

Taller de Lenguajes II

Práctica nº 4

1. **Cadena de Constructores.** Analice el siguiente código y responda

```
// archivo CadenaDeConstructores.java
class CadenaDeConstructores {
    public static void main(String[] args) {
        Hijo h = new Hijo();
    }
}

// archivo Hijo.java
class Hijo extends Padre {
    Hijo() {
        System.out.println("Constructor Hijo()");
    }
}

// archivo Padre.java
class Padre extends Abuelo{
    Padre(int x) {
        System.out.println("Constructor Padre(" + x + ") ");
    }
}

// archivo Abuelo.java
class Abuelo{
    Abuelo() {
        System.out.println("Constructor Abuelo()");
    }
}
```

- Verifique que compila. En caso de aparecer errores corrijalos de modo que compile exitosamente.
 - ¿Qué imprime la ejecución de la clase CadenaDeConstructores?
 - ¿Dónde se encuentran estas llamadas sucesivas que forman la cadena de constructores?
2. **Patrón Singleton.** Hay un número de casos donde se necesita asegurar que **NUNCA exista más de una instancia de una determinada clase** en la aplicación. A modo de ejemplo, cuando un sistema arranca su ejecución, los parámetros generales de configuración podrían levantarse en una **ÚNICA** instancia de la clase. Aquí aplica lo que se conoce como el **patrón Singleton**.
- Implemente una clase llamada **CharlyGarcia** que cumpla con el patrón Singleton. La clase **CharlyGarcia** debe proveer una manera para acceder a esa única instancia.
 - Escriba la clase **CharlyGarcia** (piense en los modificadores de acceso del constructor y en los calificadores java que tiene disponibles, para escribir la solución).
 - Agregue el siguiente método de instancia
public void cantar(){

```
        System.out.println("Charly Garcia está cantando");
    }
}
```

- c. Cree una clase TestCharly donde demuestre el uso de este Singleton.
- d. Realice el Diagrama UML de su solución.

3. **Uso de literales String, clases String, StringBuffer y StringBuilder.** En caso de ser necesario lea la siguiente información relacionada al manejo de literales String y String:

- <http://java67.blogspot.com.ar/2014/08/difference-between-string-literal-and-new-String-object-Java.html>

- a. Cree un paquete llamado `unlp.info.comparacionstring`

Escriba el siguiente código y ejecútelo.

```
package unlp.info.comparacionstring;
public class StringDemo {
    public static void main(String[] args) {
        String str1="Leones y Tigres y Osos!";
        String str2="Leones y Tigres y Osos!";
        String str3=str2;
        String str4=new String("Leones y Tigres y Osos!");
        String str5=" Y yo!";
        String str6="Leones y Tigres y Osos! Y yo!";
        String str7= str1 + str5;
        System.out.println(str1==str2);
        System.out.println(str1==str3);
        System.out.println(str1==str4);
        System.out.println(str2==str3);
        System.out.println(str2==str4);
        System.out.println(str3==str4);
        System.out.println(str6==str7);
        System.out.println(str1.equals(str4));
        System.out.println(str6.equals(str7));
    }
}
```

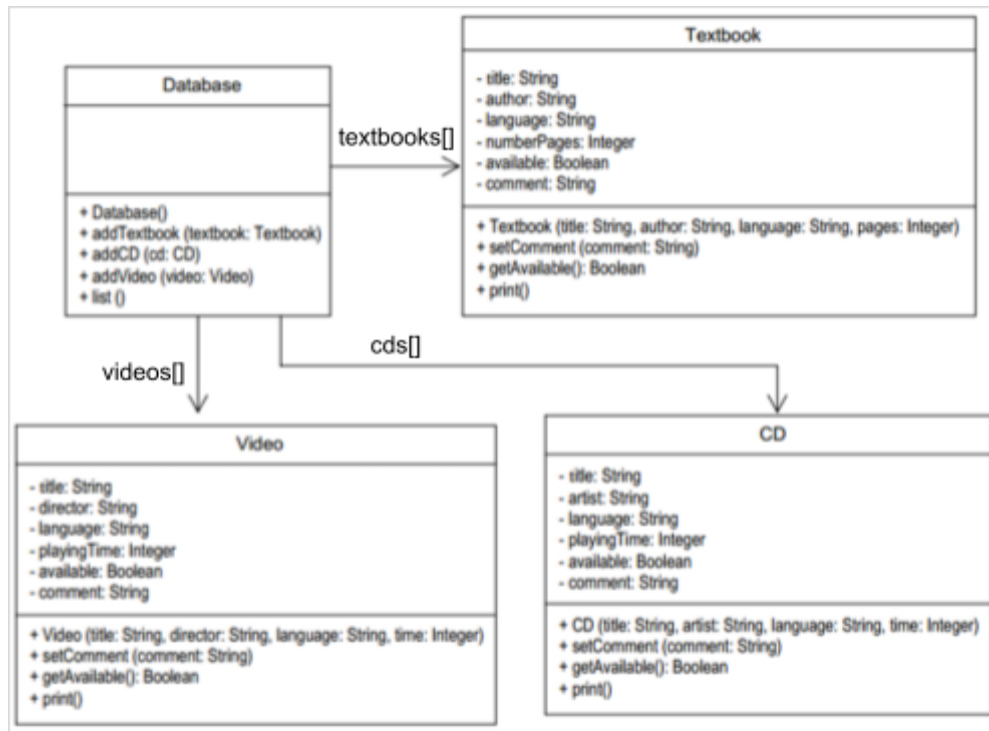
Sentencia	true/false	¿Por qué? - JUSTIFIQUE
<code>str1 == str2</code>		
<code>str1 == str3</code>		
<code>str1 == str4</code>		
<code>str2 == str3</code>		
<code>str2 == str4</code>		
<code>str3 == str4</code>		
<code>str6 == str7</code>		
<code>str1.equals(str4)</code>		
<code>str6.equals(str7)</code>		

- b. ¿Qué hace el método `equals` de la clase `String`? (puede observar la implementación adjuntando los archivos fuente `src.zip` del JDK1.8+)
- c. Suponga que cuenta con una clase `Persona` que modela a las personas del mundo real.

- i. ¿Considera que sería interesante hacer un “override” (sobreescritura) del método equals? Si la respuesta es afirmativa, indique el criterio de comparación, caso contrario **JUSTIFIQUE**.
 - ii. En caso de no sobreescribir el método equals, ¿cuál es el criterio por default en Java para comparar dos (2) personas?
 - d. Descargue del sitio de la cátedra el archivo TestString.java.
 - i. Cree un proyecto en eclipse e importe TestString.java
 - ii. Ejecute la clase TestString
 1. Indique los resultados obtenidos
 2. **JUSTIFIQUE** los resultados obtenidos. Para justificarlos puede revisar la teoría, verificar cómo están implementadas esas operaciones ó acceder a alguna de las siguientes URLs:
 - <http://stackoverflow.com/questions/2971315/string-stringbuffer-and-stringbuilder>
 - <http://www.java-tips.org/java-se-tips-100019/24-java-lang/2009-difference-between-string-stringbuffer-and-stringbuilder.html>
- Recuerde que tiene disponible el código fuente de las clases de Java en la instalación (scr.zip).
4. **Considere nuevamente el ejercicio nro. 2.** En este caso, tenemos un proyecto de sólo 2 clases, pero podríamos tener una aplicación mucho más compleja y una forma de distribuirla es creando un archivo con extensión “.jar”. Desde el eclipse, exporte las 2 clases en un archivo JAR que ejecute el método main de TestCharly (preste atención a las opciones que aparecen durante el Wizard, en particular cuando deba indicar cuál es la clase con el método “main”).
 - i. ¿Qué archivo nuevo se generó dentro del JAR?
 - ii. Ejecute el archivo JAR generado.
 5. **Instituto de Artes Audiovisuales.** El Instituto de Artes Audiovisuales está planeando adquirir un sistema de software que le permita administrar y mantener una base de datos de los recursos y materiales con los que cuentan los alumnos.
Los requerimientos del sistema son los siguientes:
 - El sistema debe permitir ingresar información acerca de textos, Memoria Externa o Video
 - El sistema debe permitir listar información de todos los recursos y su detalle
 - El sistema debe permitir registrar la siguiente información:
 - o Acerca de los textos: título, autor, número de páginas, si está o no disponible y un comentario
 - o Acerca de los CDs: título del álbum, nombre del artista, lenguaje del CD, duración, si está o no disponible y un comentario
 - o Acerca de los videos: título del video, nombre del director, lenguaje del video, duración, si está o no disponible y un comentario

El Instituto de Artes Audiovisuales está evaluando dos propuestas realizadas por dos miembros del equipo de desarrollo. Su tarea es hacer de “consultor” y ayudar a despejar las dudas de modo que se seleccione la mejor opción.

OPCIÓN 1:



```

import java.util.ArrayList;
import java.util.Iterator;
/**
 * The database class provides a facility to store Book, Video and CD
 * objects. A list of all books, CDs and videos can be printed to the
 * terminal.
 */
public class Database {
    private ArrayList textbooks;
    private ArrayList cds;
    private ArrayList videos;
    /**
     * Construct an empty Database.
     */
    public Database() {
        textbooks = new ArrayList();
        cds = new ArrayList();
        videos = new ArrayList();
    }
    /** * Add a textbook to the database.
     */
    public void addTextbook(Textbook textbook) {
        textbooks.add(textbook);
    }
    /**
     * Add a CD to the database.
     */
    public void addCD(CD cd) {

```

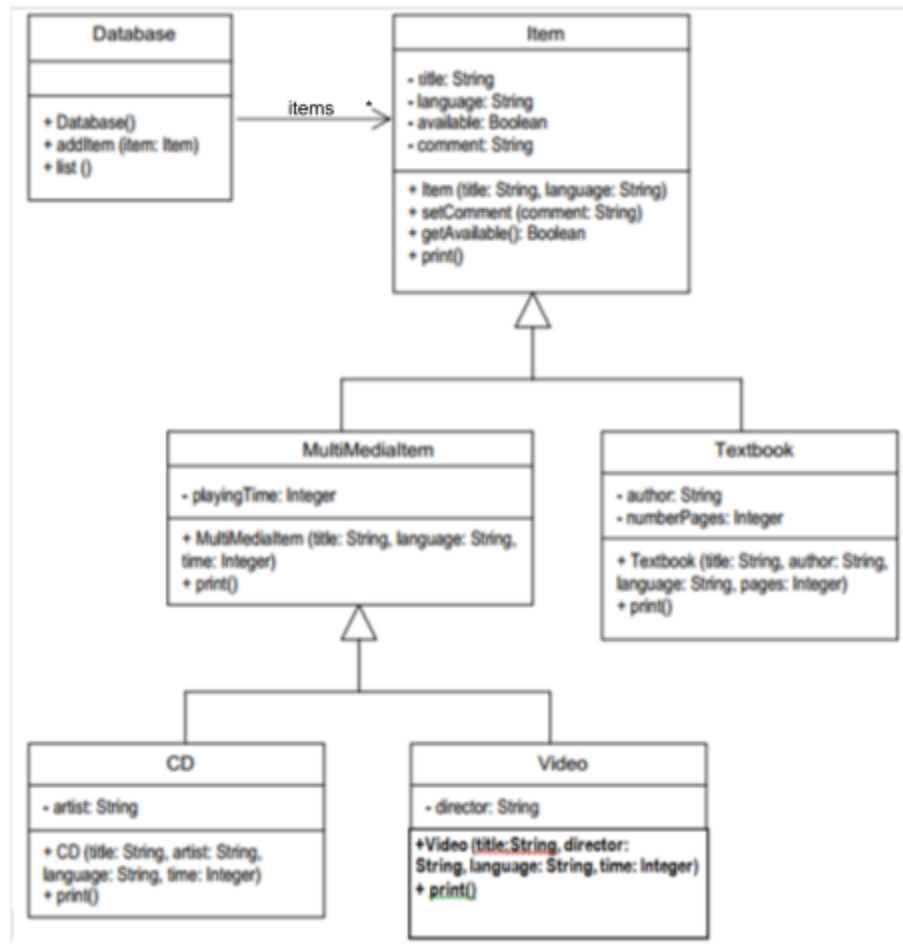
```
        cds.add(cd);
    }
    /**
     * Add a video to the database.
     */
    public void addVideo(Video video) {
        videos.add(video);
    }

    /**
     * Print a list of all currently stored CDs and videos to the text terminal.
     */
    public void list() {
        // print list of textbooks
        for(Iterator iter = textbooks.iterator(); iter.hasNext(); ) {
            Textbook textbook = (Textbook)iter.next();
            textbook.print();
            System.out.println(); // empty line between items
        }
        // print list of CDs
        for(Iterator iter = cds.iterator(); iter.hasNext(); ) {
            CD cd = (CD)iter.next();
            cd.print();
            System.out.println(); // empty line between items
        }
        // print list of videos
        for(Iterator iter = videos.iterator(); iter.hasNext(); ) {
            Video video = (Video)iter.next();
            video.print();
            System.out.println(); // empty line between items
        }
    }
}
/*****
 * The Textbook class represents a textbook object.
 *****/
public class Textbook {
    private String title;
    private String author;
    private String language;
    private int numberPages;
    private boolean available;
    private String comment;
    /**
     * Initialize the Book.
     */
    public Textbook(String title, String author, String language, int pages) {
        this.title = title;
        this.author = author;
        this.language = language;
        numberPages = pages;
        available = true;
    }
}
```

```
        comment = "<no comment>";
    }
    /**
     * Enter a comment for this Book.
     */
    public void setComment(String comment) {
        this.comment = comment;
    }
    /**
     * Return the flag indicating whether this book is available.
     */
    public boolean getAvailable() {
        return available;
    }
    /**
     * Print details about this book to the text terminal.
     */
    public void print() {
        System.out.print("Title: " + title);
        if(available) System.out.println("*");
        else System.out.println();
        System.out.println("Comment: " + comment);
        System.out.println("Language: " + language);
        System.out.println("Number Pages: " + numberPages + " pages");
        System.out.println("Author: " + author);
    }
}
/*****
 * The CD class represents a CD object.
 *****/
public class CD {
    private String title;
    private String artist;
    private String language;
    private int playingTime;
    private boolean available;
    private String comment;
    public CD(String title, String artist, String language, int time) {
        this.title = title;
        this.artist = artist;
        this.language = language;
        playingTime = time;
        available = true;
        comment = "<no comment>";
    }
    public void setComment(String comment) {
        this.comment = comment;
    }
    public boolean getAvailable() {
        return available;
    }
    public void print() {
```

```
        System.out.print("Title: " + title);
        if(available) System.out.println("*");
        else System.out.println();
        System.out.println("Comment: " + comment);
        System.out.println("Language: " + language);
        System.out.println("Playing time: " + playingTime + " mins");
        System.out.println("Artist: " + artist);
    }
}
/*****
 * The Video class represents a video object.
 * *****/
public class Video {
    private String title;
    private String director;
    private String language;
    private int playingTime;
    private boolean available;
    private String comment;
    public Video(String title, String director, String language, int time) {
        this.title = title;
        this.director = director;
        this.language = language;
        playingTime = time;
        available = true;
        comment = "<no comment>";
    }
    public void setComment(String comment) {
        this.comment = comment;
    }
    public boolean getAvailable() {
        return available;
    }
    public void print() {
        System.out.print("Title: " + title);
        if(available) System.out.println("*");
        else System.out.println();
        System.out.println("Comment: " + comment);
        System.out.println("Language: " + language);
        System.out.println("Playing time: " + playingTime + " mins");
        System.out.println("Director: " + director);
    }
}
```

OPCIÓN 2:



```

import java.util.ArrayList;
import java.util.Iterator;
/*****
 * The database class provides a facility to store CD and video
 * objects. A list of all textbooks, CDs and videos can be printed to the terminal.
 *****/

public class Database {
    private ArrayList items;
    public Database() {
        items = new ArrayList();
    }

    public void addItem(Item item) {
        items.add(item);
    }
    /**
     * Print a list of all currently stored CDs and videos to the
     * text terminal.
     */
    public void list() {
        for(Item item: items) {
            item.print();
        }
    }
}
  
```



```
        System.out.println(); // empty line between items
    }
}
}
/*****
* The Item class represents an item.
* This class serves as a superclass for more specific items.
*****/
public class Item {
    private String title;
    private boolean available;
    private String comment;
    private String language;
    public Item(String title, String language) {
        this.title = title;
        available = true;
        this.language = language;
        comment = "<no comment>";
    }
    public void setComment(String comment) {
        this.comment = comment;
    }
    public boolean available() {
        return available;
    }
    public void print() {
        System.out.print("Title: " + title);
        if(available) System.out.println("*");
        else System.out.println();
        System.out.println("Comment: " + comment);
        System.out.println("Language: " + language);
    }
}
/*****
* This subclass of the class Item represents a textbook object.
*****/
public class Textbook extends Item {
    private String author;
    private int numberPages;
    public Textbook(String title, String author, String language, int pages) {
        super(title, language);
        this.author = author;
        numberPages = pages;
    }

    public void print() {
        super.print();
        System.out.println("Number of Pages: " + numberPages + "pages");
        System.out.println("Author: " + author);
    }
}
/*****
```

```

* This subclass of Item represents multi-media resources.
*****/
public class MultiMediaItem extends Item {
    private int playingTime;
    public MultiMediaItem(String title, String language, int time) {
        super(title, language);
        playingTime = time;
    }

    public void print() {
        super.print();
        System.out.println("Playing Time: " + playingTime + "mins");
    }
}

/*****
* This subclass of MultiMediaItem represents a CD object.
*****/
public class CD extends MultiMediaItem {
    private String artist;
    public CD(String title, String artist, String language, int time) {
        super(title, language, time);
        this.artist = artist;
    }

    public void print() {
        super.print();
        System.out.println("Artist: " + artist);
    }
}

/*****
* This subclass of the class MultiMediaItem represents a video object.
*****/
public class Video extends MultiMediaItem {
    private String director;
    public Video(String theTitle, String theDirector, String theLanguage, int time) {
        super(theTitle, theLanguage, time);
        director = theDirector;
    }
    public String getDirector() {
        return director;
    }
    public void print() {
        super.print();
        System.out.println("Director: " + director);
    }
}

```

Criterio\Solución	Opción 1	Opción 2
Abstracción del mundo real		

<p>¿Considera que los objetos del mundo real están modelados en el sistema?</p> <p>¿Hay organización entre las clases (nivel de abstracción)?</p>		
<p>Duplicación de código</p> <p>¿Hay duplicación de código?</p> <p>¿Es necesaria o puede ser evitada?</p>		
<p>Re-uso de código</p> <p>¿Se re-usa código?</p> <p>¿Puede el código actual ser re-usado si el sistema crece para abarcar otros tipos de elementos audiovisuales?</p>		
<p>Mantenimiento del programa</p> <p>¿Qué cambios aparecen si se quiere agregar el DVD? Piense particularmente en el conjunto de DVDs que tendría el Database</p> <p>¿Qué tan fácil es cambiar el código?</p>		
<p>Extensibilidad</p> <p>¿Qué tan sencillo es agregar un nuevo tipo de recurso como libros electrónicos?</p> <p>¿Cuáles son los cambios a realizar?</p>		
<p>Polimorfismo</p> <p>Indique –en caso de existir- donde está aplicado este concepto</p>		

Responda verdadero o falso según corresponda. JUSTIFIQUE.

- Una clase abstracta **debe tener** al menos un método abstracto. JUSTIFIQUE.
- No pueden definirse** constructores en una clase abstracta. JUSTIFIQUE.
- Los métodos abstractos **no pueden ser declarados** en una clase no abstracta (concreta). JUSTIFIQUE
- Si una clase B extiende una clase abstracta A **debe implementar TODOS** los métodos de la clase abstracta A. JUSTIFIQUE