s[2].trim());
                estatura = Double.parseDouble(s[3].trim());
                adicionar(new Persona(codigo, nombre, peso, estatura));
            }
            br.close();
        } catch (Exception e) {
        }
    }
}

```

Ejercicio 53: Diseñe en la clase principal **DlgPersona** una GUI adecuada para realizar las operaciones básicas de todo proyecto: adicionar, consultar, modificar y eliminar. Realice las

validaciones de ingreso respectivas. Considere la declaración global **ArregloPersonas ap = new ArregloPersonas ();**

```
package guis_1;

import clases.Persona;
import arreglos.ArregloPersonas;

import java.awt.EventQueue;

import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumnModel;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class DlgPersona extends JDialog implements ActionListener {

    private JLabel lblCodigo;
    private JLabel lblNombre;
    private JLabel lblDni;
    private JLabel lblPeso;
    private JLabel lblEstatura;
    private JLabel lblEstadoCivil;
    private JTextField txtCodigo;
    private JTextField txtNombre;
    private JTextField txtDni;
    private JTextField txtPeso;
    private JTextField txtEstatura;
    private JComboBox <String> cboEstadoCivil;
    private JScrollPane scrollPane;
    private JButton btnBuscar;
    private JButton btnAdicionar;
    private JButton btnConsultar;
    private JButton btnModificar;
    private JButton btnEliminar;
    private JButton btnAceptar;
    private JButton btnVolver;
    private JTable tblPersona;
    private DefaultTableModel modelo;

    //Tipo de operación a procesar:Adicionar,Consultar,Modificar o Eliminar
    private int tipoOperacion;

    // Constantes para los tipos de operaciones
    public final static int ADICIONAR = 0;
    public final static int CONSULTAR = 1;
    public final static int MODIFICAR = 2;
    public final static int ELIMINAR = 3;
```

```
/** Launch the application. */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                DlgPersona dialog = new DlgPersona();
                dialog.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                dialog.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/** Create the dialog. */
public DlgPersona() {
    setResizable(false);
    setTitle("Mantenimiento | Persona");
    setBounds(100, 100, 800, 600);
    getContentPane().setLayout(null);

    lblCodigo = new JLabel("Código");
    lblCodigo.setBounds(10, 10, 110, 23);
    getContentPane().add(lblCodigo);

    lblNombre = new JLabel("Nombre");
    lblNombre.setBounds(10, 35, 70, 23);
    getContentPane().add(lblNombre);

    lblDni = new JLabel("DNI");
    lblDni.setBounds(10, 60, 70, 23);
    getContentPane().add(lblDni);

    lblPeso = new JLabel("Peso");
    lblPeso.setBounds(10, 85, 70, 23);
    getContentPane().add(lblPeso);

    lblEstatura = new JLabel("Estatura");
    lblEstatura.setBounds(10, 110, 70, 23);
    getContentPane().add(lblEstatura);

    lblEstadoCivil = new JLabel("Estado civil");
    lblEstadoCivil.setBounds(10, 135, 86, 23);
    getContentPane().add(lblEstadoCivil);

    txtCodigo = new JTextField();
    txtCodigo.setBounds(90, 10, 86, 23);
    getContentPane().add(txtCodigo);
    txtCodigo.setEditable(false);
    txtCodigo.setColumns(10);

    txtNombre = new JTextField();
    txtNombre.setBounds(90, 35, 251, 23);
    getContentPane().add(txtNombre);
    txtNombre.setEditable(false);
    txtNombre.setColumns(10);
```

```
txtDni = new JTextField();
txtDni.setBounds(90, 60, 86, 23);
getContentPane().add(txtDni);
txtDni.setEditable(false);
txtDni.setColumns(10);

txtPeso = new JTextField();
txtPeso.setBounds(90, 85, 50, 23);
getContentPane().add(txtPeso);
txtPeso.setEditable(false);
txtPeso.setColumns(10);

txtEstatura = new JTextField();
txtEstatura.setBounds(90, 110, 50, 23);
getContentPane().add(txtEstatura);
txtEstatura.setEditable(false);
txtEstatura.setColumns(10);

cboEstadoCivil = new JComboBox <String> ();
cboEstadoCivil.setModel(new DefaultComboBoxModel <String>
    (new String[] {"Soltero", "Casado",
        "Viudo", "Divorciado"}));
cboEstadoCivil.setBounds(90, 135, 86, 23);
getContentPane().add(cboEstadoCivil);
cboEstadoCivil.setEnabled(false);

scrollPane = new JScrollPane();
scrollPane.setBounds(10, 170, 775, 360);
getContentPane().add(scrollPane);

tblPersona = new JTable();
tblPersona.setFillsViewportHeight(true);
scrollPane.setViewportView(tblPersona);

modelo = new DefaultTableModel();
modelo.addColumn("CÓDIGO");
modelo.addColumn("NOMBRE");
modelo.addColumn("DNI");
modelo.addColumn("PESO (kg)");
modelo.addColumn("ESTATURA (mts)");
modelo.addColumn("ESTADO CIVIL");
modelo.addColumn("IMC = peso/estatura2");
tblPersona.setModel(modelo);

btnBuscar = new JButton("Buscar");
btnBuscar.addActionListener(this);
btnBuscar.setEnabled(false);
btnBuscar.setBounds(240, 10, 101, 23);
getContentPane().add(btnBuscar);

btnAdicionar = new JButton("Adicionar");
btnAdicionar.addActionListener(this);
btnAdicionar.setBounds(10, 540, 120, 23);
getContentPane().add(btnAdicionar);

btnConsultar = new JButton("Consultar");
btnConsultar.addActionListener(this);
btnConsultar.setBounds(135, 540, 120, 23);
getContentPane().add(btnConsultar);
```

```
btnModificar = new JButton("Modificar");
btnModificar.addActionListener(this);
btnModificar.setBounds(260, 540, 120, 23);
getContentPane().add(btnModificar);

btnEliminar = new JButton("Eliminar");
btnEliminar.addActionListener(this);
btnEliminar.setBounds(385, 540, 120, 23);
getContentPane().add(btnEliminar);

btnAceptar = new JButton("Aceptar");
btnAceptar.addActionListener(this);
btnAceptar.setEnabled(false);
btnAceptar.setBounds(510, 540, 120, 23);
getContentPane().add(btnAceptar);

btnVolver = new JButton("Volver");
btnVolver.addActionListener(this);
btnVolver.setEnabled(false);
btnVolver.setBounds(664, 540, 120, 23);
getContentPane().add(btnVolver);

ajustarAnchoColumnas();
listar();
}

// Declaración global
ArregloPersonas ap = new ArregloPersonas();

public void actionPerformed(ActionEvent arg0) {
    if (arg0.getSource() == btnBuscar) {
        actionPerformedBtnBuscar(arg0);
    }

    if (arg0.getSource() == btnVolver) {
        actionPerformedBtnVolver(arg0);
    }

    if (arg0.getSource() == btnAceptar) {
        actionPerformedBtnAceptar(arg0);
    }

    if (arg0.getSource() == btnEliminar) {
        actionPerformedBtnEliminar(arg0);
    }

    if (arg0.getSource() == btnModificar) {
        actionPerformedBtnModificar(arg0);
    }

    if (arg0.getSource() == btnConsultar) {
        actionPerformedBtnConsultar(arg0);
    }

    if (arg0.getSource() == btnAdicionar) {
        actionPerformedBtnAdicionar(arg0);
    }
}
```

```
protected void actionPerformedBtnAdicionar(ActionEvent arg0) {
    tipoOperacion = ADICIONAR;

    txtCodigo.setText(" " + ap.codigoCorrelativo());
    txtNombre.setText("");
    txtDni.setText("");
    txtPeso.setText("");
    txtEstatura.setText("");
    habilitarEntradas(true);
    habilitarBotones(false);
    txtNombre.requestFocus();
}

protected void actionPerformedBtnConsultar(ActionEvent arg0) {
    tipoOperacion = CONSULTAR;

    txtCodigo.setEditable(true);
    habilitarBotones(false);
    txtCodigo.requestFocus();
}

protected void actionPerformedBtnModificar(ActionEvent arg0) {
    tipoOperacion = MODIFICAR;

    txtCodigo.setEditable(true);
    habilitarEntradas(true);
    habilitarBotones(false);
    txtCodigo.requestFocus();
}

protected void actionPerformedBtnEliminar(ActionEvent arg0) {
    tipoOperacion = ELIMINAR;

    txtCodigo.setEditable(true);
    habilitarBotones(false);
}

protected void actionPerformedBtnAceptar(ActionEvent arg0) {
    switch (tipoOperacion) {
        case ADICIONAR:
            adicionarPersona();
            break;
        case CONSULTAR:
            consultarPersona();
            break;
        case MODIFICAR:
            modificarPersona();
            break;
        case ELIMINAR:
            eliminarPersona();
    }
}

protected void actionPerformedBtnBuscar(ActionEvent arg0) {
    consultarPersona();
}
```

```

protected void actionPerformedBtnVolver(ActionEvent arg0) {
    txtCodigo.setText("");
    txtNombre.setText("");
    txtDni.setText("");
    txtPeso.setText("");
    txtEstatura.setText("");
    txtCodigo.setEditable(false);
    habilitarEntradas(false);
    habilitarBotones(true);
}

// Métodos tipo void (sin parámetros)
void ajustarAnchoColumnas() {
    TableColumnModel tcm = tblPersona.getColumnModel();
    tcm.getColumn(0).setPreferredWidth(anchoColumna(10)); // código
    tcm.getColumn(1).setPreferredWidth(anchoColumna(20)); // nombre
    tcm.getColumn(2).setPreferredWidth(anchoColumna(10)); // dni
    tcm.getColumn(3).setPreferredWidth(anchoColumna(10)); // peso
    tcm.getColumn(4).setPreferredWidth(anchoColumna(15)); // estatura
    tcm.getColumn(5).setPreferredWidth(anchoColumna(15)); // estadoCivil
    tcm.getColumn(6).setPreferredWidth(anchoColumna(20)); // imc
}

void listar() {
    Persona x;
    modelo.setRowCount(0);
    for (int i=0; i<ap.tamaño(); i++) {
        x = ap.obtener(i);

        Object[] fila = {x.getCodigo(),
                         x.getNombre(),
                         x.getDni(),
                         x.getPeso(),
                         x.getEstatura(),
                         enTextoEstadoCivil(x.getEstado()),
                         x.imc() };

        modelo.addRow(fila);
    }
}

void consultarPersona() {
    try {
        Persona x = ap.buscar(leerCodigo());

        if (x != null) {
            txtNombre.setText(x.getNombre());
            txtDni.setText(x.getDni());
            txtPeso.setText(" " + x.getPeso());
            txtEstatura.setText(" " + x.getEstatura());
            cboEstadoCivil.setSelectedIndex(x.getEstado());
            txtCodigo.requestFocus();
        } else
            error("El código " + leerCodigo() + " no existe", txtCodigo);
    } catch (Exception e) {
        error("Ingrese CÓDIGO correcto", txtCodigo);
    }
}

```

```

void adicionarPersona() {
    int codigo = leerCodigo();
    String nombre = leerNombre();

    if (nombre.length() > 0) {
        String dni = leerDni();

        if (ap.buscar(dni) == null)
            try {
                double peso = leerPeso();
                try {
                    double estatura = leerEstatura();
                    int estado = leerPosEstado();
                    Persona nueva = new Persona(codigo, nombre,
                        dni, peso,
                        estatura, estado);
                    ap.adicionar(nueva);
                    listar();

                    txtCodigo.setText("") + ap.codigoCorrelativo());
                    txtNombre.setText("");
                    txtDni.setText("");
                    txtPeso.setText("");
                    txtEstatura.setText("");
                    txtNombre.requestFocus();
                } catch (Exception e) {
                    error("Ingrese ESTATURA correcta", txtEstatura);
                }
            } catch (Exception e) {
                error("Ingrese PESO correcto", txtPeso);
            }
        else
            error("Ingrese DNI correcto", txtDni);
    } else
        error("Ingrese NOMBRE correcto", txtNombre);
}

void eliminarPersona() {
    try {
        int codigo = leerCodigo();
        Persona x = ap.buscar(codigo);

        if (x != null) {
            ap.eliminar(x);
            listar();

            txtCodigo.setText("");
            txtNombre.setText("");
            txtDni.setText("");
            txtPeso.setText("");
            txtEstatura.setText("");
            txtCodigo.requestFocus();
        } else
            error("El código " + codigo + " no existe", txtCodigo);
    } catch (Exception e) {
        error("Ingrese CÓDIGO correcto", txtCodigo);
    }
}

```

```

void modificarPersona() {
    try {
        Persona x = ap.buscar(leerCodigo());
        String nombre = leerNombre();
        if (nombre.length() > 0) {
            String dni = leerDni();
            Persona y = ap.buscar(dni);
            if (y == null || x.equals(y))
                try {
                    double peso = leerPeso();
                    try {
                        double estatura = leerEstatura();
                        int estado = leerPosEstado();
                        x.setNombre(nombre);
                        x.setDni(dni);
                        x.setPeso(peso);
                        x.setEstatura(estatura);
                        x.setEstado(estado);
                        listar();
                        txtCodigo.requestFocus();
                    } catch (Exception e) {
                        error("Ingrese ESTATURA correcta",
                            txtEstatura);
                    }
                } catch (Exception e) {
                    error("Ingrese PESO correcto", txtPeso);
                }
            else
                error("Ingrese DNI correcto", txtDni);
        } else
            error("Ingrese NOMBRE correcto", txtNombre);
    } catch (Exception e) {
        error("Ingrese CÓDIGO correcto", txtCodigo);
    }
}

// Métodos tipo void (con parámetros)
void error(String s, JTextField txt) {
    mensaje(s);
    txt.setText("");
    txt.requestFocus();
}

void habilitarBotones(boolean sino) {
    if (tipoOperacion != ADICIONAR)
        btnBuscar.setEnabled(!sino);
    btnAdicionar.setEnabled(sino);
    btnConsultar.setEnabled(sino);
    btnModificar.setEnabled(sino);
    btnEliminar.setEnabled(sino);

    if (tipoOperacion == CONSULTAR)
        btnAceptar.setEnabled(false);
    else
        btnAceptar.setEnabled(!sino);

    btnVolver.setEnabled(!sino);
}

```

```
void habilitarEntradas(boolean sino) {
    txtNombre.setEditable(sino);
    txtDni.setEditable(sino);
    txtPeso.setEditable(sino);
    txtEstatura.setEditable(sino);
    cboEstadoCivil.setEnabled(sino);
}

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

// Métodos que retornan valor (sin parámetros)
int leerCodigo() {
    return Integer.parseInt(txtCodigo.getText().trim());
}

String leerNombre() {
    return txtNombre.getText().trim();
}

String leerDni() {
    return txtDni.getText().trim();
}

double leerPeso() {
    return Double.parseDouble(txtPeso.getText().trim());
}

double leerEstatura() {
    return Double.parseDouble(txtEstatura.getText().trim());
}

int leerPosEstado() {
    return cboEstadoCivil.getSelectedIndex();
}

// Métodos que retornan valor (con parámetros)
int anchoColumna(int porcentaje) {
    return porcentaje * scrollPane.getWidth() / 100;
}

String enTextoEstadoCivil(int i) {
    return cboEstadoCivil.getItemAt(i);
}
```

Mantenimiento | Persona

Código	<input type="text"/>	<input type="button" value="Buscar"/>
Nombre	<input type="text"/>	
DNI	<input type="text"/>	
Peso	<input type="text"/>	
Estatura	<input type="text"/>	
Estado civil	<input type="text" value="Soltero"/>	<input type="button" value="▼"/>

CÓDIGO	NOMBRE	DNI	PESO (kg)	ESTATURA (mts)	ESTADO CIVIL	IMC = peso/estatura ²
1001	Juan Prado Salazar	07557853	82.3	1.75	Divorciado	26.8734693877551
1002	Pedro Romero Soto	11002348	79.5	1.58	Casado	31.84585803557122
1003	Luis Pinto Garza	62279345	82.7	1.83	Soltero	24.69467586371644
1004	Daniel Rojas Saenz	20977241	80.2	1.72	Viudo	27.109248242293134
1005	Jorge Espinal Vega	06377845	75.9	1.88	Casado	21.47464916251698

Ejercicios

Ejercicio Propuesto 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:

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

Ejercicio Propuesto 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=suspense, 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:

- 1) Diseñe un método que imprima el precio unitario promedio de aquellos videos del género suspense.
- 2) Diseñe un método que elimine los videos del género ingresado desde la GUI.

Ejercicio Propuesto 3

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, modificar, eliminar y listar del mantenimiento de vendedores. Para ello implemente las clases ArregloVendedores (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:

- 1) Diseñe un método que imprima el monto promedio de aquellos vendedores cuyo nombre empiece con 'J'
- 2) Diseñe un método que imprima el nombre del vendedor que vendió que generó más ingresos.

Ejercicio Propuesto 4

Diseñe un mantenimiento de celulares que permita ingresar, consultar, modificar, eliminar y listar celulares. Para ello, cree las clases Celular, ArregloCelulares 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.

Ejercicio Propuesto 5

Diseñe un mantenimiento de videos que permita ingresar, consultar, modificar, eliminar y listar videos. Para ello, cree las clases Video, ArregloVideos 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=suspense, 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.



ANEXO

OBJETIVO ESPECIFICO

- Resolver un caso práctico integral
- Compartir otros ejercicios resueltos

ACTIVIDADES CONSIDERADAS

- Creación de paquetes, clases y objetos
- Uso de la referencia this
- Uso de especificadores de acceso public, private y protected
- Uso del modificador static (librerías)
- Uso de la clase ArrayList y archivos de texto
- Uso de clases y métodos abstractos
- Empleo de encapsulamiento, herencia y polimorfismo
- Empleo de interfaces

6.1. Caso Inmobiliaria

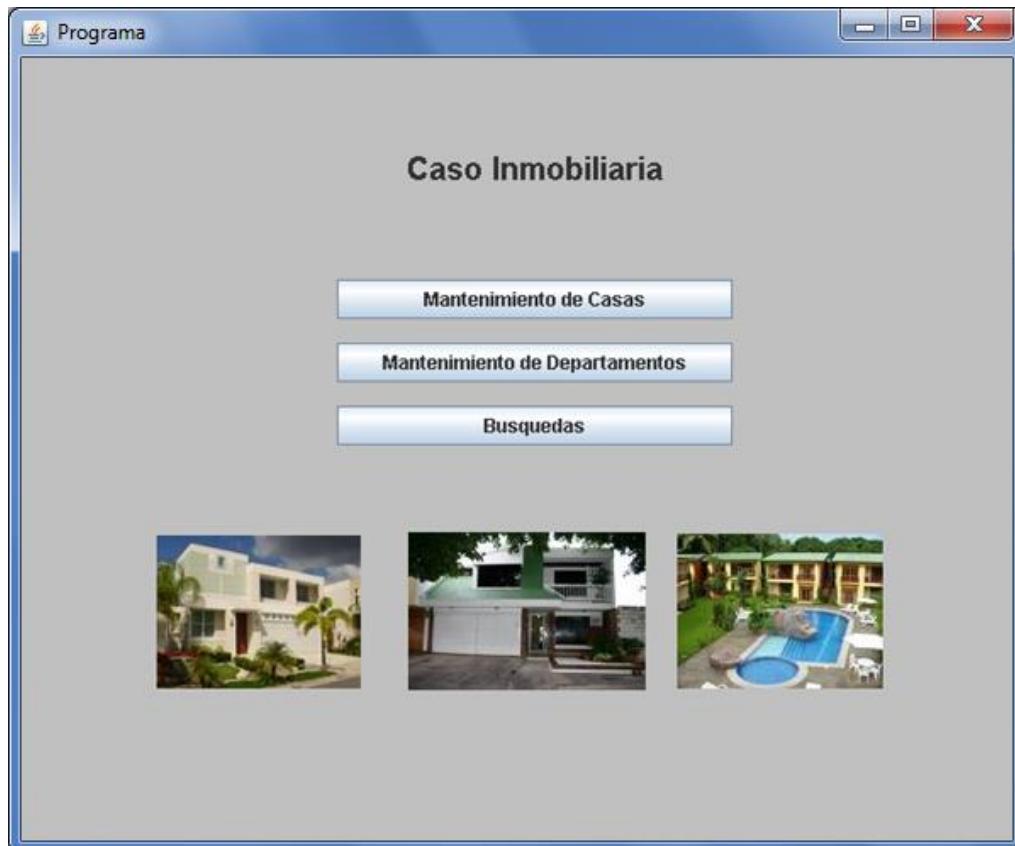
Una inmobiliaria requiere de una Aplicación que le permita dar información adecuada sobre las propiedades que renta a sus clientes, para lo cual se debe desarrollar un: **Mantenimiento de las propiedades que renta**.

Considera las opciones: ingreso, consulta, modificación y eliminación.

Diariamente acuden clientes a la inmobiliaria buscando información sobre casas y departamentos disponibles y que cubra sus expectativas. Por lo tanto se desea que la Aplicación realice la búsqueda por intervalo de área, intervalo de precio, intervalo de área y precio, por propiedad más barata y por 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.



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 búsqueda puede realizar consultas de qué casas y/o departamentos están disponibles según los criterios seleccionados: área, precio, más barato y más caro.

Utilice el JComboBox de opción para seleccionar la opción de ingreso, consulta, modificación o eliminación.

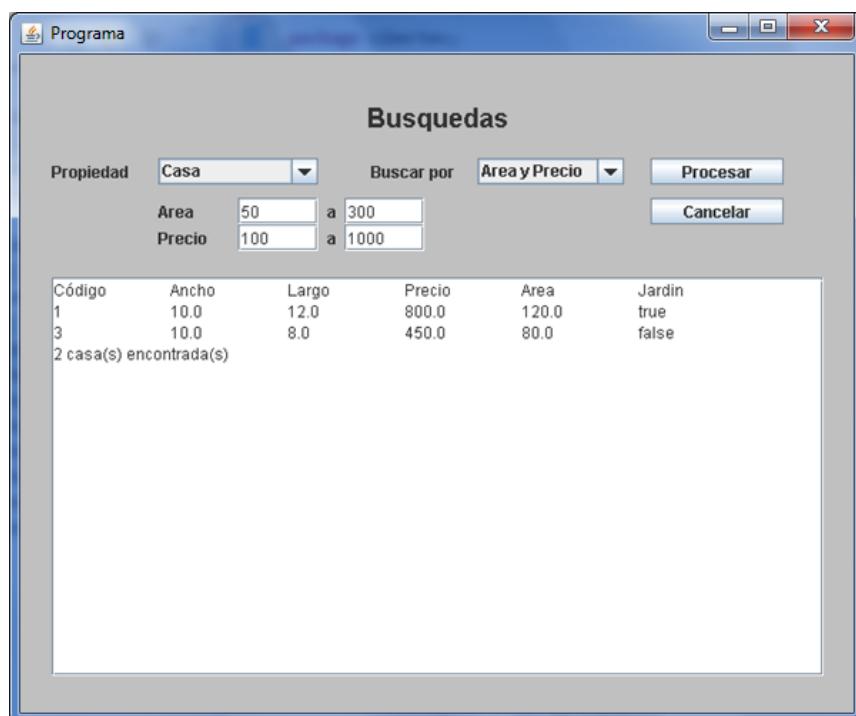
La información ingresada o modificada actualizará los archivos de texto respectivos.



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.



El JTextField de piso se refiere a la ubicación del departamento dentro del edificio o condominio.



Utilice el JComboBox **Propiedad** para seleccionar entre casa o departamento y el JComboBox **Buscar por** para seleccionar el criterio de búsqueda (según area, precio, area y precio, más barato y más caro).

Ingrese los rangos de área y precio según sea el caso y pulse clic al botón Procesar para ejecutar la búsqueda.

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.ArrayList;
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 {
        String linea, s[];
        int codigo, hab;
        double ancho, largo, precio;
        boolean disp, jardin;

        BufferedReader br;
        br = new BufferedReader(new FileReader("Casas.txt"));

        while ((linea = br.readLine()) != null) {
            s = linea.split(":");

            codigo = Integer.parseInt(s[0].trim());
            hab = Integer.parseInt(s[1].trim());
            ancho = Double.parseDouble(s[2].trim());
            largo = Double.parseDouble(s[3].trim());
            precio = Double.parseDouble(s[4].trim());
            disp = Boolean.parseBoolean(s[5].trim());
            jardin = Boolean.parseBoolean(s[6].trim());

            aCasas.add(new Casa(codigo,
                                hab,
                                ancho,
                                largo,
                                precio,
                                disp,
                                jardin));
        }

        br.close();
    } catch(Exception e) {
    }
}
```

```

public void grabarCasas() {
    try {
        PrintWriter pw = new PrintWriter(new FileWriter("Casas.txt"));
        String linea;
        for (int i=0; i<tamaño(); 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 e) {
    }
}
}

```

Clase ArregloDepartamentos

```

package arreglos;

import java.util.ArrayList;
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<tamaño(); 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<tamaño(); 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<tamaño(); i++)
        if (aDptos.get(i).getPrecio() > mayor && aDptos.get(i).getDisp())
            mayor = aDptos.get(i).getPrecio();

    return mayor;
}

public void cargarDepartamentos() {
    try {
        String      linea, s[];
        int         codigo, hab, piso;
        double      ancho, largo, precio;
        boolean     disp;

        BufferedReader br;
        br = new BufferedReader(new FileReader("Departamentos.txt"));

        while ((linea = br.readLine()) != null) {
            s = linea.split(";");
            codigo = Integer.parseInt(s[0].trim());
            hab = Integer.parseInt(s[1].trim());
            ancho = Double.parseDouble(s[2].trim());
            largo = Double.parseDouble(s[3].trim());
            precio = Double.parseDouble(s[4].trim());
            disp = Boolean.parseBoolean(s[5].trim());
            piso = Integer.parseInt(s[6].trim());

            aDptos.add(new Departamento(codigo,
                                         hab,
                                         ancho,
                                         largo,
                                         precio,
                                         disp,
                                         piso));
        }

        br.close();
    }
    catch(Exception e) {
    }
}
```

```

public void grabarDepartamento() {
    try {
        PrintWriter pw;
        pw = new PrintWriter(new FileWriter("Departamentos.txt"));
        String linea;
        for (int i=0; i<tamaño(); 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 e) {
    }
}
}

```

Clase LibGUI (librería)

```

package compartido;

import javax.swing.*;

public class LibGUI {
    public LibGUI() {
    }

    public static int getInteger(JTextField t) {
        return Integer.parseInt(t.getText());
    }

    public static double getDouble(JTextField t) {
        return Double.parseDouble(t.getText());
    }
}

```

Clase Programa (principal)

```

package cibertec;

import arreglos.*;
import java.awt.BorderLayout;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Font;

```

```
public class Programa extends JFrame {  
  
    private JPanel contentPane;  
  
    public PPrincipal pnlPrincipal;  
    public PCasa pnlCasa;  
    public PDepartamento pnlDpto;  
    public PBusqueda pnlBusqueda;  
  
    public ArregloCasas aCas = new ArregloCasas();  
    public ArregloDepartamentos aDpt = new ArregloDepartamentos();  
  
    /** Launch the application. */  
    public static void main(String[] args) {  
        EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                try {  
                    Programa frame = new Programa();  
                    frame.setVisible(true);  
                } catch (Exception e) {  
                    e.printStackTrace();  
                }  
            }  
        });  
    }  
  
    /** Create the frame. */  
    public Programa() {  
        setTitle("Programa");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 400);  
  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        contentPane.setLayout(null);  
  
        pnlPrincipal = new PPrincipal(this);  
        pnlPrincipal.setBounds(0, 0, 600, 500);  
        pnlPrincipal.setVisible(true);  
        contentPane.add(pnlPrincipal);  
  
        pnlCasa = new PCasa(this);  
        pnlCasa.setBounds(0, 0, 600, 500);  
        pnlCasa.setVisible(false);  
        contentPane.add(pnlCasa);  
  
        pnlDpto = new PDepartamento(this);  
        pnlDpto.setBounds(0, 0, 600, 500);  
        pnlDpto.setVisible(false);  
        contentPane.add(pnlDpto);  
  
        pnlBusqueda = new PBusqueda(this);  
        pnlBusqueda.setBounds(0, 0, 600, 500);  
        pnlBusqueda.setVisible(false);  
        contentPane.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 Programa pro;

    public PPrincipal(Programa pro) {
        this.pro = pro;

        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 arg0) {
        if (arg0.getSource() == btnMtoCasas) {
            actionPerformedBtnMtoCasas(arg0);
        }

        if (arg0.getSource() == btnMtoDptos) {
            actionPerformedBtnMtoDptos(arg0);
        }
    }
}
```

```

        if (arg0.getSource() == btnBusquedas) {
            actionPerformedBtnBusquedas(arg0);
        }
    }

    protected void actionPerformedBtnMtoCasas(ActionEvent arg0) {
        pro.pnlPrincipal.setVisible(false);
        pro.pnlCasa.setVisible(true);
    }

    protected void actionPerformedBtnMtoDptos(ActionEvent arg0) {
        pro.pnlPrincipal.setVisible(false);
        pro.pnlDpto.setVisible(true);
    }

    protected void actionPerformedBtnBusquedas (ActionEvent arg0) {
        pro.pnlPrincipal.setVisible(false);
        pro.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 Programa pro;  
  
    private boolean flagDispo = false;  
    private boolean flagJardin = false;  
    private boolean existe = false;  
  
    private JCheckBox chkJardin;  
  
    public PCasa(Programa pro) {  
        this.pro = pro;  
  
        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);  
  
        pro.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(){
    pro.pnlCasa.setVisible(false);
    pro.pnlPrincipal.setVisible(true);

    habilitarControles();
    limpiarControles();
    existe=false;
    cambio();
}

public void ingresar() {
    int cod=LibGUI.getInteger(txtCodigo);
    Casa Ocasa=pri.aCas.buscar(cod);
    if(Ocasa==null){
        double ancho=LibGUI.getDouble(txtAncho);
        double largo=LibGUI.getDouble(txtLargo);
        double precio=LibGUI.getDouble(txtPrecio);
        int hab=LibGUI.getInteger(txtHab);
        Ocasa=new Casa(cod,hab,ancho,largo,precio,flagDispo,flagJardin);
        pri.aCas.agregar(Ocasa);
        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 Ocasa=pri.aCas.buscar(cod);
        if(Ocasa!=null){
            txtAncho.setText(""+Ocasa.getAncho());
            txtLargo.setText(""+Ocasa.getLargo());
            txtPrecio.setText(""+Ocasa.getPrecio());
            txtHab.setText(""+Ocasa.getHab());

            boolean flag1=Ocasa.getDisp();
            if(flag1)
                chkDispo.setSelected(true);
            else
                chkDispo.setSelected(false);

            boolean flag2=Ocasa.getJardin();
            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 Ocasa=pri.aCas.buscar(cod);
        Ocasa.setAncho(LibGUI.getDouble(txtAncho));
        Ocasa.setLargo(LibGUI.getDouble(txtLargo));
        Ocasa.setPrecio(LibGUI.getDouble(txtPrecio));
        Ocasa.setHab(LibGUI.getInteger(txtHab));
        Ocasa.setDisp(flagDispo);
        Ocasa.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);
    pro.aCas.eliminar(pro.aCas.buscar(cod));
    pro.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 Programa pro;

    private boolean flagDispo=false, existe=false;

    private JLabel lblPiso;
    private JTextField txtPiso;

    public PDepartamento(Principal x) {
        pro = 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);

        pro.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();
        }
        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\tJardín");
    }
}
```

```

        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\tJardín");
        for (int i=0; i<pro.aCas.tamaño(); i++) {
            Propiedad prop = pro.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<pro.aDpt.tamaño(); i++ ) {
            Propiedad prop = pro.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_caro() {
    int indice = cboTipo.getSelectedIndex(),
    int conta = 0;
    imprimir();
    if (índice == 0) {
        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tJardín");
        for (int i=0; i<pro.aCas.tamaño(); i++) {
            Propiedad prop = pro.aCas.obtener(i);
            if (prop.getDisp() &&
                prop.getPrecio() == pro.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<pro.aDpt.tamaño(); i++ ) {
            Propiedad prop = pro.aDpt.obtener(i);
            if (prop.getDisp() &&

```

```
        prop.getPrecio() == pro.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 (índice == 0) {
        imprimir("Código\tAncho\tLargo\tPrecio\tArea\tJardín");
        for (int i=0; i<pro.aCas.tamaño(); i++) {
            Propiedad prop = pro.aCas.obtener(i);
            if (prop.getDisp() &&
                prop.getPrecio() == pro.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<pro.aDpt.tamaño(); i++) {
            Propiedad prop = pro.aDpt.obtener(i);
            if (prop.getDisp() &&
                prop.getPrecio() == pro.aDpt.precioMenor()) {
                imprimir(prop.comoCadena());
                conta++;
            }
        }
        imprimir(conta + " departamento(s) encontrado(s)");
    }
}

public void imprimir() {
    txtS.setText("");
}

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

6.2. Miscelanea

Clase ArrayList

Dada la Clase Impresora:

```
public class Impresora {  
    // Atributos privados  
    private int codProducto, codBarras;  
    private String marca;  
    private double precio;  
  
    // Constructor  
    public Impresora(int codProducto, int codBarras,  
        String marca, double precio) {  
        this.codProducto = codProducto;  
        this.codBarras = codBarras;  
        this.marca = marca;  
        this.precio = precio;  
    }  
  
    // Métodos de acceso set/get  
    public void setCodProducto(int codProducto) {  
        this.codProducto = codProducto;  
    }  
  
    public void setCodBarras(int codBarras) {  
        this.codBarras = codBarras;  
    }  
  
    public void setMarca(String marca) {  
        this.marca = marca;  
    }  
  
    public void setPrecio(double precio) {  
        this.precio = precio;  
    }  
  
    public int getCodProducto() {  
        return codProducto;  
    }  
  
    public int getCodBarras() {  
        return codBarras;  
    }  
  
    public String getMarca() {  
        return marca;  
    }  
  
    public double getPrecio() {  
        return precio;  
    }  
}
```

Diseñe la Clase ArregloImpresoras que tenga como atributo privado un objeto de la clase ArrayList y los métodos necesarios para realizar un mantenimiento de impresoras.

```
import java.util.ArrayList;

public class ArregloImpresoras {
    // Atributo privado
    private ArrayList <Impresora> imp;

    // Constructor
    public ArregloImpresoras() {
        imp = new ArrayList <Impresora> ();
        adicionar(new Impresora(4072,23843, "Epson",1350.0));
        adicionar(new Impresora(2561,82402, "HP",2300.0));
        adicionar(new Impresora(7286,39129, "Samsung",2100.0));
        adicionar(new Impresora(9114,50085, "Sony",1900.0));
        adicionar(new Impresora(1498,18553, "LG",1050.0));
        adicionar(new Impresora(5833,94897, "Samsung",4210.0));
        adicionar(new Impresora(8300,42321, "Epson",2090.0));
        adicionar(new Impresora(2615,67200, "LG",3520.0));
    }

    // Operaciones
    public void adicionar(Impresora x) {
        imp.add(x);
    }

    public Impresora obtener(int i) {
        return imp.get(i);
    }

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

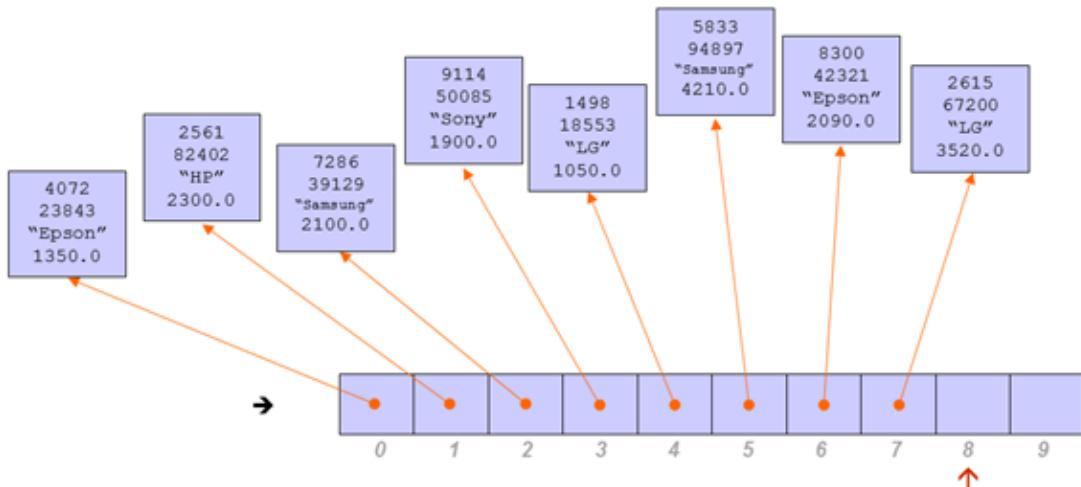
    public void eliminar(Impresora x) {
        imp.remove(x);
    }

    public void reiniciarArreglo() {
        if (tamaño() > 0)
            imp.clear();
    }
}
```

Dada la declaración global en el programa principal

```
ArregloImpresoras ai = new ArregloImpresoras();
```

Interpretación gráfica



Diseñe en el programa principal un método que visualice la información completa de las impresoras ingresadas hasta ese momento.

```

protected void actionPerformedBtnListar(ActionEvent arg0) {
    listar();
}

void listar() {
    if (ai.tamaño() == 0)
        imprimir("el ArrayList de impresoras está vacío");
    else {
        Impresora x;
        imprimir("imp\tcodProd.\tcodBar\tmarca\tprecio");
        for (int i=0; i<ai.tamaño(); i++) {
            x = ai.obtener(i);
            imprimir(i + "\t" + x.getCodProducto()
                    + "\t" + x.getCodBarras()
                    + "\t" + x.getMarca()
                    + "\t" + x.getPrecio());
        }
    }
}

```

Añada en la Clase ArregloImpresoras un método que reciba un código de producto y retorne la referencia al objeto tipo Impresora que lo contiene. En caso no exista retorne null.

```

public Impresora buscarCodProducto(int codProducto) {
    Impresora x;
    for (int i=0; i<tamaño(); i++) {
        x = obtener(i);
        if (x.getCodProducto() == codProducto)
            return x;
    }
    return null;
}

```

Añada en la Clase ArregloImpresoras un método que reciba un código de barras y retorne la referencia al objeto tipo Impresora que lo contiene. En caso no exista retorne null.

```
public Impresora buscarCodProducto(int codBarras) {
    Impresora x;
    for (int i=0; i<tamaño(); i++) {
        x = obtener(i);
        if (x.getCodBarras() == codBarras)
            return x;
    }
    return null;
}
```

Diseñe en el programa principal un método que adicione una nueva impresora validando que los códigos de producto y de barra no se repitan. Los métodos leerCodProducto(), leerCodBarras(), leerMarca() y leerPrecio() obtienen datos de la GUI. El método mensaje(String) muestra una ventana emergente por GUI.

```
protected void actionPerformedBtnIngresar(ActionEvent arg0) {
    Impresora x;
    x = ai.buscarCodProducto(leerCodProducto());
    if (x == null) {
        x = ai.buscarCodBarras(leerCodBarras());
        if (x == null) {
            x = new Impresora(leerCodProducto(),leerCodBarras(),
                               leerMarca(),leerPrecio());
            ai.adicionar(x);
            listar();
            mensaje("impresora ingresada");
        } else
            mensaje("código de barras ya existe");
    } else
        mensaje("código de producto ya existe");
}
```

Diseñe en el programa principal un método que visualice la información completa de las impresoras pero de una determinada marca.

```
protected void actionPerformedBtnReportar(ActionEvent arg0) {
    Impresora x;
    String marca = leerMarca();
    txtS.setText("");
    for (int i=0; i< ai.tamaño(); i++) {
        x = ai.obtener(i);
        if (x.getMarca().equals(marca))
            imprimir(i + "\t" + x.getCodProducto()
                      + "\t" + x.getCodBarras()
                      + "\t" + x.getMarca()
                      + "\t" + x.getPrecio());
    }
}
```

Añada en la Clase ArregloImpresoras un método que reciba una marca de impresora e incremente en 2.75% el precio de todas las impresoras de dicha marca.

```
public void aumentarPrecioDeMarca(String marca) {
    Impresora x;
    for (int i=0; i<tamaño(); i++) {
        x = obtener(i);
        if (x.getMarca().equals(marca))
            x.setPrecio(1.0275*x.getPrecio());
    }
}
```

Diseñe en el programa principal un método que invoque al método anterior.

```
protected void actionPerformedBtnAumentarPrecioDeMarca(ActionEvent arg0) {
    ai.aumentarPrecioDeMarca(leerMarca());
    listar();
}
```

Añada en la clase ArregloImpresoras un método que reciba un código de producto, lo busque y si existe elimine el objeto respectivo.

```
public void eliminarPorCodProducto(int codProducto) {
    Impresora x = buscarCodProducto(codProducto);
    if (x != null)
        eliminar(x);
}
```

Diseñe en el programa principal un método que invoque al método anterior.

```
protected void actionPerformedBtnEliminarPorCodProducto(ActionEvent arg0) {
    ai.eliminarPorCodProducto(leerCodProducto());
    listar();
}
```

Añada en la Clase ArregloImpresoras un método que reciba una marca de impresora y retorne la referencia al objeto tipo Impresora que la contiene. En caso no exista retorne null.

```
public Impresora buscarMarca(String marca) {
    Impresora x;
    for (int i=0; i<tamaño(); i++) {
        x = obtener(i);
        if (x.getMarca().equals(marca))
            return x;
    }
    return null;
}
```

Diseñe en el programa principal un método que invoque al método anterior para eliminar todas las impresoras de una determinada marca.

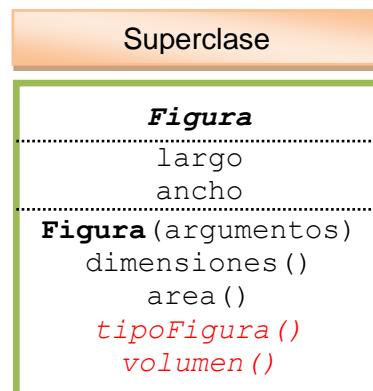
```
protected void actionPerformedBtnEliminarMarca(ActionEvent arg0) {  
    Impresora x;  
    String marca = leerMarca();  
    while ((x = ai.buscarMarca(marca)) != null) {  
        ai.eliminar(x);  
    }  
    listar();  
}
```

Diseñe en el programa principal un método que busque un código de producto y si existe modifique el código de barras de la impresora verificando que el código de barras no se repita.

```
protected void actionPerformedBtnBuscarModificar(ActionEvent arg0) {{  
    Impresora x = ai.buscarCodProducto(leerCodProducto());  
    if (x != null) {  
        int codBarras = leerCodBarras();  
        Impresora y = ai.buscarCodBarras(codBarras);  
        if (y == null) {  
            x.setCodBarras(codBarras);  
            listar();  
            mensaje("código de barras cambiado");  
        } else  
            mensaje("código de barras repetido");  
    } else  
        mensaje("código de producto no existe");  
}
```

Herencia y Polimorfismo

Dada la Clase abstracta Figura:



La representación de las clases abstractas se hace colocando su nombre en cursiva. Del mismo modo, los métodos u operaciones abstractos, también se colocan en cursiva.

```

public abstract class Figura {
    // Métodos protegidos
    protected double largo, ancho;

    // Constructor
    public Figura(double largo, double ancho) {
        this.largo = largo;
        this.ancho = ancho;
    }

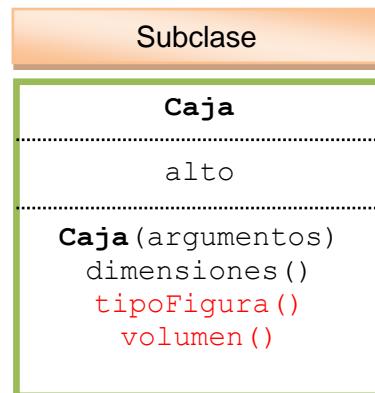
    // Operaciones
    public String dimensiones() {
        return "Largo : " + largo + "\n" +
               "Ancho : " + ancho;
    }

    public double area() {
        return largo*ancho;
    }

    // Métodos abstractos
    public abstract String tipoFigura();

    public abstract double volumen();
}
  
```

Diseñe la Clase hijo Caja que herede de la Clase Padre Figura. A través del Constructor reciba las dimensiones. Derive las dos primeras a la Clase Padre. Implemente el método sobrescrito dimensiones() que retorne en una cadena las medidas de la caja y los métodos abstractos que obliga implementar la Clase Padre.



```

public class Caja extends Figura {
    // Métodos protegidos
    protected double alto;

    // Constructor
    public Caja(double largo, double ancho, double alto) {
        super(largo, ancho);
        this.alto = alto;
    }

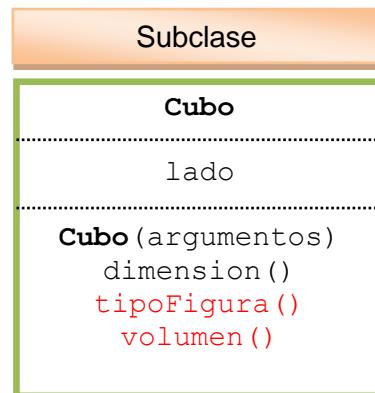
    // Operaciones
    public String dimensiones() {
        return super.dimensiones() + "\n" + "Alto:" + alto;
    }

    // Desarrollo de métodos abstractos
    public String tipoFigura() {
        return "ES UNA CAJA";
    }

    public double volumen() {
        return area()*alto;
    }
}

```

Diseñe la Clase hijo Cubo que herede de la Clase Padre Figura. A través del Constructor reciba el lado del cuadrado. Derive a la Clase Padre el lado en dos parámetros. Implemente el método dimension() que retorne en una cadena la medida del lado y los métodos abstractos que obliga implementar la Clase Padre.



```
public class Cubo extends Figura {  
    // Métodos protegidos  
    protected double lado;  
  
    // Constructor  
    public Cubo(double lado) {  
        super(lado,lado);  
        this.lado = lado;  
    }  
  
    // Operaciones  
    public String dimension() {  
        return "Lado : " + lado;  
    }  
  
    // Métodos abstractos  
    public String tipoFigura() {  
        return "ES UN CUBO";  
    }  
  
    public double volumen() {  
        return area()*lado;  
    }  
}
```

Diseñe en el programa principal un método que declare y cree dos objetos, uno de tipo Caja y otro de tipo Cubo con datos fijos. Luego a través de un método listado, aplique polimorfismo, upcasting y downcasting según sea el caso.

```
protected void actionPerformedBtnProcesr(ActionEvent arg0) {  
    Caja ca = new Caja(10,15,20);  
    listado(ca);  
  
    Cubo cu = new Cubo(30);  
    listado(cu);  
}  
  
void listado(Figura x) {  
    imprimir("Tipo de figura : " + x.tipoFigura());  
  
    if (x instanceof Caja)  
        imprimir(x.dimensiones());  
    else  
        imprimir(((Cubo)x).dimension());  
  
    imprimir("Volumen : " + x.volumen());  
}
```

Consideraciones:

El **polimorfismo** se produce en el método listado al momento de recibir el parámetro a nombre de la Clase Padre **Figura**.

La **sobreescritura** se da en la Clase Hijo **Caja** al momento de codificar el método `dimensiones()`.

Si la **Figura** recibida es del tipo **Caja** entonces se ejecuta el método sobrescrito `dimensiones()` de la Clase Hijo **Caja**: es aquí donde el Java aplica **upcasting**.

Al ejecutar los métodos `tipoFigura()` y `volumen()` también se aplica **upcasting**.

Si la **Figura** recibida es del tipo **Cubo** entonces para poder ejecutar el método `dimension()` de la Clase Hijo **Cubo** es necesario castear el objeto recibido: es aquí donde el Java aplica **downcasting**.