Universidad Autónoma del Estado de México
Centro Universitario UAEM Texcoco

Departamento de Ciencias Aplicadas.

Ingeniería en Computación.

Lenguaje de Programación Orientado a Objetos.

Unidad de competencia III: “Enumeraciones”

Presenta:

M. en C. C. J. Jair Vázquez Palma.

Lenguaje de Programación Orientado a Objetos

Objetivo de la Unidad de la Unidad de Aprendizaje

Familiarizar al alumno con un lenguaje de programación orientado a objetos, los patrones de diseño en las que se sustentan algunas de las clases en interfaces de su API, así como con una plataforma de desarrollo, que le permita la aplicación de ésta para el desarrollo de software de forma eficiente y efectiva.

CONTENIDO DE UNIDAD 3.

Enumeraciones.

- Tipos de clases.
- Aspectos de diseño aplicables para enumeraciones.
- Características de una enumeración.
- Objetos de enumeraciones.
- Métodos y constructores en enumeraciones.
- Uso de enumeraciones.

Objetivos de la Unidad de Competencia III

- Conocer los tipos de clases en Java.
- Conocer los aspectos de diseño aplicables a enumeraciones.
- Implementación de objetos de enumeraciones.
- Uso de enumeraciones.

Enumeraciones, tipos de clases.

Tipo de clases:

- Interfaces.
- Abstractas.
- Internas.
  - Internas miembro.
  - Internas locales.
  - Internas *static.*
- Clases anónimas.
- Clases derivadas (herencia).

Tipos de clases.

Interfaces:

- Java incorpora en su lenguaje la declaración de *interface,* que permite enunciar un conjunto de constantes y de cabeceras de métodos abstractos; éstos deben implementarse en las clases y constituyen la interfaz de la clase.

- En cierto modo, es una forma de declarar que todos lo métodos de una clase son públicos y abstractos, con ello se especifica el comportamiento común de todas las clases que implementen la interfaz.

Tipos de clases.

Interfaces:

- Su declaración es similar a la de una clase; en la cabecera se utiliza la palabra reservada *interface* en vez de *class, por ejemplo:*

```
public interface NodoG{
      boolean igual(NodoG t);
      NodoG asignar(NodoG t);
      void escribir(NodoG t);
}
```

La interfaz *nodo* define 3 métodos abstractos y además públicos; sin embargo, no debe especificarse ni *abstract* ni *public ya que los métodos de la* interface lo son.

Tipos de clases.

Interfaces: Sintaxis.

```
acceso interface NombreInterface{
      constante1;
      …
      constanten;

      tipo1 nombreMetodo1(argumentos);
      …
      tipon nombreMetodon(argumentos);
}
```

acceso visibilidad de la interfaz definida, normalmente *public.*

Tipos de clases.

Múltiples Interfaces:

- Java no permite que una clase derive de dos o mas clases, es decir, no permite herencia múltiple.

- Sin embargo, una clase sí puede implementar mas de una interfaz y tener el comportamiento común de varias de ellas, para esto.

- Sintaxis:

```
class NombreClase implements Interfaz1, Interfaz2, …, Interfazn{

}
```

Tipos de clases.

Múltiples Interfaces: Ejercicio.

Revisar el ejercicio en código de Java sobre implementación de una interfaz.

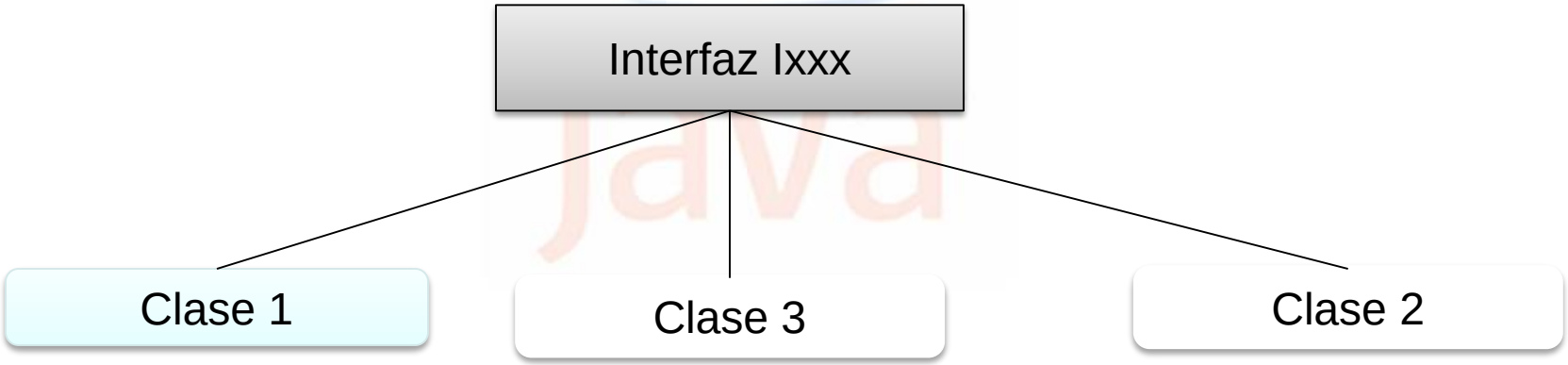
Tarea: Terminar el programa creando el main() para su ejecución.

Tipos de clases.

Utilizar una interfaz como un tipo.

- Una interfaz es un nuevo tipo de datos; un tipo referenciado. Por lo tanto, el nombre de una interfaz se puede utilizar en cualquier lugar donde pueda aparecer el nombre de cualquier otro tipo de datos.

- Una variable del tipo interfaz espera referenciar un objeto que tenga implementada dicha interfaz, de lo contrario el compilador Java mostrará un error.



Tipos de clases.

Utilizar una interfaz como un tipo.

- Ejemplo que muestra tres clases no relacionadas pero por el hecho de implementar la misma interfaz permite definir una matriz de objetos de esas clases y aplicar la definición de polimorfismo.

```
public interface Ixxx{
        public abstract void m();
        public abstract void p();
}
public class Clase1 implements Ixxx{
public void m(){
        System.out.println("método m de clase1");
}
public void p(){}
}
public class Clase2 implements Ixxx{
public void m(){
        System.out.println("método m de clase2");
}
public void p(){}
}
```

Tipos de clases.

Utilizar una interfaz como un tipo.

```
public class Clase3 implements Ixxx{
public void m(){
        System.out.println("método m de clase3");
}
public void p(){}
}

public class Test{
public static void main(String args[]){
        Ixxx[] objs = new Ixxx[3]; //matriz de referencia a objetos
        objs[0] = new Clase1();
        objs[1] = new Clase2();
        objs[2] = new Clase3();
        for(int i = 0; i< objs.lenght; i++)
                objs[].m();//invoca al método m del objeto Clase1.

        }
}
```

Tipos de clases.

Interfaces frente a herencia múltiple.

A menudo se piensa en las interfaces como en una alternativa a la herencia múltiple, pero en realidad es que ambos conceptos son bastante diferentes, a pesar de que las interfaces pueden resolver problemas similares. En particular:

- Desde una interfaz, una clase sólo hereda constantes.
- Desde una interfaz, una clase no puede heredar definiciones de métodos.
- La jerarquía de interfaces es independiente de la jerarquía de clases, de hecho varias clases pueden implementar la misma interfaz y no pertenecer a la misma jerarquía de clases; sin en cambio la herencia múltiple, todas las clases pertenecen a la misma jerarquía.

Tipos de clases.

Para qué sirve una interfaz.

Una interfaz se utiliza para definir un protocolo de conducta que puede ser implementado por cualquier clase en una jerarquía de clases. La utilidad se resume en:

- Captar similitudes entre clases no relacionadas sin forzar entre ellas una relación artificial.
- Declarar métodos que una o más clases deben implementar en determinadas situaciones.
- Publicar la interfaz de programación de una clase sin describir cómo está implementada.

Tipos de clases.

Abstractas:

- Representan conceptos generales, engloban características comunes de un conjunto de objetos.
- Es una clase que engloba las propiedades y métodos comunes del objeto.
- En java el modificador *abstract* declara una clase abstracta.

```
abstract class Nombre clase {…}
```

- Por ejemplo:

```
public abstract class Persona{
        private String apellido;
        //
        public void identificacion (String a, String c){…}
}
```

Tipos de clases.

Abstractas:

- Las clases abstractas declaran métodos y variables instancia, y normalmente tienen métodos abstractos; si una clase tiene un método abstracto debe declararse abstracta.

- Una característica importante de estas clases es que ellas no se pueden definir objetos, es decir, no se puede instanciar de una clase abstracta; el compilador devuelve un error siempre que se intenta crear un objeto de dichas clases. Por ejemplo:

Tipos de clases.

Abstractas:

```
public abstract class Metal {…}

Metal mer = new Metal(); //Error: no se puede
                         instanciar de clase
                         abstracta.
```

- Las clases abstractas están en lo más alto de la jerarquía de clases, son superclases base y, por consiguiente, siempre se establece una conversión automática de clases derivada a base abstracta.

Tipos de clases.

Clase abstracta frente a interfaz:

¿Qué diferencia existe entre una interfaz y una clase abstracta?

- Una interfaz es simplemente una lista de constantes y métodos abstractos. Una interfaz no es más que un protocolo que una clase implementa cuando necesita utilizarlo.

- Una clase abstracta debe forzar una relación entre clases, lo único que importa es implementar uno o mas métodos específicos.

Tipos de clases.

Clase abstracta o interfaz?

- En algunas situaciones se tiene que elegir entre usar una clase o una interfaz. Algunas veces la elección es fácil: cuando se pretende que la clase contenga implementaciones para algunos métodos necesitamos usar una clase abstracta. En otros casos tanto la clase abstracta como la interfaz pueden hacer el mismo trabajo.

- Si tenemos que elegir, es preferible usar interfaces. Si proveemos un tipo mediante una clase abstracta, las subclases no pueden extender ninguna otra clase; dado que las interfaces permiten la herencia múltiple el uso de una interfaz no crea tal restricción. Por lo tanto, el uso de interfaces da por resultado una estructura mas flexible y más extensible.

Tipos de clases.

Internas:

- Una clase interna es la que se declara dentro de otra clase, se puede decir que es anidada.

- Las clases internas declaran atributos y métodos de igual forma que las externas o de nivel superior, con la peculiaridad de que sus métodos pueden acceder a los atributos de su clase externa. Hay cuatro tipos:

  - Miembro de una clase.
  - Locales.
  - *static.*
  - Anónimas.

Tipos de clases.

Internas: Ejemplo.

```
public class Alumno {
        int edad;
        String nombre;
        Direccion direc;

        public Alumno(String nom, int num, String calle, String city,
        String codePostal){
                direc = new Direccion(calle, num, city, codePostal);
                nombre = nom;
}
```

Tipos de clases.

Internas: Ejemplo.

```
//Clase interna
class Direccion{
        int numero;
        String calle;
        String ciudad;
        String code;

public Direccion(String c, int n, String ct, String d){
        calle = c;
        numero = n;
        ciudad = ct;
        code = d;
}
void escribirDir(){ }
//Sigue la definición de los métodos de la clase Alumno.
}
}
```

Tipos de clases.

Anónimas:

- Este tipo de clases internas no se identifican con un nombre, se definen a la vez que se crea la instancia u objeto de la clase.
- De ellas no se pueden crear múltiples objetos, sólo es ligado con la definición.
- Las clases anónimas no sólo pueden implementar una interfaz sino que también pueden derivar o extender de una clase.

Tipos de clases.

Anónimas: Sintaxis.

a) Al implementar una interfaz:

```
new TipoInterface(){
        //miembros de objeto anónimo
}
```

b) Al extender de una clase:

```
New TipoClase(argumentos_construcción_claseExtendida){
//miembros de objeto anónimo
}
```

c) En caso de no implementar ni extender:

```
New {
//miembros de objeto anónimo
}
```

Tipos de clases.

Anónimas: Sintaxis.

Se escribe un método cuyo argumento es del tipo `interface Inmaterial`:

```
public void mostrar(Inmaterial q){
        String px;
        px = q.datCadena();
        System.out.println("Muestra: " + px);
}
```

Objeto anónimo que implementa a `Inmaterial`:

```
mostrar(new Inmaterial() {
        public String datCadena(){
                return "Cadena de objeto 1";
                }
        }
);
```

Tipos de clases.

Anónimas: Sintaxis.

Otra llamada:

```
mostrar ( new Inmaterial (){
        public String datCadena(){
                return "Cadena desde objeto 2"
                }
        }
);
```

Revisar ejemplo en Java.

Tipos enumerados.

Enumeraciones:

- A partir de la versión Java SE 5.0 se pueden definir tipos de datos enumerados o enumeraciones que constan de diferentes elementos, especifican diversos valores por medio de identificadores y son definidos por el programador mediante la palabra reservada *enum.*

- Un tipo enumerado define un conjunto de constantes que se representan como identificadores únicos.

- Cada constante `enum` dentro de este tipo toma un valor especifico denominado ordinal, el de la primera constante `enum` es 0, el de la segunda es 1, y así sucesivamente. Ejemplo:

```
enum DEPORTES{TENIS, ESQUÍ, FUTBOL, BALONCESTO, GOLF };
```

Tipos enumerados.

Enumeraciones:

```
enum DEPORTES{TENIS, ESQUÍ, FUTBOL, BALONCESTO, GOLF )
```

- Cada *enum* es un tipo especial de clase y los valores que le pertenecen son objetos de la clase.
- Después de definir un tipo *enum,* se pueden declarar variables con referencia a tal tipo.

```
DEPORTES miDeporte;
```

- Se le puede asignar un valor a la variable.

```
miDeporte = DEPORTES.TENIS;
```

Tipos enumerados.

Enumeraciones: Reglas.

- Los tipos enumeración se definen utilizando la palabra reservada *enum*, en lugar de *class*.
- Las constantes *enum* son implícitamente estáticas (*static*).
- Los tipos *enum* son tácitamente de tipo *final*, ya que declaran constantes que no pueden modificarse.
- Una vez que se crea un tipo enumeración se pueden declarar variables referencia de este tipo, pero no se pueden instanciar objetos utilizando el operador *new*, si se intenta instanciar un objeto utilizándolo se producirá un error de compilación.
- Dado el punto anterior, si existe constructor de tipo enumeración, éste no puede ser público(*public*) por que es implícitamente privado (*private*).

Tipos enumerados.

Enumeraciones: Métodos.

Método	Descripción
ordinal ()	Describe el valor ordinal de una constante `enum`.
name ()	Devuelve el nombre del valor de `enum`.
values ()	Devuelve los valores de un tipo `enum` en forma de lista.

Enumeraciones. Aspectos de diseño.

- Java, como los restantes lenguajes de programación, suministra una serie de tipos de datos básicos y una serie de operaciones para su manipulación.
- En ocasiones, estos tipos de datos no son los adecuados para resolver un problema y es necesario crear nuevos tipos de datos, que será preciso definir especificando tanto sus datos componentes como las operaciones que los manipulan.
- Estos tipos de datos se definen mediante clases y proporcionan una de las características más importantes de la *PO0* (Programación Orientada a Objetos), la *reutilización de componentes.*

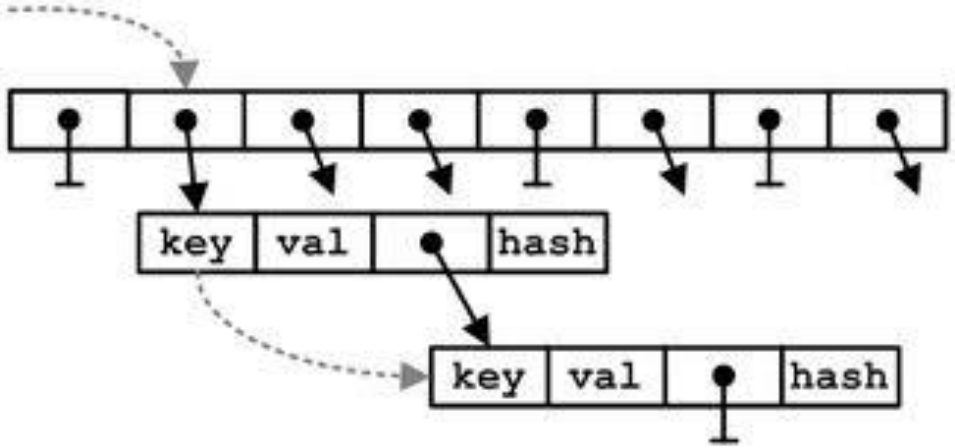
Enumeraciones. Aspectos de diseño.

- Las colecciones de datos organizados y a las que se accede de forma perfectamente definida constituyen una *estructura* ***de*** *datos.*
- Existe un cierto número de estructuras de datos de reconocida utilidad, cuyo uso se repite frecuentemente en los programas, y que pueden llegar a considerarse como una extensión de los tipos básicos del lenguaje; estas estructuras tienen un nombre y unas características establecidas y entre ellas destacan: *listas, pilas, colas, árboles.*
- Dada la importancia de las estructuras de datos, Java ofrece clases que implementan algunas de ellas: java. utiL .vector, java.util.Stack, java.util.Hastable, java.util.BitSet.

Tipos enumerados. Características.

- Las enumeraciones nos sirven para definir listas enumeradas de elementos y algo más, ya que en cierto modo son como clases, pueden tener constructores, métodos y atributos.

- El primer elemento de la enumeración tiene la posición 0. La única restricción que tiene las enumeraciones sobre las clases es que las enumeraciones no se pueden extender.



Uso de clases enumeradas:

Clase Vector
y
ArrayList

Clase Vector

Java proporciona un grupo de clases que almacenan secuencias de objetos de cualquier tipo: las colecciones; estás se diferencian en la forma de organizar los objetos y, en consecuencia, en la manera de recuperarlos.

Clase Vector

La clase Vector se encuentra en el paquete java.util y es una de estas colecciones, tiene un comportamiento similar a un arreglo unidimensional; guarda objetos o referencias de cualquier tipo, crece dinámicamente, sin necesidad de tener que programar operaciones adicionales; el arreglo almacena los elementos de tipo Object, y su declaración es:

```
protected Object elementData[ ];
```

Clase Vector - Constructor

A partir de Java 5 se puede establecer el tipo concreto de elemento, el cual siempre debe ser una clase, que puede guardar una colección, particularmente un vector, para realizar comprobaciones de tipo durante el proceso de compilación; un vector de cadenas por ejemplo:

```
Vector<String> vc = Vector<String>();
```

Creación de un vector

Se utiliza el operador *new* de igual forma que se crea cualquier objeto; la clase Vector dispone de diversos constructores:

- `public Vector(); //crea un vector vacio`
- `public Vector (int capacidad); //crea un vector con capacidad inicial`
- `public Vector (Collection org); // crean un vector con los elementos de org.`

- **Ejemplo:**

- **`Vector v1 = new Vector();`**
- **`Vector v2 = new Vector(100);`**
- **`Vector v3 = new Vector(v2); //v3 contiene los mismo elementos que v2.`**

Insertar elementos.

Hay diferentes métodos para insertar o añadir elementos a un vector; los elementos que se insertan deben ser objetos, no pueden ser datos de tipos primitivos:

Métodos de la clase para insertar:

- `boolean add(Object ob); //añade el objeto después del último elemento del vector.`
- `void addElement(object ob); //añade el objeto después del último elemento del vector.`
- `void insertarElement(Object ob, int p); // inserta el objeto en la posición` ***p*** `; los elementos posteriores a` ***p*** `se desplazan`

Debe considerarse que cuando se crea el vector con un tipo concreto, el elemento que se inserta ha de ser el mismo tipo, ode uno derivado; por ejemplo:

```
Vector<String> vc = new <String>();
vc.add("Jueves");
vc.addElement (new Integer(12)); // error de tipo
```

Acceso a un elemento.

Se accede a un elemento del vector por la posición que ocupa; los métodos de acceso devuelven el elemento con el tipo ***Object***, entonces puede ser necesario realizar una conversión al tipo del objeto:

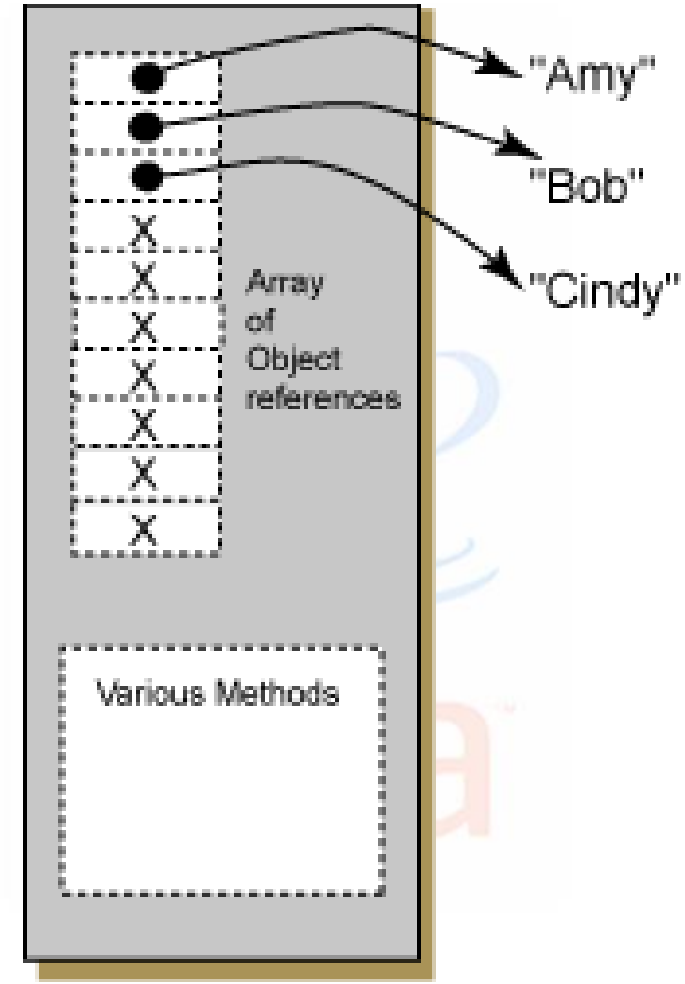
- `Object elementAt(int p); //devuelve el elemento cuya posición es` ***`p`***`.`
- `Object get(int p); //devuelve el elemento cuya posición es` ***`p`***`.`
- `int size(); // devuelve el número de elementos.`

Eliminar un elemento.

Un vector es una estructura dinámica, crece o decrece si se añaden o eliminan objetos; un elemento se puede eliminar de distintas formas, un de ellas es por la posición que ocupa el índice, a partir de esa posición el resto de elementos del vector se mueven una posición a la izquierda disminuyendo el numero de elementos; otra forma es transmitir el objeto que se desea retirar del vector; también hay métodos de la clase para eliminar todo los elementos de una colección.

- `void removeElementAt(int indice); //elimina elemento índice y el resto se reenumera.`
- `boolean removeElement(Object op);//elimina la primera aparición de` ***`op;`*** `devuelve true si realizo la eliminación.`
- `void removeAll(Collection gr); //elimina los elementos que están en` ***`gr`***`.`
- `void removeAllElements(); //elimina todo los elementos.`

ArrayList



After three add()

Clase ArrayList - Constructor

Esta clase agrupa elementos como un arreglo; es equivalente a Vector, pero con las mejoras introducidas en Java2; permite acceder a cualquier elemento, insertar o borrar a partir del índice en cualquier posición, aunque un tanto ineficiente se realiza en posiciones intermedias. A partir de Java 5 es una clase genérica y, por consiguiente, se puede establecer el tipo de concreto de los elementos; esta clase tiene 3 constructores:

- `public ArrayList();`
- `public ArrayList(int capacidad);`
- `public ArrayList(Collection c);`

Por ejemplo, se crea una colección con los elementos de un vector:

```
ArrayList a1= new ArrayList(100);
```

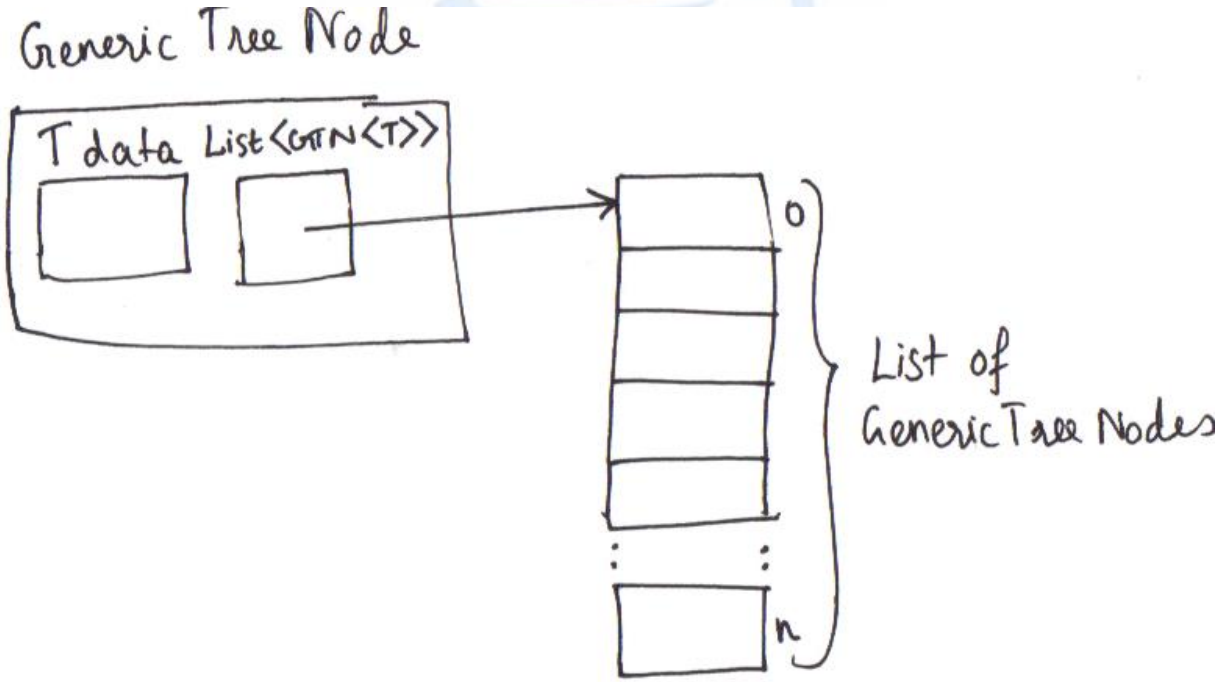
A continuación se crea una colección de elementos tipo `Estudiante`:

```
ArrayList<Estudiante> a1 = new ArrayList<Estudiante>();
```

Clase ArrayList

Se realian las operaciones de añadir, eliminar, buscar y reemplazar cadenas con ArrayList.

Ejemplo. Revisar programa en Java.



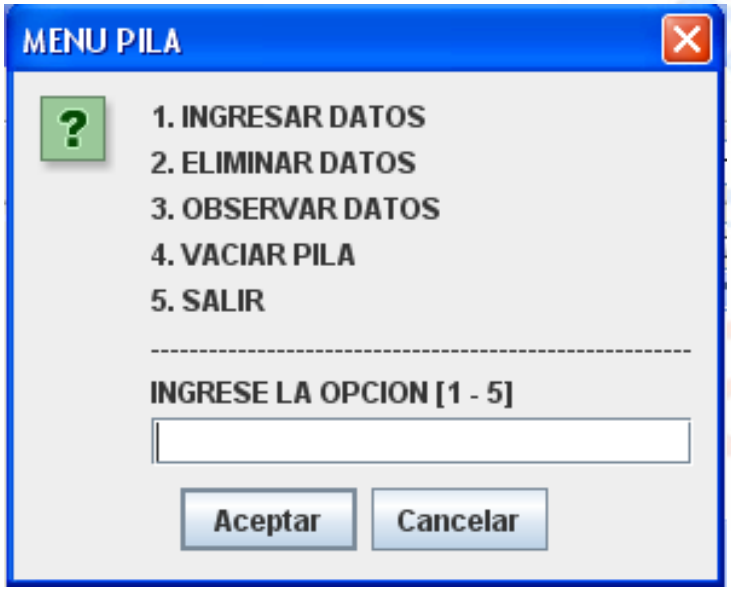
Búsqueda.

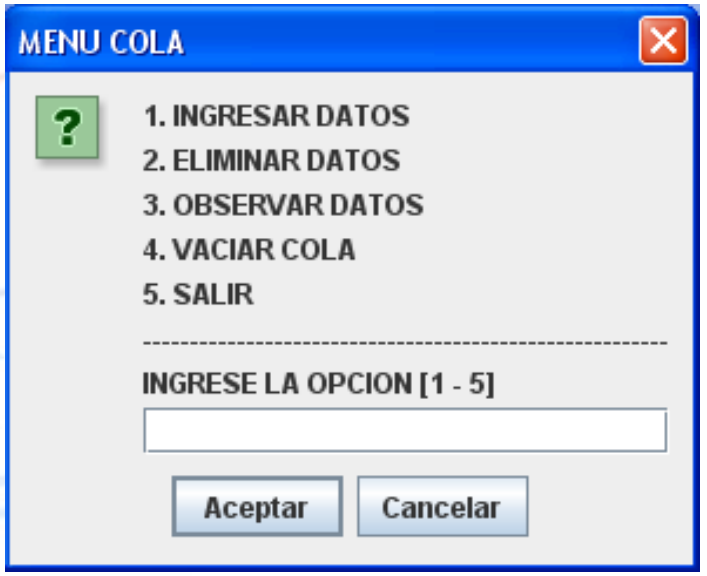
Los diversos métodos de búsqueda de Vector devuelven la posición de la primera ocurrencia del objeto buscado, o bien verdadero-falso según el resultado de la búsqueda.

- `boolean contains(Object op); //devuelve true si se encuentra` ***`op`***`.`
- `int indexOf(Object op);//devuelve la primera posición de` ***`op`***`, -1 si no existe.`

Ejemplificación de Pila y Cola, ArrayList y ArrayObject.

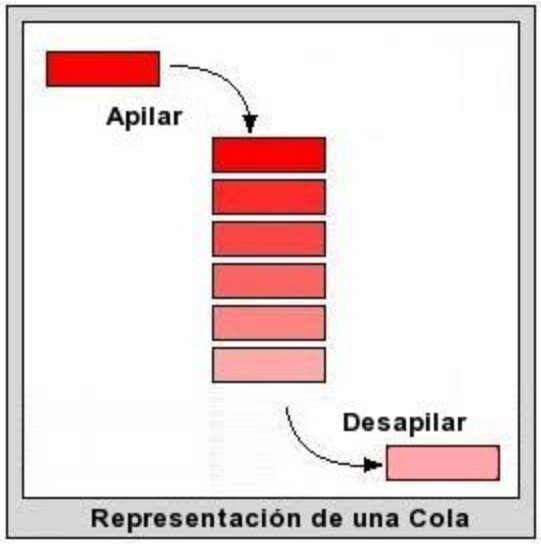
- Pila.
- Cola.





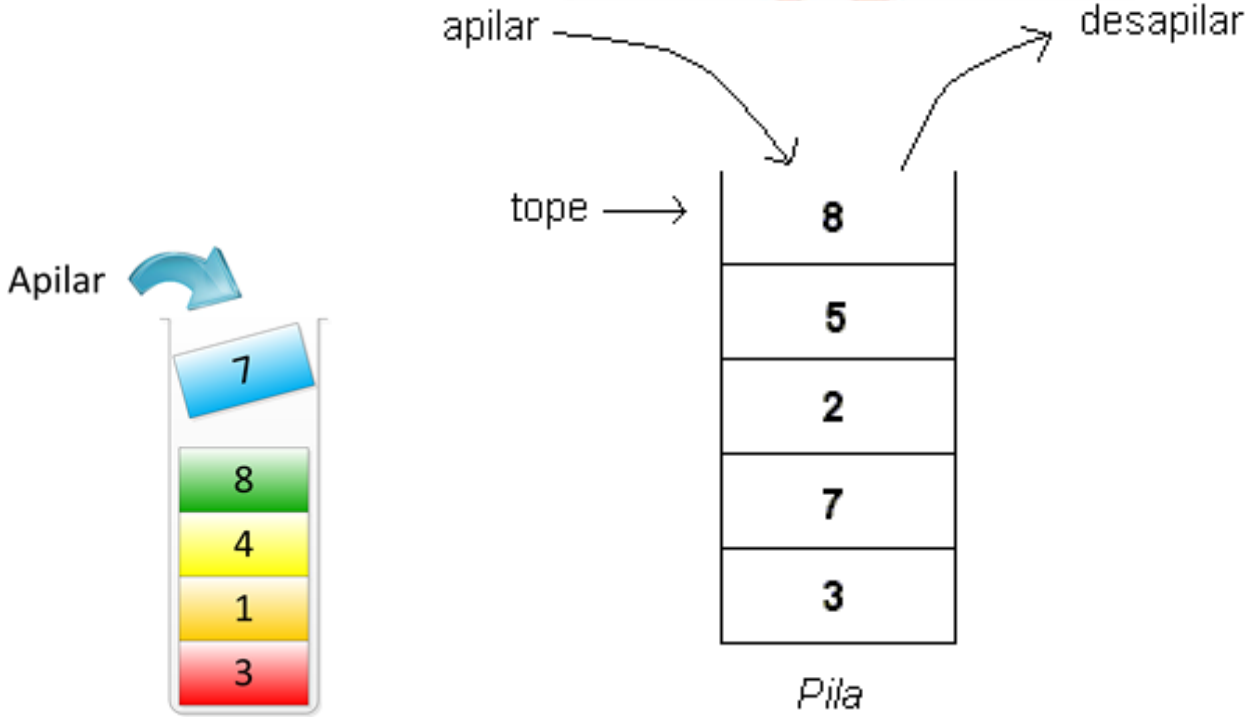
Uso de ArrayList.

COLAS(FIFO)



Uso de ArrayList.

PILAS(LIFO)



Bibliografía:

- Joyanes A. L., Zahonero M. I., "Programación en Java 6, Algoritmos, programación orientada a objetos e interfaz gráfica de usuario", 2011, 1ra edición, Ed:McGrawHill.

- Joyanes Aguilar L. Programación orientada a objetos, 1ra edición, Ed:McGrawHill.

- Roger S. Pressman, Ingeniería del Software, un enfoque practico. 5ta edición, Ed: McGrawHill.

- Ian Sommerville, Ingeniería del Software. 7ma edición, Ed: Pearson.