

CIFP César Manrique.

Programación 1º de Desarrollo de Aplicaciones Web

Profesor: José David Díaz Díaz

Actividades de la Unidad 8: Herencia

Juan Carlos Francisco Mesa



Esta obra está licenciada bajo la Licencia Creative Commons Atribución 4.0 Internacional.
Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by/4.0/> o
envíe una carta a Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Índice

Actividades.....	1
Desarrollo.....	2
Actividades de comprobación.....	2
8.1. Sobre una subclase es correcto afirmar que:.....	2
8.2. En relación con las clases abstractas es correcto señalar que:.....	2
8.3. ¿En qué consiste la sustitución u overriding?.....	2
8.4. Sobre la clase Object es cierto indicar que:.....	3
8.5. ¿Cuál de las siguientes afirmaciones sobre el método equals() es correcta?.....	3
8.6. ¿Cuál de las siguientes afirmaciones sobre el método toString() es correcta?.....	3
8.7. ¿Cuál de las siguientes afirmaciones sobre el método getClass() es correcta?.....	4
8.8. Una clase puede heredar:.....	4
8.9. La selección dinámica de métodos:.....	4
8.10. ¿Cuál de las siguientes afirmaciones sobre el método super() es correcta?.....	5
Actividades de Aplicación.....	6
8.11. Crea la clase Campana que hereda de Instrumento (definida en la Actividad resuelta 8.4).....	6
8.12. Las empresas de transporte, para evitar daños en los paquetes, embalan todas sus mercancías en cajas con el tamaño adecuado.....	7
8.14. Reimplementa la clase Lista de la Actividad resuelta 7.11, sustituyendo el método mostrar () por el método toString ().....	9
8.15. Escribe en la clase Lista un método equals() para compararlas. Dos listas se considerarán iguales si tienen los mismos elementos (incluidas las repeticiones) en el mismo orden.....	11

8.16. Diseña la clase Pila heredando de Lista (ver Actividad resuelta 7.13).....	13
8.17. Escribe la clase Cola heredando de Lista (ver Actividad final 7.18).....	15

Actividades

Actividades de la Unidad 8: Herencia.

En este documento se detallan las actividades a realizar. Se entregará al profesor en la plataforma digital dos ficheros. Un primer fichero pdf con todas las actividades a realizar, el nombre del fichero será “unidad2 + nombre del alumno.pdf”. Añadir en el fichero pdf por cada actividad de programación dos capturas de pantalla, una del código y otra de su ejecución. También en el fichero pdf copiar todas las preguntas y las respuestas correctas de las actividades de comprobación. Además, entregar un segundo fichero comprimido con todos los códigos fuentes de cada actividad de programación realizada.

Todas las actividades resueltas se deberán de analizar y no se entregarán.

A continuación, detallamos las actividades a realizar:

- **Actividades propuestas.** Esta unidad no tiene dichas actividades.
- **Actividades de comprobación.** Realizarlas todas. Copiar todas las preguntas y sus respuestas correctas.
- **Actividades de aplicación.** Realizar las siguientes **8.11, 8.12, 8.14, 8.15, 8.16, 8.17.**
- **Actividades de ampliación.** No realizar ninguna.

[Volver al índice](#)

Desarrollo

Actividades de comprobación.

8.1. Sobre una subclase es correcto afirmar que:

- a) Tiene menos atributos que su superclase.
- b) Tiene menos miembros que su superclase.
- c) Hereda los miembros no privados de su superclase.
- d) Hereda todos los miembros de su superclase.

[Volver al índice](#)

8.2. En relación con las clases abstractas es correcto señalar que:

- a) Implementan todos sus métodos.
- b) No implementan ningún método.
- c) No tienen atributos.
- d) Tienen algún método abstracto.

[Volver al índice](#)

8.3. ¿En qué consiste la sustitución u overriding?

- a) En sustituir un método heredado por otro implementado en la propia clase.
- b) En sustituir un atributo por otro del mismo nombre.
- c) En sustituir una clase por una subclase.
- d) En sustituir un valor de una variable por otro.

[Volver al índice](#)

8.4. Sobre la clase Object es cierto indicar que:

- a) Es abstracta.
- b) Hereda de todas las demás.
- c) Tiene todos sus métodos abstractos.
- d) Es superclase de todas las demás clases.

[Volver al índice](#)

8.5. ¿Cuál de las siguientes afirmaciones sobre el método equals() es correcta?

- a) Hay que implementarlo, ya que es abstracto.
- b) Sirve para comparar solo objetos de la clase Object.
- c) Se hereda de Object, pero debemos reimplementarlo al definirlo en una clase.
- d) No hay que implementarlo, ya que se hereda de Object.

[Volver al índice](#)

8.6. ¿Cuál de las siguientes afirmaciones sobre el método toString() es correcta?

- a) Sirve para mostrar la información que nos interesa de un objeto.
- b) Convierte automáticamente un objeto en una cadena.
- c) Encadena varios objetos.
- d) Es un método abstracto de Object que tenemos que implementar.

[Volver al índice](#)

8.7. ¿Cuál de las siguientes afirmaciones sobre el método getClass() es correcta?

- a) Convierte los objetos en clases.
- b) Obtiene la clase a la que pertenece un objeto.**
- c) Obtiene la superclase de una clase.
- d) Obtiene una clase a partir de su nombre.

[Volver al índice](#)

8.8. Una clase puede heredar:

- a) De una clase.**
- b) De dos clases.
- c) De todas las clases que queramos.
- d) Solo de la clase Object .

[Volver al índice](#)

8.9. La selección dinámica de métodos:

- a) Se produce cuando una variable cambia de valor durante la ejecución de un programa.
- b) Es el cambio de tipo de una variable en tiempo de ejecución.
- c) Es la asignación de un mismo objeto a más de una variable en tiempo de ejecución.
- d) Es la ejecución de distintas implementaciones de un mismo método, asignando objetos de distintas clases a una misma variable, en tiempo de ejecución.**

[Volver al índice](#)

8.10. ¿Cuál de las siguientes afirmaciones sobre el método super() es correcta?

- a) Sirve para llamar al constructor de la superclase.
- b) Sirve para invocar un método escrito más arriba en el código.
- c) Sirve para llamar a cualquier método de la superclase.
- d) Sirve para hacer referencia a un atributo de la superclase .

[Volver al índice](#)

Actividades de Aplicación.

8.11. Crea la clase Campana que hereda de Instrumento (definida en la Actividad resuelta 8.4).

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
3  * to change this license
4  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
5  * this template
6  */
7
8  package Main;
9
10 /**
11  * @author juancfm
12  */
13
14 public class Main {
15
16     /**
17     * 8.11. Crea la clase Campana que hereda de Instrumento (definida en la
18     * Actividad resuelta 8.4).
19     */
20     public static void main(String[] args) {
21
22         Nota cancion[] = {Nota.DO, Nota.SI, Nota.SOL, Nota.RE, Nota.FA};
23         Campana p = new Campana();
24         for (Nota nota : cancion) {
25             p.add(nota);
26         }
27         p.interpretar();
28     }
29 }
30
```

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
3  * to change this license
4  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
5  * this template
6  */
7
8  package Main;
9
10 import java.util.Arrays;
11
12 /**
13  * @author juancfm
14  */
15
16 public abstract class Instrumento {
17
18     protected Nota[] melodia; //tabla que almacena las notas a interpretar
19
20     public Instrumento() {
21         melodia = new Nota[0]; //creamos la tabla
22     }
23
24     //Usa el algoritmo de inserción no ordenada
25
26     void add(Nota n) {
27         //redimensionamos
28         melodia = Arrays.copyOf(original:melodia, melodia.length + 1);
29         melodia[melodia.length - 1] = n; //insertamos el nuevo elemento al final
30     }
31
32     abstract void interpretar(); //a implementar en cada subclase
33 }
34
```

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
3  * to change this license
4  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
5  * this template
6  */
7
8  package Main;
9
10 /**
11  * @author juancfm
12  */
13
14 //Enumerado con las nota musicales
15 public enum Nota {
16     DO, RE, MI, FA, SOL, LA, SI
17 }
18
```

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
3  * to change this license
4  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
5  * this template
6  */
7
8  package Main;
9
10 /**
11  * @author juancfm
12  */
13
14 public class Campana extends Instrumento {
15
16     @Override
17     public void interpretar() {
18         System.out.println("La melodía es:\n");
19         for (Nota nota : melodia) {
20             // Se imprimen las notas de la melodía cambiando de mayúsculas
21             // a minúsculas para señalar un timbre diferente
22             System.out.println(":" + nota.toString().toLowerCase());
23         }
24         System.out.println("\nGracias");
25     }
26 }
27
```

```
Output - Main (run) x
run:
La melodía es:
do
si
sol
re
fa

Gracias
BUILD SUCCESSFUL (total time: 0 seconds)
```

[Volver al índice](#)

8.12. Las empresas de transporte, para evitar daños en los paquetes, embalan todas sus mercancías en cajas con el tamaño adecuado.

Una caja se crea expresamente con un ancho, un alto y un fondo y, una vez creada, se mantiene inmutable. Cada caja lleva pegada una etiqueta, de un máximo de 30 caracteres, con información útil como el nombre del destinatario, dirección, etc. Implementa la clase Caja con los siguientes métodos:

Caja (int ancho, int alto, int fondo, Unidad unidad): que construye una caja con las dimensiones especificadas, que pueden encontrarse en «cm (centímetros)» o «m» (metros).

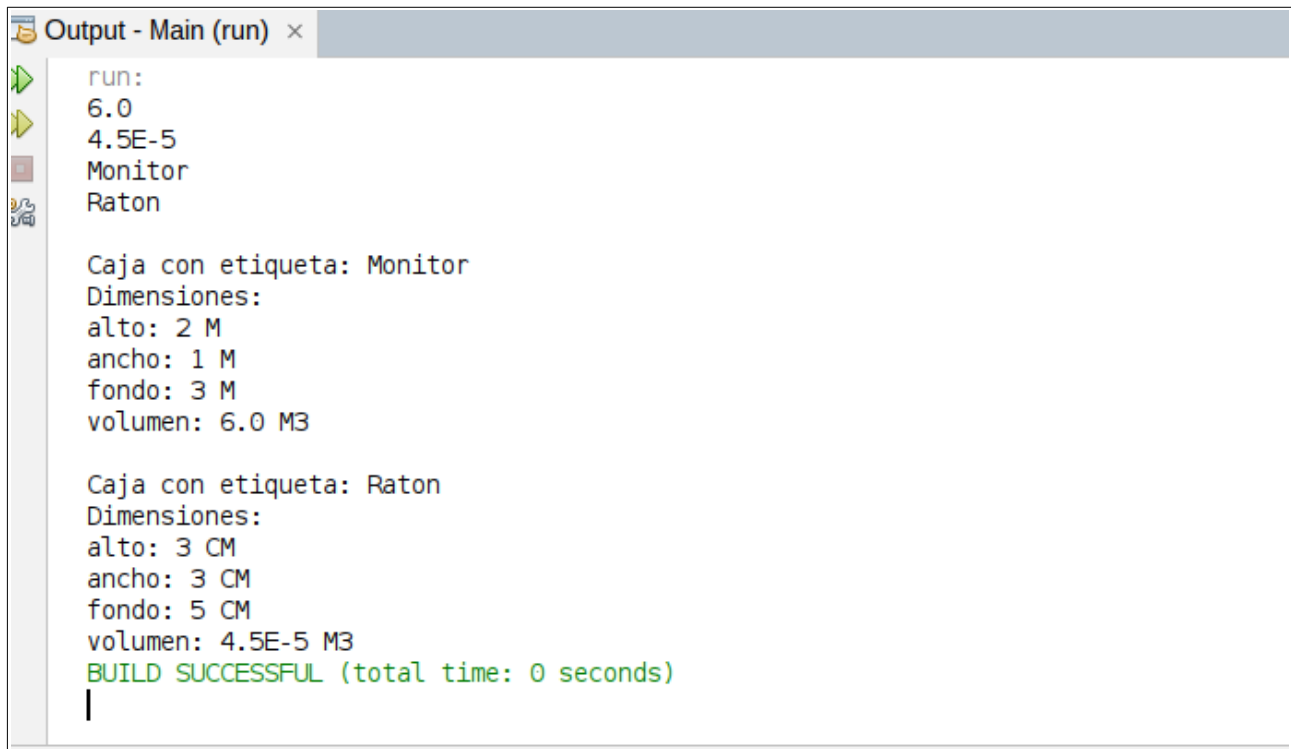
double getVolumen (): que devuelve el volumen de la caja en metros cúbicos.

void setEtiqueta (String etiqueta): que modifica el valor de la etiqueta de la caja.

String toString (): que devuelve una cadena con la representación de la caja.

```
1 package Main;
2
3 /**
4  *
5  * @author juancfm
6  */
7 public class Main {
8
9     /**
10      * @param args the command line arguments
11      */
12     public static void main(String[] args) {
13
14         /**
15          * 8.12. Las empresas de transporte, para evitar daños en los paquetes,
16          * embalan todas sus mercancías en cajas con el tamaño adecuado.
17          * Una caja se crea expresamente con un ancho, un alto y un fondo y,
18          * una vez creada, se mantiene inmutable. Cada caja lleva pegada una
19          * etiqueta, de un máximo de 30 caracteres, con información útil como
20          * el nombre del destinatario, dirección, etc. Implementa la clase Caja
21          * con los siguientes métodos:
22          * Caja (int ancho, int alto, int fondo, Unidad unidad): que construye
23          * una caja con las dimensiones especificadas, que pueden encontrarse
24          * en «cm (centímetros)» o «m» (metros).
25          * double getVolumen (): que devuelve el volumen de la caja en metros
26          * cúbicos.
27          * void setEtiqueta (String etiqueta): que modifica el valor de la
28          * etiqueta de la caja.
29          * String toString (): que devuelve una cadena con la representación
30          * de la caja.
31          */
32
33         Unidad Unidades = Unidad.M;
34         Caja cajaGrande = new Caja( ancho: 1, alto: 2, fondo: 3, unidad: Unidades);
35
36         Unidades = Unidad.CM;
37         Caja cajaPequeña = new Caja( ancho: 3, alto: 3, fondo: 5, unidad: Unidades);
38
39         System.out.println((double)cajaGrande.getVolumen());
40
41         System.out.println((double)cajaPequeña.getVolumen());
42
43         // Consultar si se desea definir etiqueta como privada o no
44         cajaGrande.setEtiqueta( etiqueta: "Monitor");
45         cajaPequeña.setEtiqueta( "Raton");
46
47         System.out.println( x: cajaGrande.etiqueta);
48         System.out.println( x: cajaPequeña.etiqueta);
49
50         System.out.println( x: cajaGrande);
51         System.out.println( x: cajaPequeña);
52     }
53 }
54
```

```
1 package Main;
2
3 /**
4  *
5  * @author David
6  */
7 public enum Unidad {
8     CM, M
9 }
10
11
12 package Main;
13
14 /**
15  *
16  * @author juancfm
17  */
18 public class Caja {
19
20     int ancho;
21     int alto;
22     int fondo;
23     Unidad unidad;
24     String etiqueta;
25
26     public Caja(int ancho, int alto, int fondo, Unidad unidad){
27         this.ancho = ancho;
28         this.alto = alto;
29         this.fondo = fondo;
30         this.unidad = unidad;
31         this.etiqueta = "Etiqueta por defecto";
32     }
33
34     public double getVolumen(){
35         double result = this.alto * this.ancho * this.fondo;
36
37         if (this.unidad.toString().equals( anObject: "CM")) {
38             result /= 1000000;
39         }
40
41         return result;
42     }
43
44     public void setEtiqueta(String etiqueta){
45         this.etiqueta = etiqueta;
46     }
47
48     // Se sobrescribe el método toString() de la clase Object
49     @Override
50     public String toString(){
51         String result;
52         result = "\nCaja con etiqueta: " + this.etiqueta +
53             "\nDimensiones: " +
54             "\nalto: " + this.alto + " " + this.unidad +
55             "\nancho: " + this.ancho + " " + this.unidad +
56             "\nfondo: " + this.fondo + " " + this.unidad +
57             "\nvolumen: " + this.getVolumen() + " M3";
58         return result;
59     }
60 }
61
```



```
run:
6.0
4.5E-5
Monitor
Raton

Caja con etiqueta: Monitor
Dimensiones:
alto: 2 M
ancho: 1 M
fondo: 3 M
volumen: 6.0 M3

Caja con etiqueta: Raton
Dimensiones:
alto: 3 CM
ancho: 3 CM
fondo: 5 CM
volumen: 4.5E-5 M3
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

[Volver al índice](#)

8.14. Reimplementa la clase Lista de la Actividad resuelta 7.11, sustituyendo el método mostrar () por el método toString ().

```
1 package Main;
2
3 /**
4  *
5  * @author juancfm
6  */
7 public class Main {
8
9     /**
10      * @param args the command line arguments
11      */
12     public static void main(String[] args) {
13         /**
14          * 8.14. Reimplementa la clase Lista de la Actividad resuelta 7.11,
15          * sustituyendo el método mostrar () por el método toString ().
16          */
17
18         Lista l1 = new Lista();
19         for (int i = 0; i < 3; i++) {
20             l1.insertarFinal(nuevo: i);
21         }
22         System.out.println("l1: "+l1);
23         Lista l2 = new Lista();
24         for (int i = 0; i < 3; i++) {
25             l2.insertarFinal(nuevo: i);
26         }
27         System.out.println("l2: "+l2);
28     }
29 }
30
31
```

```
1 package Main;
2
3 import java.util.Arrays;
4
5 /**
6  *
7  * @author juancfm
8  */
9 public class Lista {
10     Integer[] tabla;
11
12     public Lista() {
13         tabla = new Integer[0];
14     }
15
16     void insertarPrincipio(Integer nuevo) {
17         tabla = Arrays.copyOf(
18             original: tabla,
19             tabla.length + 1
20         );
21         System.arraycopy(
22             src: tabla,
23             srcPos: 0,
24             dest: tabla,
25             destPos: 1,
26             tabla.length - 1
27         );
28         tabla[0] = nuevo;
29     }
30
31     void insertarFinal(Integer nuevo) {
32         tabla = Arrays.copyOf(
33             original: tabla,
34             tabla.length + 1
35         );
36         tabla[tabla.length - 1] = nuevo;
37     }
38
39     void insertarFinal(Lista otraLista) {
40         int tamIni = tabla.length;
41         tabla = Arrays.copyOf(
42             original: tabla,
43             tabla.length + otraLista.tabla.length
44         );
45         System.arraycopy(
46             src: otraLista.tabla,
47             srcPos: 0,
48             dest: tabla,
49             destPos: tamIni,
50             length: otraLista.tabla.length
51         );
52     }
53
54     void insertar(int posicion, Integer nuevo) {
55         tabla = Arrays.copyOf(
56             original: tabla,
57             tabla.length + 1
58         );
59         System.arraycopy(
60             src: tabla,
61             srcPos: posicion,
62             dest: tabla,
63             posicion + 1,
64             tabla.length - posicion - 1
65         );
66         tabla[posicion] = nuevo;
67     }
68 }
69
```

Actividades de la Unidad 8: Herencia

```
70 Integer eliminar(int indice) {
71     Integer eliminado = null;
72     if (indice >= 0 && indice < tabla.length) {
73         eliminado = tabla[indice];
74         for (int i = indice + 1; i < tabla.length; i++) {
75             tabla[i - 1] = tabla[i];
76         }
77         tabla = Arrays.copyOf(tabla, tabla.length - 1);
78     }
79     return eliminado;
80 }
81
82 Integer get(int indice) {
83     Integer resultado = null;
84     if (indice >= 0 && indice < tabla.length) {
85         resultado = tabla[indice];
86     }
87     return resultado;
88 }
89
90 int buscar(Integer claveBusqueda) {
91     int indice = -1;
92     for (int i = 0; i < tabla.length && indice == -1; i++) {
93         if (tabla[i].equals(claveBusqueda)) {
94             indice = i;
95         } else {
96             continue;
97         }
98     }
99     return indice;
100 }
101
102 public int numeroElementos() {
103     return tabla.length;
104 }
105
106 @Override
107 public String toString() {
108     String result;
109     result = "Lista: " + Arrays.toString(tabla);
110     return result;
111 }
```

```
Output - Main (run) x
run:
l1: Lista: [0, 1, 2]
l2: Lista: [0, 1, 2]
BUILD SUCCESSFUL (total time: 0 seconds)
```

[Volver al índice](#)

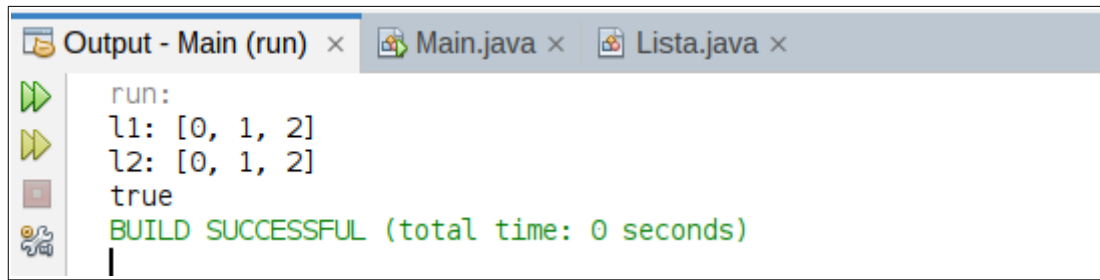
8.15. Escribe en la clase Lista un método equals() para compararlas. Dos listas se considerarán iguales si tienen los mismos elementos (incluidas las repeticiones) en el mismo orden.

```
1 package Main;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Lista l1 = new Lista();
7         for (int i = 0; i < 3; i++) {
8             l1.insertarFinal( nuevo:i);
9         }
10        System.out.println("l1: "+l1);
11        Lista l2 = new Lista();
12        for (int i = 0; i < 3; i++) {
13            l2.insertarFinal( nuevo:i);
14        }
15        System.out.println("l2: "+l2);
16        System.out.println( x:l1.equals( tabla:l2));
17    }
18 }
19
20 }
```

```
1 package Main;
2
3 import java.util.Arrays;
4
5 /**
6  * @author juancfma
7  */
8 public class Lista {
9     Integer[] tabla;
10
11     public Lista() {
12         tabla = new Integer[0];
13     }
14
15     void insertarPrincipio(Integer nuevo) {
16         tabla = Arrays.copyOf(
17             original:tabla,
18             tabla.length + 1
19         );
20         System.arraycopy(
21             src:tabla,
22             srcPos:0,
23             dest:tabla,
24             destPos:1,
25             tabla.length - 1
26         );
27         tabla[0] = nuevo;
28     }
29
30     void insertarFinal(Integer nuevo) {
31         tabla = Arrays.copyOf(
32             original:tabla,
33             tabla.length + 1
34         );
35         tabla[tabla.length - 1] = nuevo;
36     }
37
38     void insertarFinal(Lista otraLista) {
39         int tamIni = tabla.length;
40         tabla = Arrays.copyOf(
41             original:tabla,
42             tabla.length + otraLista.tabla.length
43         );
44         System.arraycopy(
45             src:otraLista.tabla,
46             srcPos:0,
47             dest:tabla,
48             tamIni,
49             otraLista.tabla.length
50         );
51     }
52
53 }
```

```
54 void insertar(int posicion, Integer nuevo) {
55     tabla = Arrays.copyOf(
56         original:tabla,
57         tabla.length + 1
58     );
59     System.arraycopy(
60         src:tabla,
61         srcPos:posicion,
62         dest:tabla,
63         posicion + 1,
64         tabla.length - posicion - 1
65     );
66     tabla[posicion] = nuevo;
67 }
68
69 Integer eliminar(int indice) {
70     Integer eliminado = null;
71     if (indice >= 0 && indice < tabla.length) {
72         eliminado = tabla[indice];
73         for (int i = indice + 1; i < tabla.length; i++) {
74             tabla[i - 1] = tabla[i];
75         }
76         tabla = Arrays.copyOf( original:tabla, tabla.length - 1);
77     }
78     return eliminado;
79 }
80
81 Integer get(int indice) {
82     Integer resultado = null;
83     if (indice >= 0 && indice < tabla.length) {
84         resultado = tabla[indice];
85     }
86     return resultado;
87 }
88
89 int buscar(Integer claveBusqueda) {
90     int indice = -1;
91     for (int i = 0; i < tabla.length && indice == -1; i++) {
92         if (tabla[i].equals( obj:claveBusqueda)) {
93             indice = i;
94         } else {
95             continue;
96         }
97     }
98     return indice;
99 }
100
101 public int numeroElementos() {
102     return tabla.length;
103 }
104
105 @Override
106 public String toString() {
107     String result;
108     result = Arrays.toString( =tabla);
109     return result;
110 }
111
112 @Override
113 public boolean equals(Object tabla){
114     boolean status;
115
116     status = (
117         this.getClass().getSimpleName().equals(
118             withObject:tabla.getClass().getSimpleName())
119         && this.toString().equals( withObject:tabla.toString())
120     );
121     return status;
122 }
123 }
```

Actividades de la Unidad 8: Herencia



The screenshot shows an IDE window with three tabs: 'Output - Main (run)', 'Main.java', and 'Lista.java'. The 'Output - Main (run)' tab is active and displays the following text:

```
run:
l1: [0, 1, 2]
l2: [0, 1, 2]
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

On the left side of the output window, there are four icons: a green double arrow, a yellow double arrow, a red square, and a gear icon.

[Volver al índice](#)

8.16. Diseña la clase Pila heredando de Lista (ver Actividad resuelta 7.13).

```

1 package Main;
2
3 /**
4  *
5  * @author juancfr
6  */
7 public class Main {
8
9     public static void main(String[] args) {
10
11         /**
12          * 8.16. Diseña la clase Pila heredando de Lista (ver Actividad
13          * resuelta 7.13).
14          */
15
16         Pila p = new Pila();
17         for (int i = 0; i < 10; i++) {
18             p.apilar( elemento:i);
19         }
20
21         System.out.println( x:p);
22
23         Integer n = p.desapilar();
24         while (n != null) {
25             System.out.println( x:n);
26             n = p.desapilar();
27         }
28     }
29 }

```

```

1 package Main;
2
3 public class Pila extends Lista {
4
5     void apilar(Integer elemento) {
6         this.insertarFinal( nuevo:elemento);
7     }
8
9     Integer desapilar() {
10         return this.eliminar(this.tabla.length - 1);
11     }
12
13     @Override
14     public void mostrar() {
15         this.mostrar();
16     }
17 }

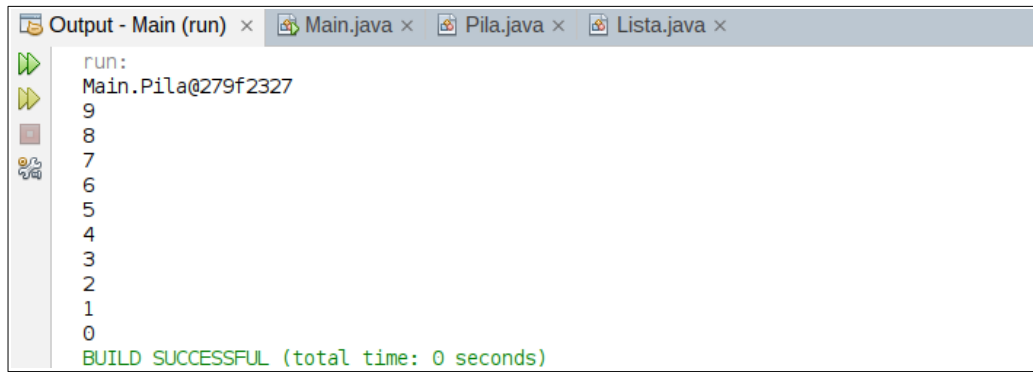
```

```

1 package Main;
2
3 import java.util.Arrays;
4
5 public class Lista {
6     Integer[] tabla;
7
8     public Lista() {
9         tabla = new Integer[0];
10    }
11
12    void insertarPrincipio(Integer nuevo) {
13        tabla = Arrays.copyOf( tabla, tabla.length + 1);
14        System.arraycopy( nuevo: nuevo, 0, tabla, 0, tabla.length - 1);
15        tabla[0] = nuevo;
16    }
17
18    void insertarFinal(Integer nuevo) {
19        tabla = Arrays.copyOf( tabla, tabla.length + 1);
20        tabla[tabla.length - 1] = nuevo;
21    }
22
23    void insertarFinal(Lista otraLista) {
24        int tamIni = tabla.length; //tamaño inicial tabla
25        tabla = Arrays.copyOf( tabla, tabla.length + otraLista.tabla.length);
26        System.arraycopy( otraLista.tabla, 0, tabla, tamIni, otraLista.tabla.length);
27    }
28
29    void insertar(int posicion, Integer nuevo) {
30        tabla = Arrays.copyOf( tabla, tabla.length + 1);
31        System.arraycopy( nuevo: nuevo, 0, tabla, posicion, tabla.length - 1);
32        tabla[posicion] = nuevo;
33    }
34
35    Integer eliminar(int indice) {
36        Integer eliminado = null;
37        if (indice >= 0 && indice < tabla.length) {
38            eliminado = tabla[indice];
39            for (int i = indice + 1; i < tabla.length; i++) {
40                tabla[i - 1] = tabla[i];
41            }
42            tabla = Arrays.copyOf( tabla, tabla.length - 1);
43        }
44        return eliminado;
45    }
46
47    Integer get(int indice) {
48        Integer resultado = null;
49        if (indice >= 0 && indice < tabla.length) {
50            resultado = tabla[indice];
51        }
52        return resultado;
53    }
54
55    int buscar(Integer claveBusqueda) {
56        int indice = -1;
57        for (int i = 0; i < tabla.length && indice == -1; i++) {
58            if (tabla[i].equals( claveBusqueda)) {
59                indice = i;
60            }
61        }
62        return indice;
63    }
64
65    void ordenar() {
66        Arrays.sort( tabla);
67    }
68
69    public void mostrar() {
70        System.out.println( Arrays.toString( tabla));
71    }
72
73 }

```

Actividades de la Unidad 8: Herencia



```
run:
Main.Pila@279f2327
9
8
7
6
5
4
3
2
1
0
BUILD SUCCESSFUL (total time: 0 seconds)
```

[Volver al índice](#)

8.17. Escribe la clase Cola heredando de Lista (ver Actividad final 7.18).

```
1 package Main;
2
3 /**
4  *
5  * @author juancfm
6  */
7 public class Main {
8
9     public static void main(String[] args) {
10
11         /**
12          * 8.17. Escribe la clase Cola heredando de Lista (ver Actividad final 7.18).
13          */
14
15         Cola c = new Cola();
16         for (int i = 0; i < 10; i++) {
17             c.encolar(i);
18         }
19
20         System.out.println("x:c");
21
22         Integer n = c.desencolar();
23
24         while (n != null) {
25             System.out.println("x:n");
26             n = c.desencolar();
27         }
28     }
29 }
30
31 }
```

```
1 package Main;
2
3 /**
4  *
5  * @author juancfm
6  */
7 public class Cola extends Lista {
8
9     public void encolar(int i) {
10         this.insertarFinal(nuevo:i);
11     }
12
13     public Integer desencolar() {
14         return this.eliminar(indice:0);
15     }
16
17 }
18 }
```

```
1 package Main;
2
3 import java.util.Arrays;
4
5 public class Lista {
6     Integer[] tabla;
7
8     public Lista() {
9         tabla = new Integer[0];
10     }
11
12     void insertarPrincipia(Integer nuevo) {
13         tabla = Arrays.copyOf(tabla, tabla.length + 1);
14         System.arraycopy(tabla, 0, tabla, 1, tabla.length - 1);
15         tabla[0] = nuevo;
16     }
17
18     void insertarFinal(Integer nuevo) {
19         tabla = Arrays.copyOf(tabla, tabla.length + 1);
20         tabla[tabla.length - 1] = nuevo;
21     }
22
23     void insertarFinal(Lista otraLista) {
24         int tamIni = tabla.length; //tamaño inicial tabla
25         tabla = Arrays.copyOf(tabla, tabla.length + otraLista.tabla.length);
26         System.arraycopy(otraLista.tabla, 0, tabla, tamIni, otraLista.tabla.length);
27     }
28
29     void insertar(int posicion, Integer nuevo) {
30         tabla = Arrays.copyOf(tabla, tabla.length + 1);
31         System.arraycopy(tabla, posicion, tabla, posicion + 1, tabla.length - posicion - 1);
32         tabla[posicion] = nuevo;
33     }
34
35     Integer eliminar(int indice) {
36         Integer eliminado = null;
37         if (indice >= 0 && indice < tabla.length) {
38             for (int i = indice + 1; i < tabla.length; i++) {
39                 tabla[i - 1] = tabla[i];
40             }
41             tabla = Arrays.copyOf(tabla, tabla.length - 1);
42         }
43         return eliminado;
44     }
45
46     Integer get(int indice) {
47         Integer resultado = null;
48         if (indice >= 0 && indice < tabla.length) {
49             resultado = tabla[indice];
50         }
51         return resultado;
52     }
53
54     int buscar(Integer claveBusqueda) {
55         int indice = -1;
56         for (int i = 0; i < tabla.length && indice == -1; i++) {
57             if (tabla[i].equals(claveBusqueda)) {
58                 indice = i;
59             }
60         }
61         return indice;
62     }
63
64     void ordenar() {
65         Arrays.sort(tabla);
66     }
67
68     public void mostrar() {
69         System.out.println("Arrays.toString(" + Arrays.toString(tabla) + ")");
70     }
71
72 }
73
74 }
```

```
Debugger Console x Main (run) x
run:
Main.Cola@279f2327
0
1
2
3
4
5
6
7
8
9
BUILD SUCCESSFUL (total time: 0 seconds)
```

[Volver al índice](#)