

**Tema**

HILOS JAVA (168-178)

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

10/07/2023

DOMINIOS DE SEGURIDAD EN APLICACIONES JEE

[1. MARCO TEÓRICO 4](#_Toc120481017)

[1.1. OBJETIVOS 4](#_Toc120481018)

[1.2.1. OBJETIVO GENERAL 4](#_Toc120481019)

[1.2.2. OBJETIVOS ESPECÍFICOS 4](#_Toc120481020)

[1.2. JAVA. 4](#_Toc120481021)

[1.3. APACHE NETBEANS 4](#_Toc120481022)

[1.4. CONCURRENCIA 5](#_Toc120481023)

[1.5. HILOS 5](#_Toc120481024)

1.6 . SINCRONIZACION DE HILOS……………………………………………………………………………………7

[1.7. HILOS PARALELOS Y CONCURRENTES 5](#_Toc120481025)

[1.8. TERMINOLOGÍA Y METODOS DE HILOS EN JAVA. 5](#_Toc120481026)

[2. PARTE PRÁCTICA 7](#_Toc120481027)

[2.1. Video 171 7](#_Toc120481028)

[2.1.1 CREACIÓN DEL PROYECTO. 7](#_Toc120481029)

[2.1.2 Creación de paquetes MVC. 8](#_Toc120481030)

[2.1.3 CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 10](#_Toc120481031)

[2.1.4. CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 12](#_Toc120481032)

[2.1.5. CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 16](#_Toc120481033)

[2.1.6. CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN 18](#_Toc120481034)

[2.1.7. EJECUCIÓN DEL PROYECTO. 20](#_Toc120481035)

[2.2. Video 172 21](#_Toc120481036)

[2.1.1 CREACIÓN DEL PROYECTO. 21](#_Toc120481037)

[2.1.4 Creación de paquetes MVC. 22](#_Toc120481038)

[2.1.5 CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 23](#_Toc120481039)

[2.1.8. CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 26](#_Toc120481040)

[2.1.9. CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 28](#_Toc120481041)

[2.1.10. CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN 32](#_Toc120481042)

[2.1.11. EJECUCIÓN DEL PROYECTO. 34](#_Toc120481043)

[3. CONCLUSIONES 35](#_Toc120481044)

[4. RECOMENDACIONES 35](#_Toc120481045)

[5. BIBLIOGRAFIA 35](#_Toc120481046)

**INDICE DE IMÁGENES**

[Figura 1. Logotipo identificativo de Java 4](#_Toc120481112)

[Figura 2. Logotipo identificativo de Netbeans 7](#_Toc120481113)

[Figura 3. Selección del tipo de aplicación a desarrollar 7](#_Toc120481114)

[Figura 4. Nombre del proyecto 8](#_Toc120481115)

[Figura 5. Estructura del proyecto generado 8](#_Toc120481116)

[Figura 6. Creación de paquete. 9](#_Toc120481117)

[Figura 7. Selección del nombre de los paquetes del video 171 9](#_Toc120481118)

[Figura 8. Creación de clase Java. 10](#_Toc120481119)

[Figura 9. Codificación archivo HilosVista. 10](#_Toc120481120)

[Figura 10. Creación de clase Java. 12](#_Toc120481121)

[Figura 11. Codificación archivo HilosModelo. 12](#_Toc120481122)

[Figura 12. Creación de clase Java. 14](#_Toc120481123)

[Figura 13. Codificación archivo HilosVarios 14](#_Toc120481124)

[Figura 14. Creación de clase Java. 16](#_Toc120481125)

[Figura 15.Codificación archivo HilosControlador. 16](#_Toc120481126)

[Figura 16. Codificación archivo principal. 18](#_Toc120481127)

[Figura 17. Resultado de la ejecución 20](#_Toc120481128)

[Figura 18. Logotipo identificativo de Netbeans 21](#_Toc120481129)

[Figura 19. Selección del tipo de aplicación a desarrollar 21](#_Toc120481130)

[Figura 20. Nombre del proyecto 22](#_Toc120481131)

[Figura 21. Estructura del proyecto generado. 22](#_Toc120481132)

[Figura 22. Creación de paquete. 23](#_Toc120481133)

[Figura 23. Selección del nombre de los paquetes del video 172 23](#_Toc120481134)

[Figura 24. Creación de clase Java 24](#_Toc120481135)

[Figura 25. Codificación archivo HilosVista 24](#_Toc120481136)

[Figura 26. Creación de clase Java 26](#_Toc120481137)

[Figura 27. Codificación archivo HilosModelo 26](#_Toc120481138)

[Figura 28. Creación de clase Java. 28](#_Toc120481139)

[Figura 29. Codificación archivo HilosVariosControlador 28](#_Toc120481140)

[Figura 30. Creación de clase Java. 30](#_Toc120481141)

[Figura 31. Codificación archivo HilosVarios2Controlador 30](#_Toc120481142)

[Figura 32. Codificación archivo principal 32](#_Toc120481143)

[Figura 33. Resultado de la ejecución. 34](#_Toc120481144)

**ÍNDICE DE TABLAS**

[Tabla 1. Terminología utilizada en Hilos Java 5](#_Toc120481057)

[Tabla 2. Código de HilosVista. 11](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481058)

[Tabla 3. Codificación del archivo HilosModelo 13](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481059)

[Tabla 4. Codificación del archivo HilosVarios 15](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481060)

[Tabla 5. Codificación del archivo HilosControlador 17](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481061)

[Tabla 6.Codificación del archivo principal del proyecto 19](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481062)

[Tabla 7. Código de HilosVista. 25](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481063)

[Tabla 8. Codificación del archivo HilosModelo 27](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481064)

[Tabla 9. Codificación del archivo Controlador 29](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481065)

[Tabla 10. Codificación del archivo HilosVarios2Controlador 31](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481066)

[Tabla 11. Codificación del archivo principal del proyecto 33](file:///C:\Users\rikye\Desktop\HilosJava_Video172_grupo6.docx#_Toc120481067)

# MARCO TEÓRICO

## OBJETIVOS

### OBJETIVO GENERAL

Comprender la funcionalidad de los hilos en Java para ejecutar multiples tareas al mismo así como también ver la manera en la que un programa realizado con hilos actúa de manera diferente a uno que no los utiliza.

### OBJETIVOS ESPECÍFICOS

* Analizar le proceso de los hilos en java.
* Conocer la forma de utilizar los hilos en el lenguaje de programación de java.
* Conocer la forma de ejecutar múltiples hilos al mismo tiempo y a su vez detenerlos.

## JAVA.

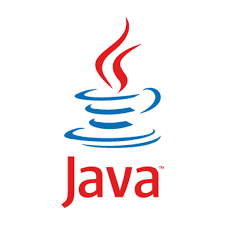


Figura 1. Logotipo identificativo de Java

Java es según la definición dada por [1] es un lenguaje de programación ampliamente utilizado para codificar aplicaciones de distintos indoles, desde aplicaciones de escritorio hasta tipo web, pasando por desarrollo de videojuegos, macrodatos, IA e IoT. Actualmente la ultima versión numero 8 y cuenta con 3 versiones disponibles de Java las cuales son Java SE, Java EE y Java ME.

La mayor fortaleza de java es su capacidad de ser un lenguaje sumamente apto para ser aplicado en el paradigma orientado a objetos, lo cual es su mayor fortaleza en este aspecto.

## APACHE NETBEANS

Apache netbeans es [2] un entorno de desarrollo integrado (IDE) creado para principalmente el lenguaje de programación de Java, anteriormente conocido simplemente como netbeans, cambio su nombre a apache netbeans y actualmente se encuentra en la versión 15.

Este IDE ofrece una serie de plugins que pueden ser instalados para facilitar el trabajo al momento de desarrollar aplicaciones y ejecutarlas, como por ejemplo poder instalar de manera rápida y fácil el servidor de payara o glassfish.

Cabe recalcar que apache netbeans es completamente gratis para su uso tanto para fines empresariales como educativos, y que se puede encontrar fácilmente para su descarga en el sitio oficial, teniendo versiones para los sistemas operativos Windows, Mac, Linux y Solaris.

## CONCURRENCIA

La concurrencia es [3] la ejecución de varios procesos a la vez, es decir, es la ejecución simultánea de múltiples tareas interactivamente. Estas tareas pueden ser un conjunto de procesos o hilos de ejecución creados por un único programa. Las tareas se pueden ejecutar en una sola CPU (multiprogramación), en varios procesadores, o en una red de computadores distribuidos.

## HILOS

Un hilo es [3] un flujo de control dentro de un programa. Creando varios hilos podremos realizar varias tareas simultáneamente. Cada hilo tendrá sólo un contexto de ejecución (contador de programa, pila de ejecución). Es decir, a diferencia de los procesos UNIX, no tienen su propio espacio de memoria, sino que acceden todos al mismo espacio de memoria común, por lo que será importante su sincronización cuando tengamos varios hilos accediendo a los mismos objetos.

Los hilos son útiles porque permiten que el flujo del programa sea divido en dos o más partes, cada una ocupándose de alguna tarea de forma independiente. Por ejemplo un hilo puede encargarse de la comunicación con el usuario, mientras que otros actúan en segundo plano, realizando la transmisión de un fichero, accediendo a recursos del sistema (cargar sonidos, leer ficheros ...), etc.

## SINCRONIZACION DE HILOS

La sincronización de hilos es fundamental para garantizar la consistencia y la integridad de los datos compartidos entre hilos. Sin una sincronización adecuada, pueden surgir problemas como la condición de carrera, donde múltiples hilos intentan acceder o modificar un recurso compartido al mismo tiempo, y la inconsistencia de datos, donde los cambios realizados por un hilo no son visibles para otros hilos.

Existen varias técnicas de sincronización de hilos, como los bloqueos (locks), semáforos, monitores, entre otros. Estas técnicas permiten establecer secciones críticas, donde solo un hilo puede acceder al recurso compartido a la vez, o sincronizar eventos para garantizar un orden específico de ejecución entre hilos.

## HILOS PARALELOS Y CONCURRENTES

Un hilo como se indicó con anterioridad se utiliza sobre todo para dividir tareas y hacerla de manera más eficiente, además como indica [4] permitiendo que estas se ejecuten de manera simultánea en vez de terminar una para continuar otra.

De manera más formal [5] “cada hilo progresa de forma independiente al resto. Esto provoca que cada uno de los hilos pueda potencialmente viajar a una velocidad distinta, ejecutándose "concurrentemente" con el resto. Esto provoca que la ejecución de un programa sea diferente en cada da pasada y que sea independiente.”

## TERMINOLOGÍA Y METODOS DE HILOS EN JAVA.

Antes de realizar la practica presente es necesario conocer algunos términos que son necesarios para entender los códigos y algunas definiciones que se utilizaran durante la práctica.

Tabla 1. Terminología utilizada en Hilos Java

|  |  |
| --- | --- |
| TÉRMINO O METODO | DESCRIPCIÓN |
| isInterrupted() | Método que verifica si un hilo esta interrumpido o no. |
| Concurrencia | En programación significa cuando tareas se ejecutan de manera simultanea. |
| resume() | Nombre o etiqueta que clasifica a los usuarios con un conjunto de características comunes que generalmente conduce a un conjunto de permisos comunes. Los grupos pueden definirse ampliamente y reflejar los grupos de la vida real como "admin" o "manager”. |
| join() | Al utilizar este método nos aseguramos que no se ejecute otro hilo hasta que éste finalice |
| sleep(long ms) | Pone el hilo en pausa el tiempo en milisegundos que le introduzcamos |
| yield() | Pausa el hilo en ejecución para permitir la ejecución de otros |
| start() | Indica al intérprete de Java que cree un contexto del hilo del sistema y comience a ejecutarlo |
| IDE | Entorno de desarrollo integrado (IDE), es un software que permite desarrollar aplicaciones con herramientas que ayudan al programador a organizar su flujo de trabajo y agilizando todo el proceso de diseño de software, a través de una interfaz gráfica. |
| stop() | Provoca que el hilo se detenga de manera inmediata |
| resume() | Reanuda un hilo suspendido |
| setPriority() | Asigna la prioridad al hilo indicada por el valor pasado como parámetro |
| getPriority() | Devuelve la prioridad del hilo de ejecución en curso |
| setName() | Asigna un nombre para identificar a los hilos de una forma más cómoda |
| getName() | Devuelve el nombre asignado |

# PARTE PRÁCTICA

## Video 168

### 2.1.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.



Figura 2. Logotipo identificativo de Netbeans

Primero dentro del disco C y cree una carpeta llamada DISTRIBUIDAS allí crea otra subcarpeta llamada FORMULARIOSWEB, dentro de esta crea otra subcarpeta con el nombre de HilosJava a su vez dentro de esta última cree 2 carpetas, una con el nombre de “aplicativo” y otra con el nombre de “documentación”. Una vez haya creado las carpetas abre Apache Netbeans y se sitúa en la Pestaña superior y selecciona File->New Project -> Java with Mave -> Web Application

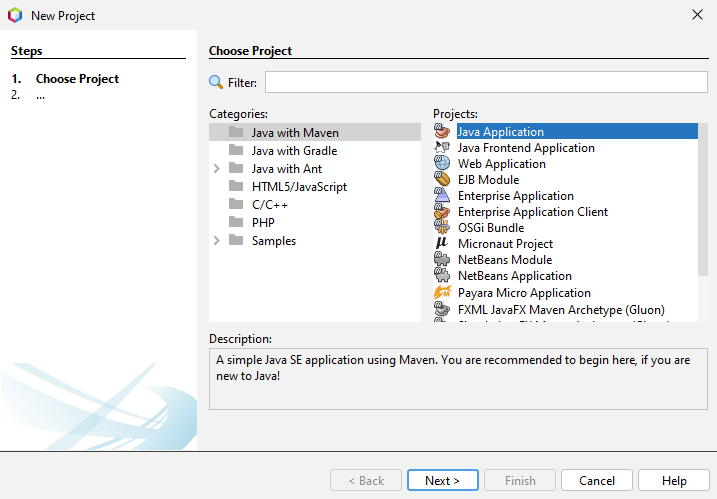


Figura 3. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre HilosJava\_Video168\_grupo#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “aplicativo” que fue creada con anterioridad, en el grupo Id “ec.edu.monster.main” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

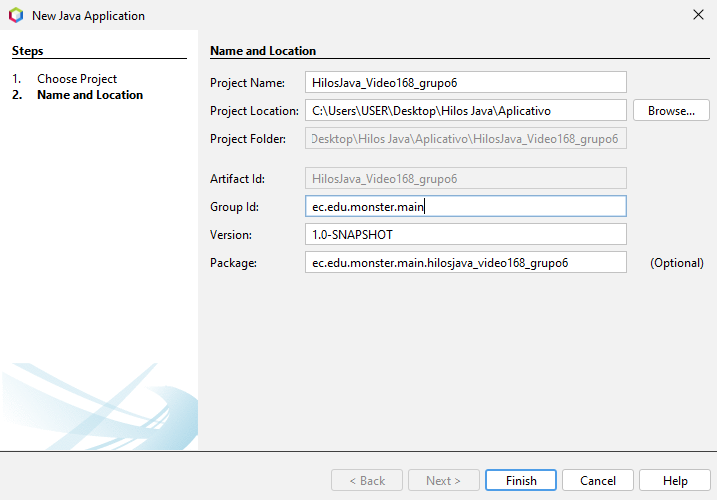


Figura 4. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

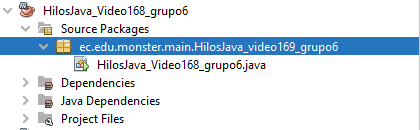


Figura 5. Estructura del proyecto generado.

Si desea para mas orden renombre el paquete creado inicialmente (ec.edu.monster.main.HilosJava\_video169\_grupo#) por “ec.edu.monster.main” Seleccione el paquete, luego->refactor->Rename.

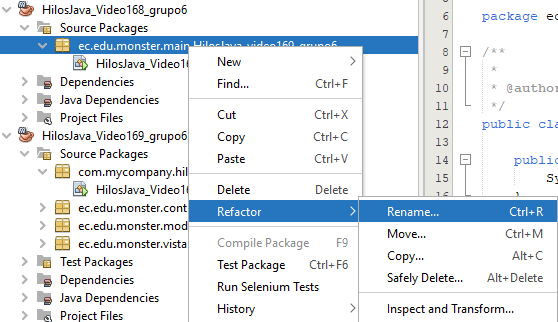


Figura 6. Renombre del paquete principal.

Abra la carpeta Project Files y selecciones el archivo pom.xml y en la etiqueta <group Id> coloque “ec.edu.monster.main”, mientras que en la etiqueta <exec.mainClass> “ec.edu.monster.main.HilosJava\_Video168\_grupo#”.

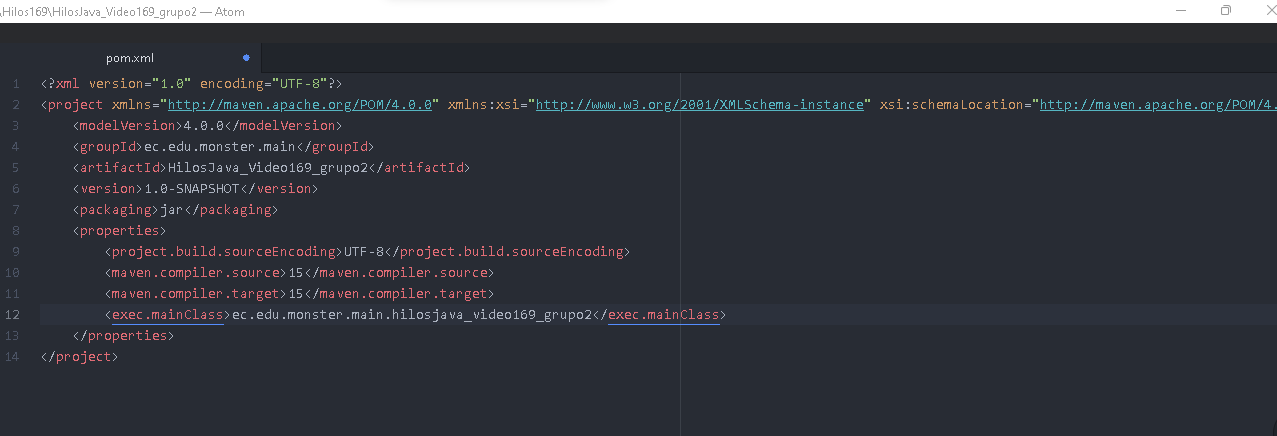


Figura 7. Modificacion del archivo pom.xml.

### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

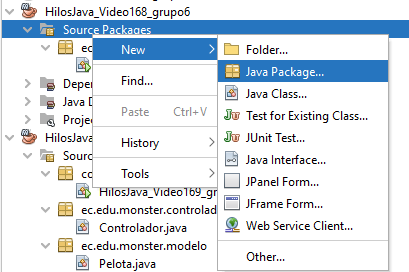


Figura 8. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

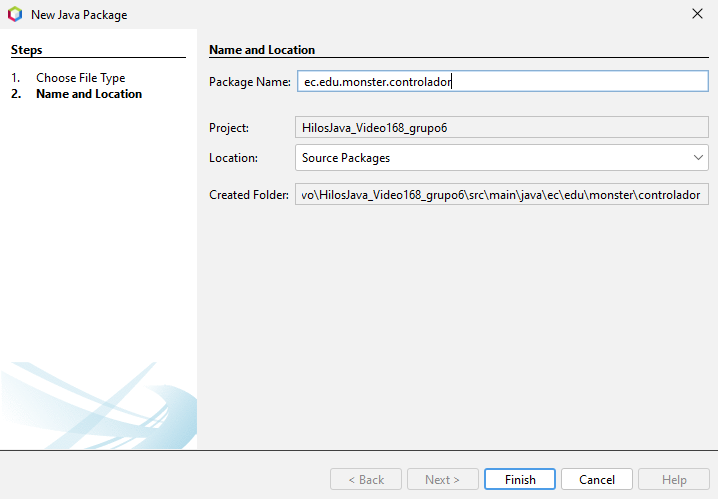


Figura 9. Selección del nombre de los paquetes del video 168

Repita el proceso hasta tener los 3 paquetes creados

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

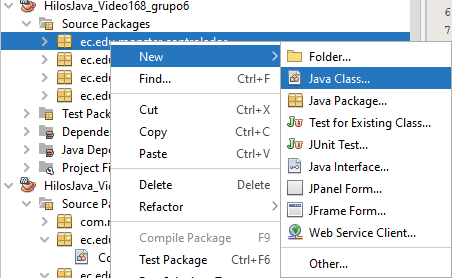


Figura 10. Creación de clase Java.

Le da el nombre de “Vista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

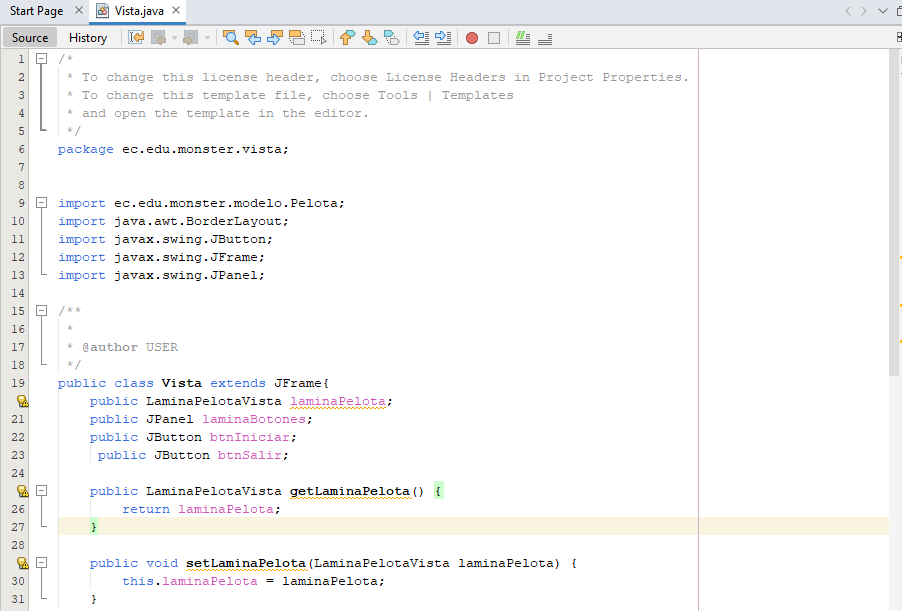


Figura 11. Codificación archivo Vista.

Tabla 2. Código de Vista.

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Pelota;

import java.awt.BorderLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\*

\* @author USER

\*/

public class Vista extends JFrame{

public LaminaPelotaVista laminaPelota;

public JPanel laminaBotones;

public JButton btnIniciar;

public JButton btnSalir;

public LaminaPelotaVista getLaminaPelota() {

return laminaPelota;

}

public void setLaminaPelota(LaminaPelotaVista laminaPelota) {

this.laminaPelota = laminaPelota;

}

public Vista(){

setBounds(600,300,400,350);

setTitle ("Pelotas en movimiento");

laminaBotones=new JPanel();

this.laminaPelota=new LaminaPelotaVista();

add(this.laminaPelota, BorderLayout.CENTER);

btnIniciar=new JButton();

btnSalir=new JButton();

add(laminaBotones, BorderLayout.SOUTH);

}

public void AñadirPelota(Pelota m){

laminaPelota.add(m);

}

} \*/

public class Vista extends JFrame{

public LaminaPelotaVista laminaPelota;

public JPanel laminaBotones;

public JButton btnIniciar;

public JButton btnSalir;

public LaminaPelotaVista getLaminaPelota() {

return laminaPelota;

}

public void setLaminaPelota(LaminaPelotaVista laminaPelota) {

this.laminaPelota = laminaPelota;

}

public Vista(){

setBounds(600,300,400,350);

setTitle ("Pelotas en movimiento");

laminaBotones=new JPanel();

this.laminaPelota=new LaminaPelotaVista();

add(this.laminaPelota, BorderLayout.CENTER);

btnIniciar=new JButton();

btnSalir=new JButton();

add(laminaBotones, BorderLayout.SOUTH);

}

public void AñadirPelota(Pelota m){

laminaPelota.add(m);

}

}

</tr>

<tr>

<td>PHP</td>

<td><input type="checkbox" name="chkLeng" value="PHP" /></td>

</tr>

<tr>

<td>Ruby</td>

<td><input type="checkbox" name="chkLeng" value="Ruby" /></td>

</tr>

<tr>

<td>Python</td>

<td><input type="checkbox" name="chkLeng" value="Python" /></td>

</tr>

<tr>

<td></td>

<td><input type="submit" value="Enviar" /></td>

</tr>

</tbody>

</table>

</form>

</body>

</html>

public void setLaminaPelota(LaminaPelotaVista laminaPelota) {

this.laminaPelota = laminaPelota;

}

public Vista(){

setBounds(600,300,400,350);

setTitle ("Pelotas en movimiento");

laminaBotones=new JPanel();

this.laminaPelota=new LaminaPelotaVista();

add(this.laminaPelota, BorderLayout.CENTER);

btnIniciar=new JButton();

btnSalir=new JButton();

add(laminaBotones, BorderLayout.SOUTH);

}

public void AñadirPelota(Pelota m){

laminaPelota.add(m);

}

}

Cree otra clase Java.

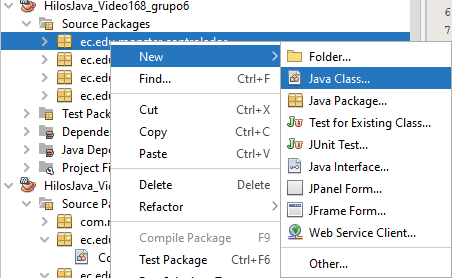


Figura 13. Creación de clase java.

Le da el nombre de “LaminaPelotaVista”, y proceda a colocar el siguiente código para construir la segunda clase en el paquete de vista.

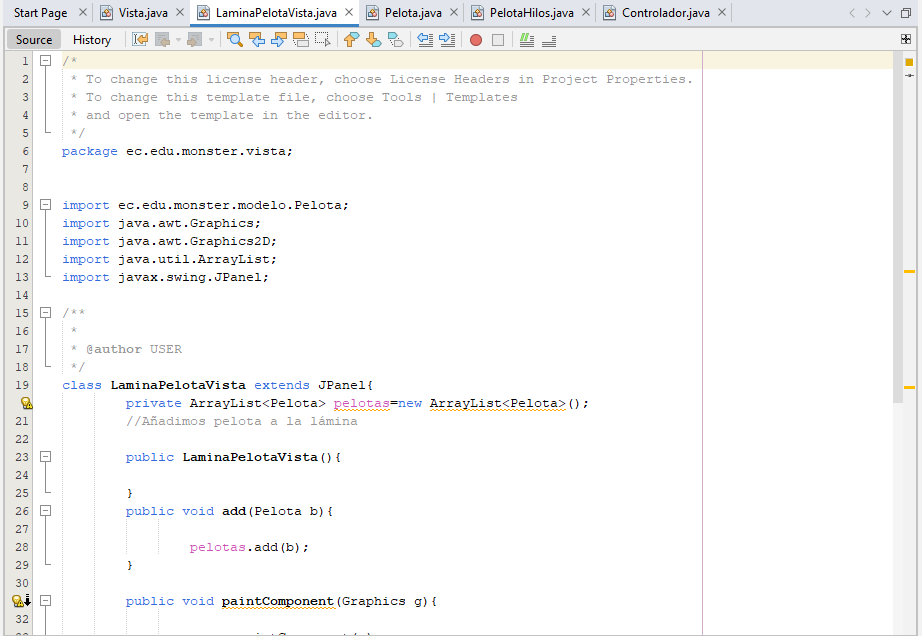


Figura 14. Codificación archivo LaminaPelotaVista.

Tabla 3. Codificación del archivo Lamina Pelota Vista.

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Pelota;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.util.ArrayList;

import javax.swing.JPanel;

/\*\*

\*

\* @author USER

\*/

class LaminaPelotaVista extends JPanel{

private ArrayList<Pelota> pelotas=new ArrayList<Pelota>();

//Añadimos pelota a la lámina

public LaminaPelotaVista(){

}

public void add(Pelota b){

pelotas.add(b);

}

public void paintComponent(Graphics g){

super.paintComponent(g);

Graphics2D g2=(Graphics2D)g;

for(Pelota b: pelotas){

g2.fill(b.getShape());

}

}

}

public void paintComponent(Graphics g){

super.paintComponent(g);

Graphics2D g2=(Graphics2D)g;

for(Pelota b: pelotas){

g2.fill(b.getShape());

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

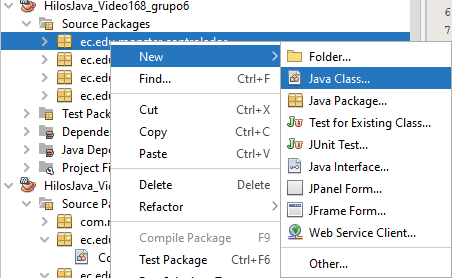


Figura 15. Creación de clase Java.

Le da el nombre de “Pelota”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

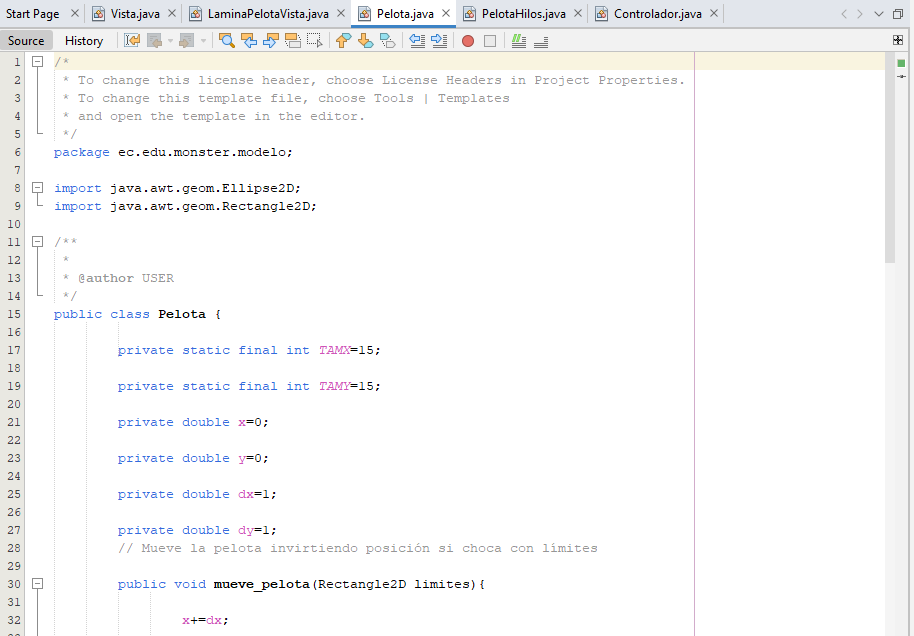


Figura 16. Codificación archivo Pelota.

Tabla 4. Codificación del archivo Pelota

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.modelo;

import java.awt.geom.Ellipse2D;

import java.awt.geom.Rectangle2D;

/\*\*

\*

\* @author USER

\*/

public class Pelota {

private static final int TAMX=15;

private static final int TAMY=15;

private double x=0;

private double y=0;

private double dx=1;

private double dy=1;

// Mueve la pelota invirtiendo posición si choca con límites

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

} <%= lenguajesSeleccionados[i]%>

</li>

<%}

}

%>

</ul>

</body>

</html>

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

}

Cree una nueva clase en el paquete del modelo.

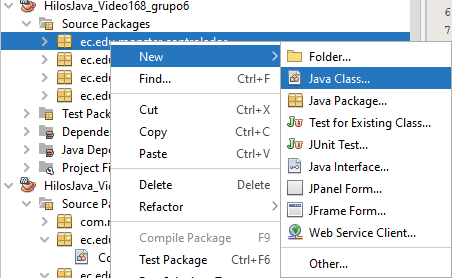


Figura 17. Creación de clase Java.

Le da el nombre de “PelotaHilos”, y proceda a colocar el siguiente código para construir su la segunda clase en el paquete de modelo.

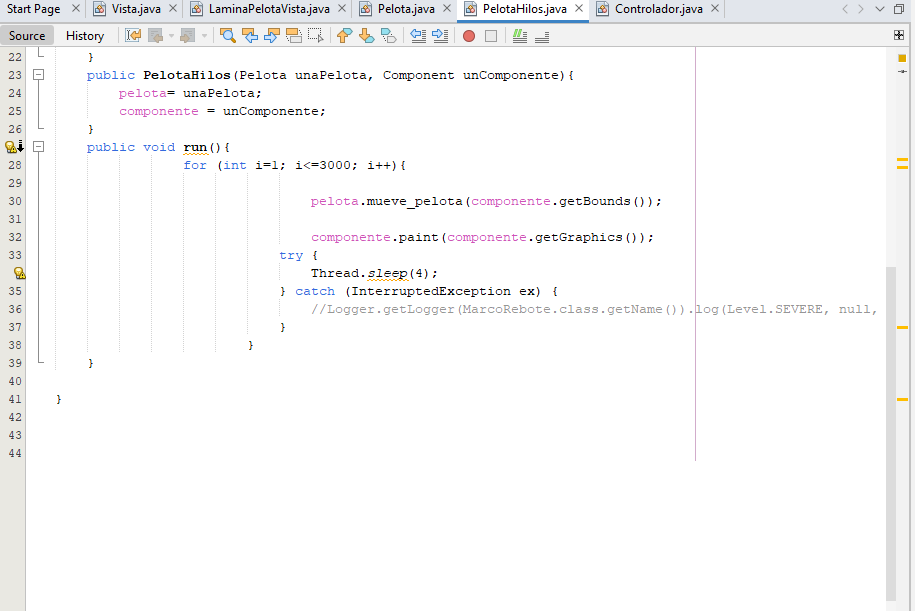


Figura 18. Codificación archivo PelotaHilos

*Tabla 5.* Codificación del archivo PelotaHilos Vista.

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.modelo;

import java.awt.Component;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author USER

\*/

public class PelotaHilos implements Runnable {

private Pelota pelota;

private Component componente;

public PelotaHilos(){

}

public PelotaHilos(Pelota unaPelota, Component unComponente){

pelota= unaPelota;

componente = unComponente;

}

public void run(){

for (int i=1; i<=3000; i++){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

try {

Thread.sleep(4);

} catch (InterruptedException ex) {

//Logger.getLogger(MarcoRebote.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

}

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

} <%= lenguajesSeleccionados[i]%>

</li>

<%}

}

%>

</ul>

</body>

</html>

*aaaa*

public void run(){

for (int i=1; i<=3000; i++){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

try {

Thread.sleep(4);

} catch (InterruptedException ex) {

//Logger.getLogger(MarcoRebote.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

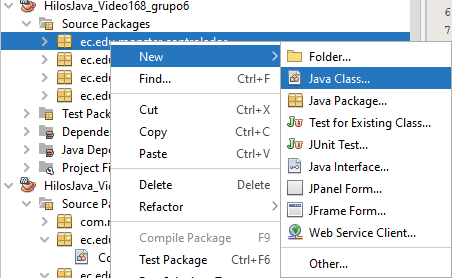


Figura 19. Creación de clase Java.

Le da el nombre de “Controlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

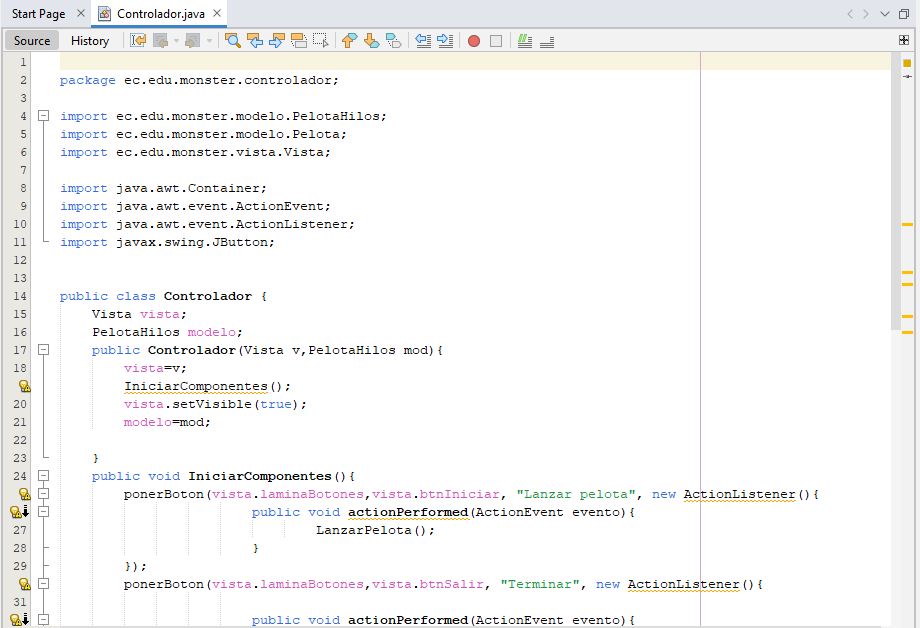


Figura 20. Codificación archivo Controlador.

Tabla 6. Codificación del archivo Controlador

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.PelotaHilos;

import ec.edu.monster.modelo.Pelota;

import ec.edu.monster.vista.Vista;

import java.awt.Container;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

public class Controlador {

Vista vista;

PelotaHilos modelo;

public Controlador(Vista v,PelotaHilos mod){

vista=v;

IniciarComponentes();

vista.setVisible(true);

modelo=mod;

}

public void IniciarComponentes(){

ponerBoton(vista.laminaBotones,vista.btnIniciar, "Lanzar pelota", new ActionListener(){

public void actionPerformed(ActionEvent evento){

LanzarPelota();

}

});

ponerBoton(vista.laminaBotones,vista.btnSalir, "Terminar", new ActionListener(){

public void actionPerformed(ActionEvent evento){

System.exit(0);

}

});

}

public void ponerBoton(Container c,JButton btn, String titulo, ActionListener accion){

btn.setText(titulo);

c.add(btn);

btn.addActionListener(accion);

}

public void LanzarPelota(){

Pelota pelota=new Pelota();

vista.AñadirPelota(pelota);

Runnable r = new PelotaHilos(pelota, vista.laminaPelota);

Thread t = new Thread(r);

t.start();

}

}

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

} <%= lenguajesSeleccionados[i]%>

</li>

<%}

}

%>

</ul>

</body>

</html>

ponerBoton(vista.laminaBotones,vista.btnSalir, "Terminar", new ActionListener(){

public void actionPerformed(ActionEvent evento){

System.exit(0);

}

});

}

public void ponerBoton(Container c,JButton btn, String titulo, ActionListener accion){

btn.setText(titulo);

c.add(btn);

btn.addActionListener(accion);

}

public void LanzarPelota(){

Pelota pelota=new Pelota();

vista.AñadirPelota(pelota);

Runnable r = new PelotaHilos(pelota, vista.laminaPelota);

Thread t = new Thread(r);

t.start();

}

}

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

} <%= lenguajesSeleccionados[i]%>

</li>

<%}

}

%>

</ul>

</body>

</html>

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código.



Figura 21. Codificación archivo principal.

Tabla 7. Codificación del archivo principal del proyecto

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Project/Maven2/JavaApp/src/main/java/${packagePath}/${mainClassName}.java to edit this template

\*/

package ec.edu.monster.main;

import ec.edu.monster.controlador.Controlador;

import ec.edu.monster.modelo.PelotaHilos;

import ec.edu.monster.vista.Vista;

/\*\*

\*

\* @author USER

\*/

public class HilosJava\_Video168\_grupo6 {

public static void main(String[] args) {

Vista vista=new Vista();

PelotaHilos modelo=new PelotaHilos();

Controlador controlador=new Controlador(vista,modelo);

}

}

### EJECUCIÓN DEL PROYECTO.

Ejecute el proyecto y podrá observar como puede lanzar varias pelotas en diferentes momentos, siendo cada pelota una representación de un hilo.



Figura 22. Resultado de la ejecución

## Video 169

### 2.2.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.



Figura 23. Logotipo identificativo de Netbeans

Abra Apache Netbeans y se sitúa en la Pestaña superior y selecciona File->New Project -> Java with Mave -> Web Application

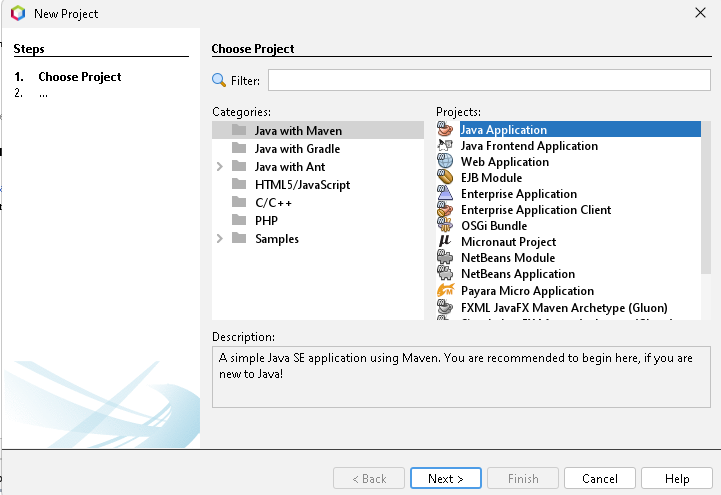


Figura 24. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre HilosJava\_Video169\_grupo#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “aplicativo” que fue creada con anterioridad, en el grupo Id “ec.edu.monster.main” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

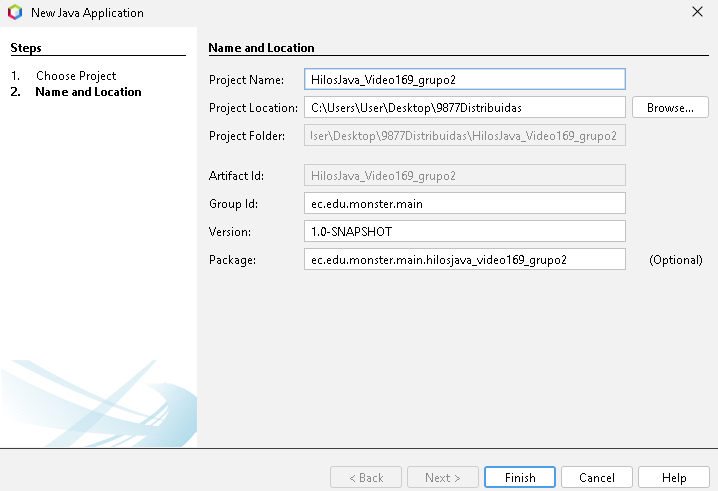
****

Figura 25. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

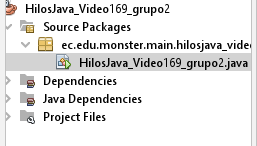
****

Figura 26. Estructura del proyecto generado.

Si desea para mas orden renombre el paquete creado inicialmente (ec.edu.monster.main.HilosJava\_video169\_grupo#) por “ec.edu.monster.main” Seleccione el paquete, luego->refactor->Rename.

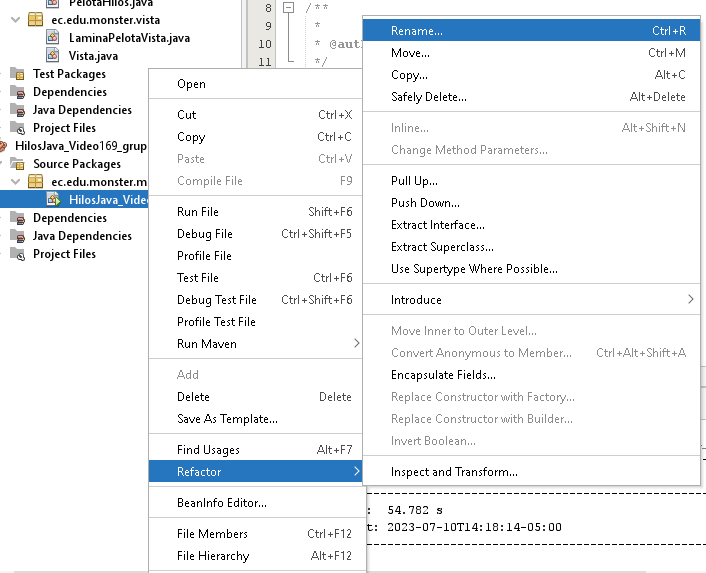


Figura 27. Renombre del paquete principal del proyecto 169

Abra la carpeta Project Files y selecciones el archivo pom.xml y en la etiqueta <group Id> coloque “ec.edu.monster.main”, mientra que en la etiqueta <exec.mainClass> “ec.edu.monster.main.HilosJava\_Video169\_grupo#”.

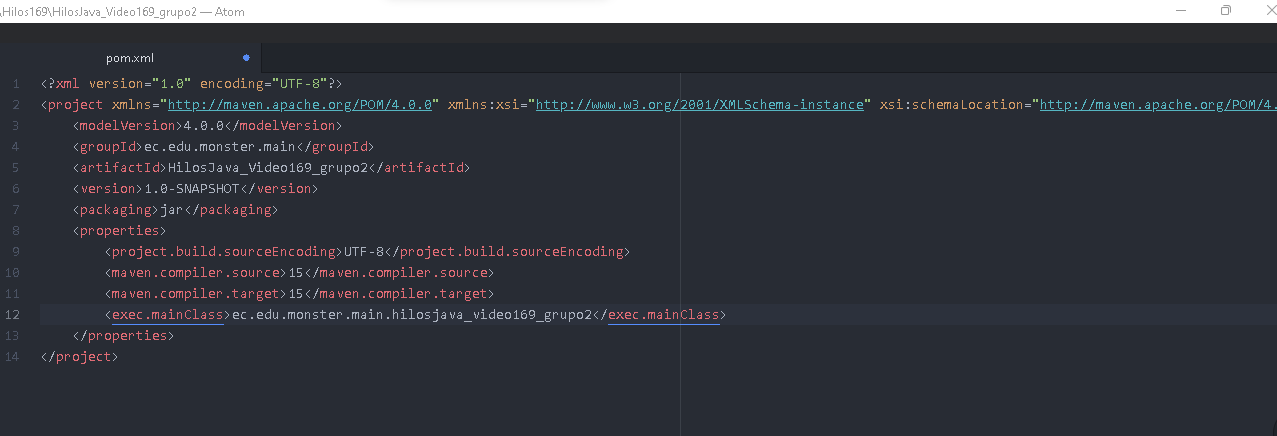


Figura 28. Modificación del paquete pom.xml.

### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

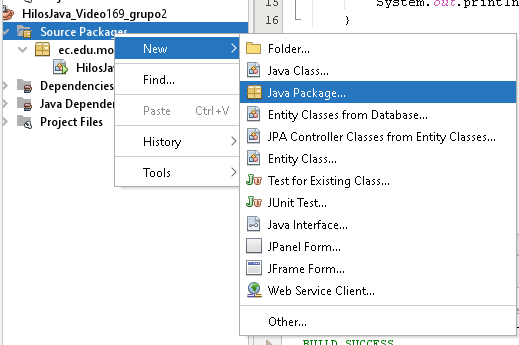


Figura 29. Creación de clase Java.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

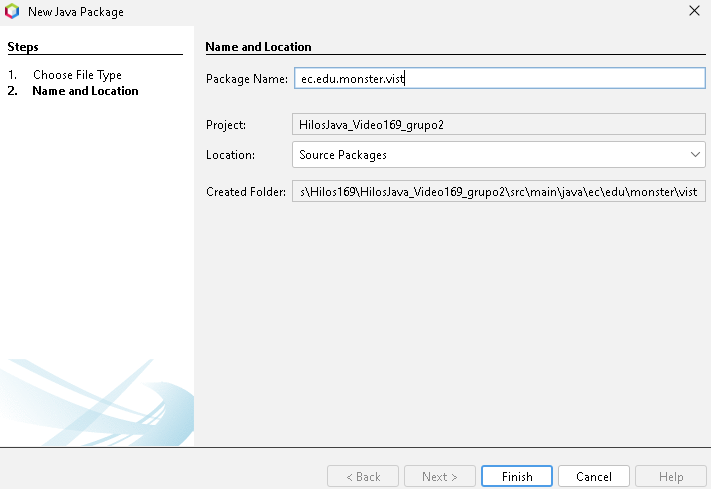


Figura 30. Creación de paquetes proyecto 169.

Repita el proceso hasta tener los 3 paquetes creados

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

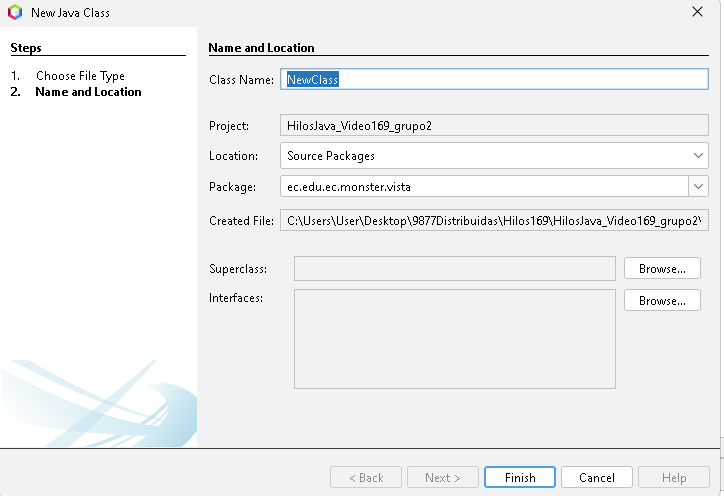


Figura 31. Creación de clase Java.

Le da el nombre de “Vista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

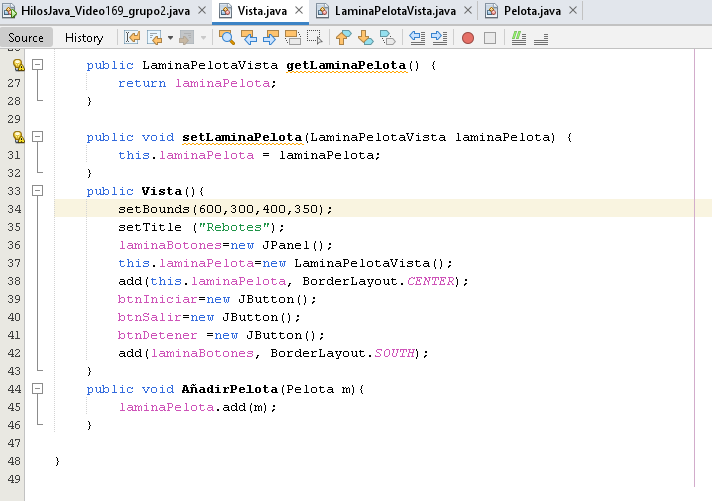


Figura 32. Codificación archivo Vista.

Tabla 8. Codificación del archivo Vista.

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Pelota;

import java.awt.BorderLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\*

\* @author USER

\*/

public class Vista extends JFrame{

public LaminaPelotaVista laminaPelota;

public JPanel laminaBotones;

public JButton btnIniciar;

public JButton btnSalir;

public JButton btnDetener;

public LaminaPelotaVista getLaminaPelota() {

return laminaPelota;

}

public void setLaminaPelota(LaminaPelotaVista laminaPelota) {

this.laminaPelota = laminaPelota;

}

public Vista(){

setBounds(600,300,400,350);

setTitle ("Rebotes");

laminaBotones=new JPanel();

this.laminaPelota=new LaminaPelotaVista();

add(this.laminaPelota, BorderLayout.CENTER);

btnIniciar=new JButton();

btnSalir=new JButton();

btnDetener =new JButton();

add(laminaBotones, BorderLayout.SOUTH);

}

public void AñadirPelota(Pelota m){

laminaPelota.add(m);

}

}

public LaminaPelotaVista getLaminaPelota() {

return laminaPelota;

}

public void setLaminaPelota(LaminaPelotaVista laminaPelota) {

this.laminaPelota = laminaPelota;

}

public Vista(){

setBounds(600,300,400,350);

setTitle ("Rebotes");

laminaBotones=new JPanel();

this.laminaPelota=new LaminaPelotaVista();

add(this.laminaPelota, BorderLayout.CENTER);

btnIniciar=new JButton();

btnSalir=new JButton();

btnDetener =new JButton();

add(laminaBotones, BorderLayout.SOUTH);

}

public void AñadirPelota(Pelota m){

laminaPelota.add(m);

}

}

Cree otra clase Java.

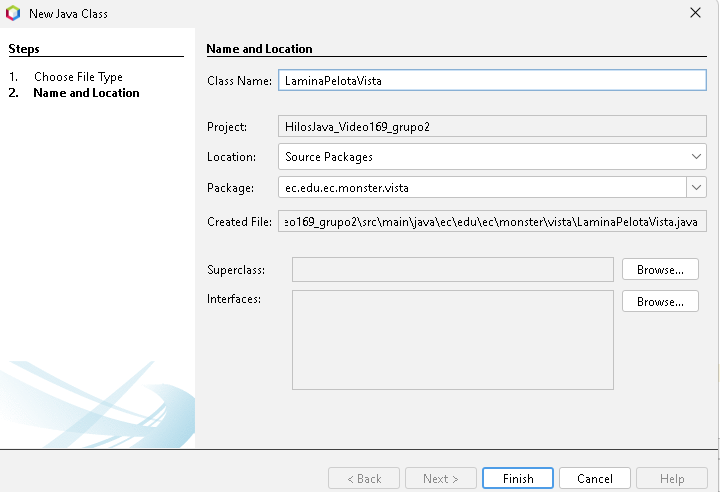


Figura 33. Creación de clase Java.

Le da el nombre de “LaminaPelotaVista”, y proceda a colocar el siguiente código para construir la segunda clase en el paquete de vista.

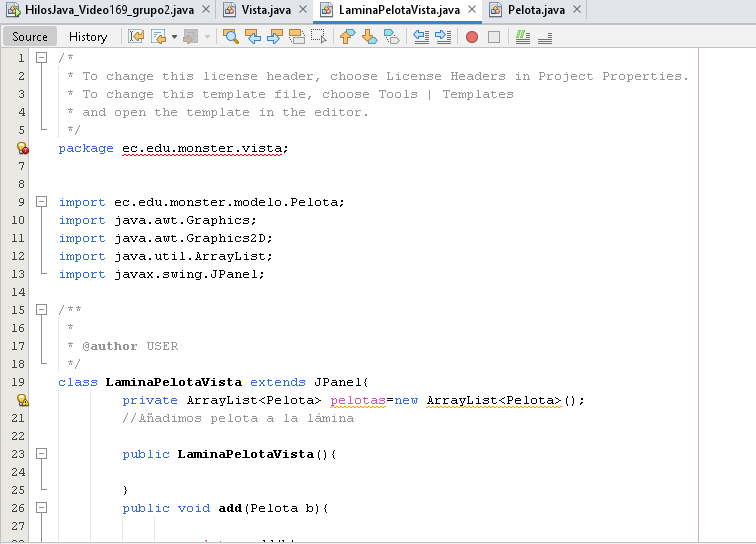


Figura 34. Codificación archivo Lamina Pelota Vista.

Tabla 9. Codificación del archivo Lamina Pelota Vista

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Pelota;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.util.ArrayList;

import javax.swing.JPanel;

/\*\*

\*

\* @author USER

\*/

class LaminaPelotaVista extends JPanel{

private ArrayList<Pelota> pelotas=new ArrayList<Pelota>();

//Añadimos pelota a la lámina

public LaminaPelotaVista(){

}

public void add(Pelota b){

pelotas.add(b);

}

public void paintComponent(Graphics g){

super.paintComponent(g);

Graphics2D g2=(Graphics2D)g;

for(Pelota b: pelotas){

g2.fill(b.getShape());

}

}

}

public void paintComponent(Graphics g){

super.paintComponent(g);

Graphics2D g2=(Graphics2D)g;

for(Pelota b: pelotas){

g2.fill(b.getShape());

}

}

}

}}

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

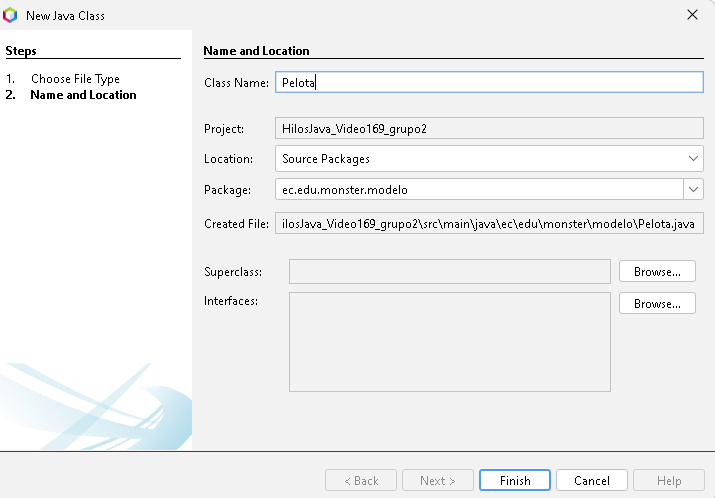


Figura 35. Creación de clase java.

Le da el nombre de “Pelota”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

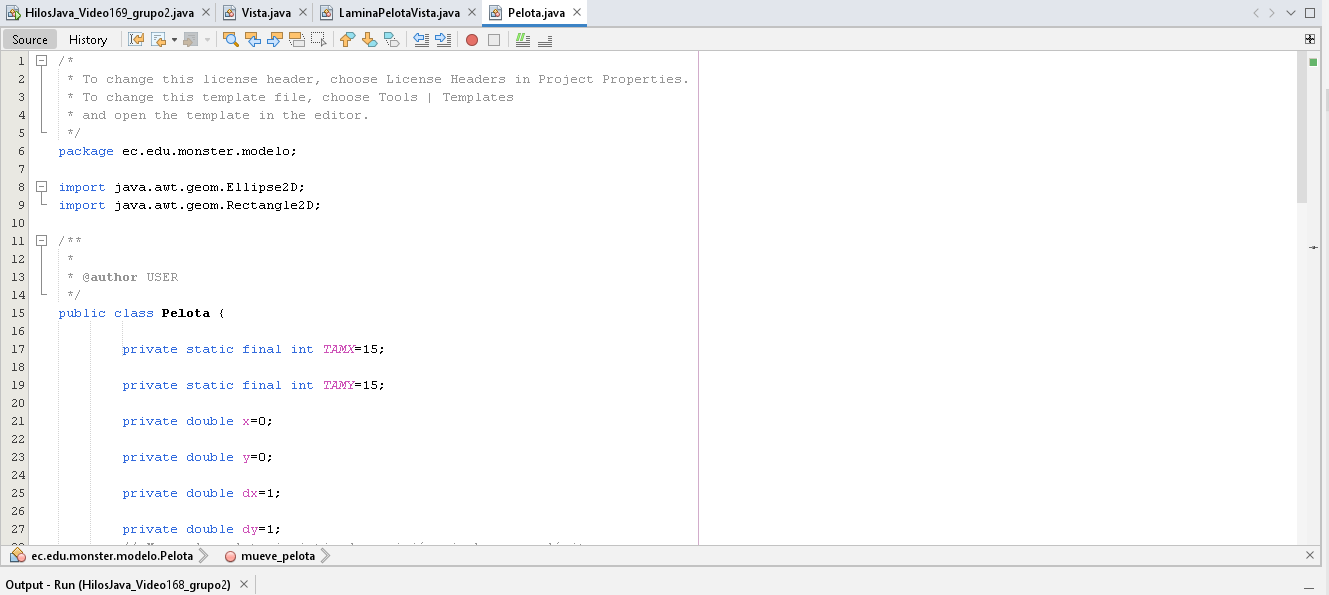


Figura 36. Codificación archivo Pelota.

Tabla 10. Codificación del archivo Pelota

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.modelo;

import java.awt.geom.Ellipse2D;

import java.awt.geom.Rectangle2D;

/\*\*

\*

\* @author USER

\*/

public class Pelota {

private static final int TAMX=15;

private static final int TAMY=15;

private double x=0;

private double y=0;

private double dx=1;

private double dy=1;

// Mueve la pelota invirtiendo posición si choca con límites

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

} <%= lenguajesSeleccionados[i]%>

</li>

<%}

}

%>

</ul>

</body>

</html>

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

}

Cree una nueva clase en el paquete del modelo.

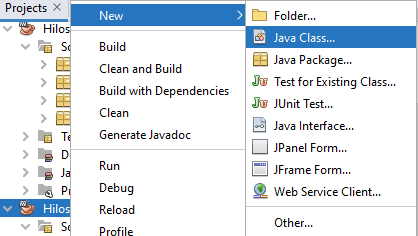


Figura 37. Creación de clase Java.

Le da el nombre de “PelotaHilos”, y proceda a colocar el siguiente código para construir su la segunda clase en el paquete de modelo.

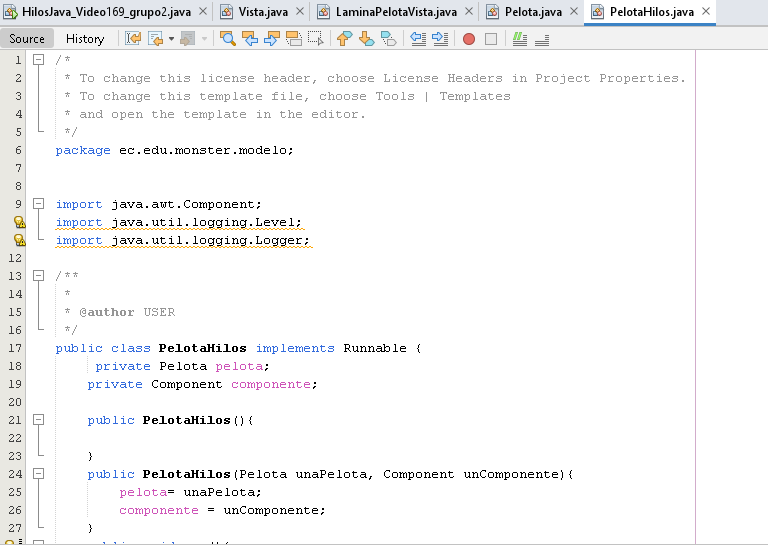


Figura 38. Codificación archivo Pelota Hilos.

*Tabla 11.* Codificación del archivo Pelota Hilos

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.modelo;

import java.awt.Component;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author USER

\*/

public class PelotaHilos implements Runnable {

private Pelota pelota;

private Component componente;

public PelotaHilos(){

}

public PelotaHilos(Pelota unaPelota, Component unComponente){

pelota= unaPelota;

componente = unComponente;

}

public void run(){

while (!Thread.currentThread().isInterrupted()){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

/\*for (int i=1; i<=3000; i++){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

/\*try {

Thread.sleep(4);

} catch (InterruptedException ex) {

//Logger.getLogger(MarcoRebote.class.getName()).log(Level.SEVERE, null, ex);

}

}\*/

}

}

}

*aaaa*

while (!Thread.currentThread().isInterrupted()){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

/\*for (int i=1; i<=3000; i++){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

/\*try {

Thread.sleep(4);

} catch (InterruptedException ex) {

//Logger.getLogger(MarcoRebote.class.getName()).log(Level.SEVERE, null, ex);

}

}\*/

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

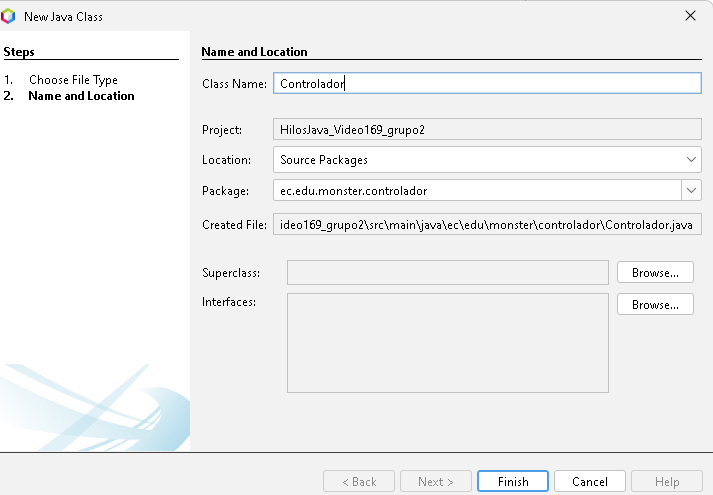


Figura 39. Creación de clase Java.

Le da el nombre de “Controlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

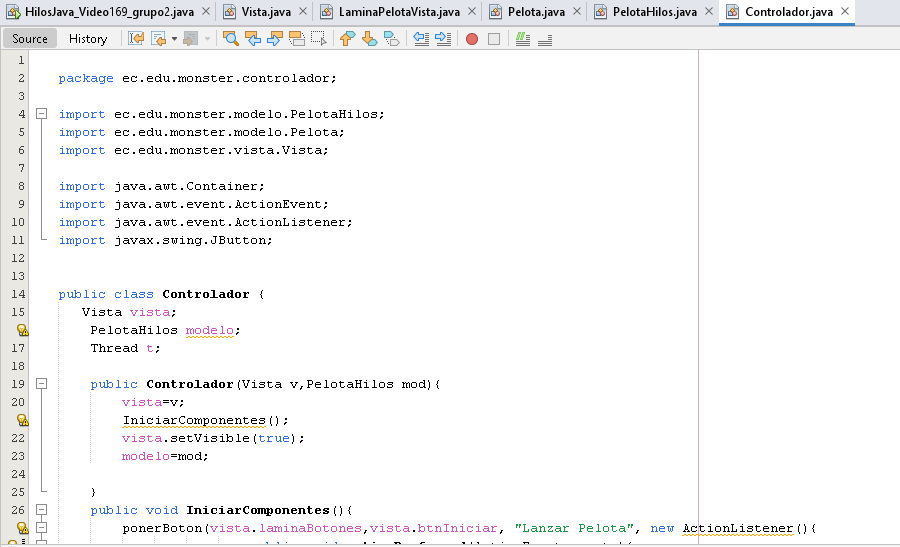


Figura 40. Codificación archivo controlador.

Tabla 12. Codificación del archivo Controlador

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.PelotaHilos;

import ec.edu.monster.modelo.Pelota;

import ec.edu.monster.vista.Vista;

import java.awt.Container;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

public class Controlador {

Vista vista;

PelotaHilos modelo;

Thread t;

public Controlador(Vista v,PelotaHilos mod){

vista=v;

IniciarComponentes();

vista.setVisible(true);

modelo=mod;

}

public void IniciarComponentes(){

ponerBoton(vista.laminaBotones,vista.btnIniciar, "Lanzar Pelota", new ActionListener(){

public void actionPerformed(ActionEvent evento){

LanzarPelota();

}

});

ponerBoton(vista.laminaBotones,vista.btnSalir, "Salir", new ActionListener(){

public void actionPerformed(ActionEvent evento){

System.exit(0);

}

});

ponerBoton(vista.laminaBotones,vista.btnDetener, "Parar pelota", new ActionListener(){

public void actionPerformed(ActionEvent evento){

detener();

}

});

}

public void ponerBoton(Container c,JButton btn, String titulo, ActionListener oyente){

btn.setText(titulo);

c.add(btn);

btn.addActionListener(oyente);

}

public void LanzarPelota (){

Pelota pelota=new Pelota();

vista.AñadirPelota(pelota);

Runnable r = new PelotaHilos(pelota, vista.laminaPelota);

t = new Thread(r);

t.start();

}

public void detener(){

//t.stop();

t.interrupt();

}

}

}

});

}

public void ponerBoton(Container c,JButton btn, String titulo, ActionListener accion){

btn.setText(titulo);

c.add(btn);

btn.addActionListener(accion);

}

public void LanzarPelota(){

Pelota pelota=new Pelota();

vista.AñadirPelota(pelota);

Runnable r = new PelotaHilos(pelota, vista.laminaPelota);

Thread t = new Thread(r);

t.start();

}

}

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

} <%= lenguajesSeleccionados[i]%>

</li>

<%}

}

%>

</ul>

</body>

</html>

LanzarPelota();

}

});

ponerBoton(vista.laminaBotones,vista.btnSalir, "Salir", new ActionListener(){

public void actionPerformed(ActionEvent evento){

System.exit(0);

}

});

ponerBoton(vista.laminaBotones,vista.btnDetener, "Parar pelota", new ActionListener(){

public void actionPerformed(ActionEvent evento){

detener();

}

});

}

public void ponerBoton(Container c,JButton btn, String titulo, ActionListener oyente){

btn.setText(titulo);

c.add(btn);

btn.addActionListener(oyente);

}

public void LanzarPelota (){

Pelota pelota=new Pelota();

vista.AñadirPelota(pelota);

Runnable r = new PelotaHilos(pelota, vista.laminaPelota);

t = new Thread(r);

t.start();

}

public void detener(){

t.interrupt(); }

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código.

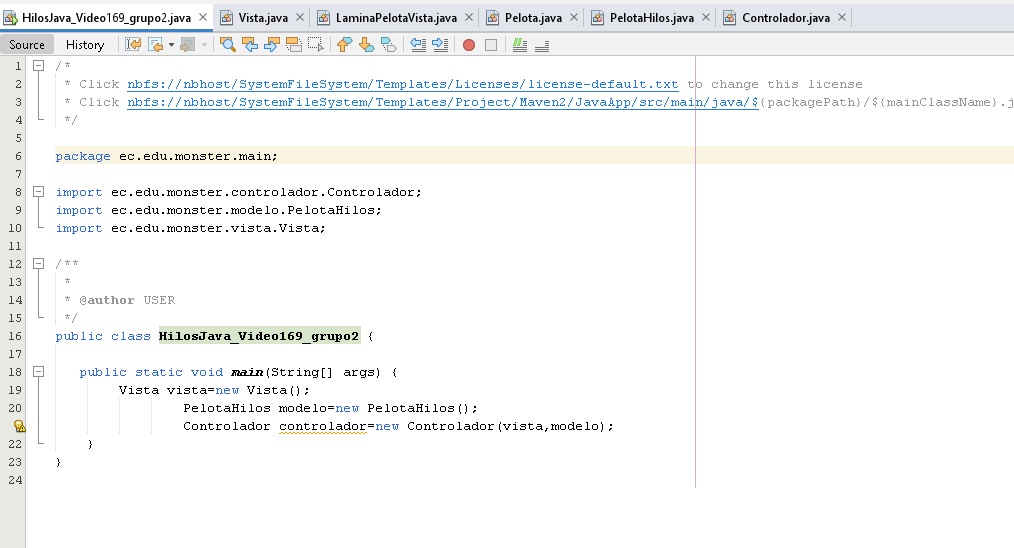


Figura 41. Codificación archivo principal del video 169.

Tabla 13. Codificación del archivo principal del proyecto.

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Project/Maven2/JavaApp/src/main/java/${packagePath}/${mainClassName}.java to edit this template

\*/

package ec.edu.monster.main;

import ec.edu.monster.controlador.Controlador;

import ec.edu.monster.modelo.PelotaHilos;

import ec.edu.monster.vista.Vista;

/\*\*

\*

\* @author USER

\*/

public class HilosJava\_Video169\_grupo6 {

public static void main(String[] args) {

Vista vista=new Vista();

PelotaHilos modelo=new PelotaHilos();

Controlador controlador=new Controlador(vista,modelo);

}

}

### EJECUCIÓN DEL PROYECTO.

Ejecute el proyecto y podrá observar como puede lanzar una pelota, además de tener la capacidad de detenerla según se desee, esto se logra con las modificaciones realizadas en comparación al proyecto anterior.

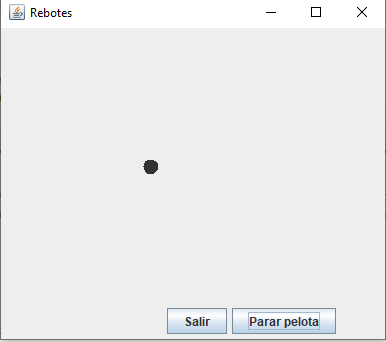


Figura 42. Resultado del video 169.

## Video 170

### 2.3.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.



Figura 43. Logotipo identificativo de Apache NetBeans

Abra Apache Netbeans y se sitúa en la Pestaña superior y selecciona File->New Project -> Java with Mave -> Web Application

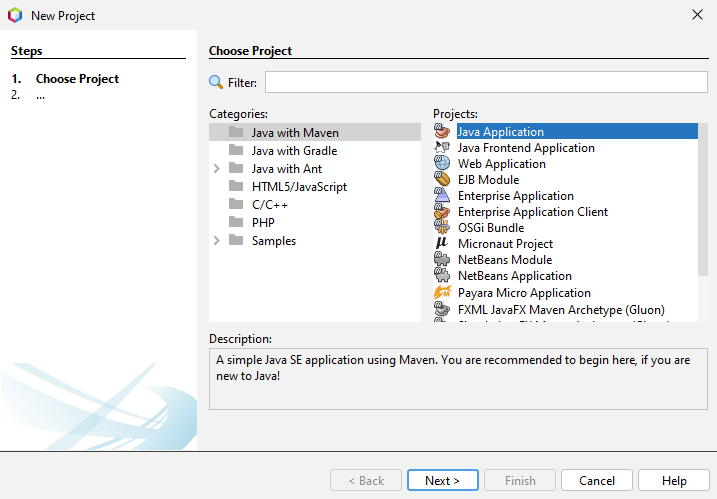


Figura 42. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre HilosJava\_Video170\_grupo#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “aplicativo” que fue creada con anterioridad, en el grupo Id “ec.edu.monster.main” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

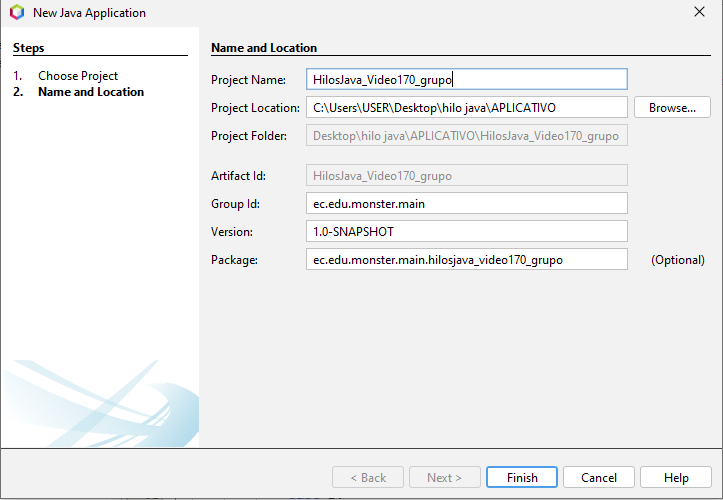


Figura 43. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

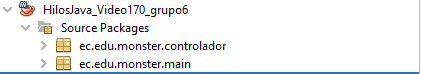


Figura 44. Estructura del proyecto generado.

Si desea para mas orden renombre el paquete creado inicialmente (ec.edu.monster.main.HilosJava\_video169\_grupo#) por “ec.edu.monster.main” Seleccione el paquete, luego->refactor->Rename.



Figura 45. Eliminación del archivo index.html del proyecto.

Abra la carpeta Project Files y selecciones el archivo pom.xml y en la etiqueta <group Id> coloque “ec.edu.monster.main”, mientra que en la etiqueta <exec.mainClass> “ec.edu.monster.main.HilosJava\_Video168\_grupo#”.

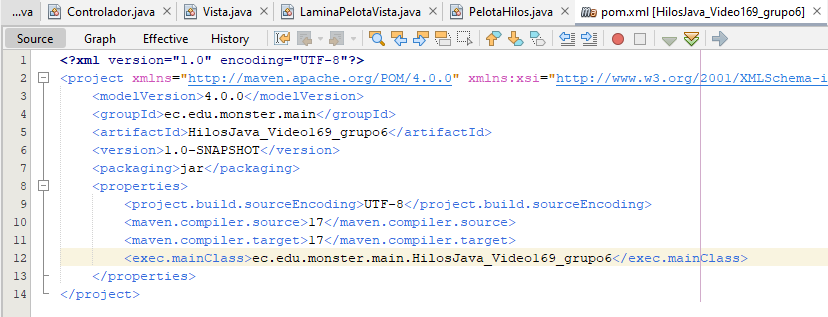


Figura 46. Modificación del archivo pom.xml video 170.

### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

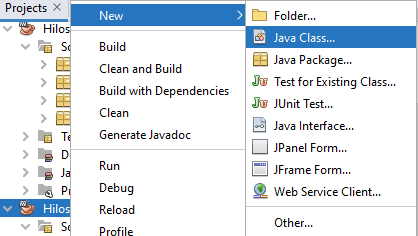


Figura 47. Creación de clase Java.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

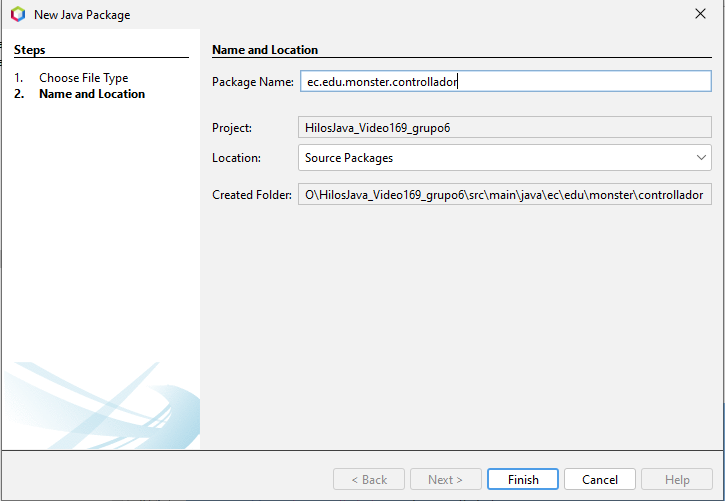


Figura 48. Creación de paquete 170.

Repita el proceso hasta tener los 3 paquetes creados

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

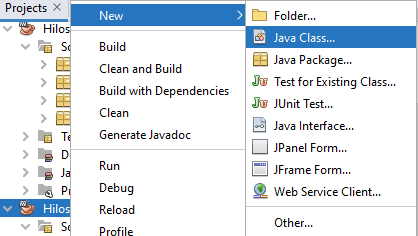


Figura 50. Creación de clase Java.

Le da el nombre de “Vista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

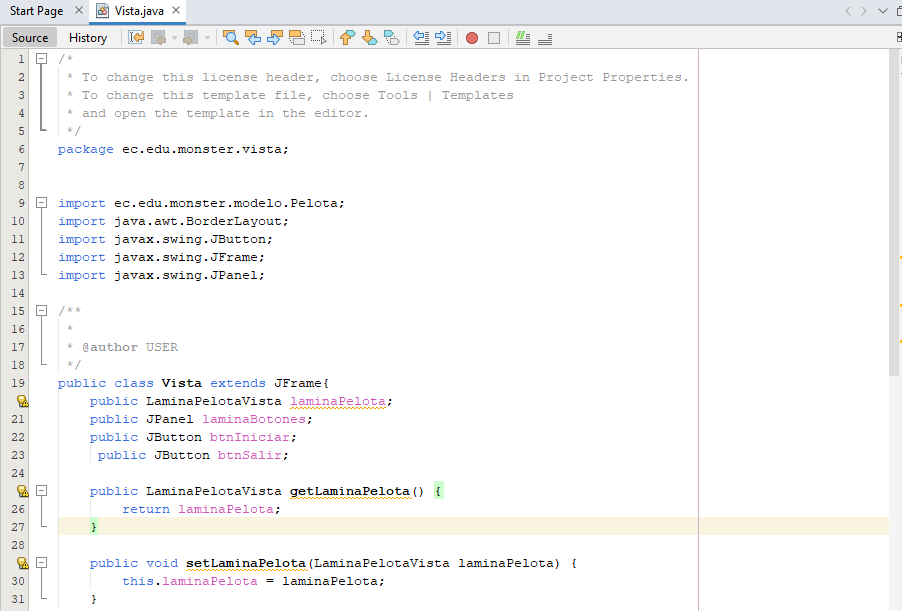


Figura 51. Codificación archivo Vista.

Tabla 14. Codificación del archivo Vista

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Pelota;

import java.awt.BorderLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\*

\* @author USER

\*/

public class Vista extends JFrame{

public LaminaPelotaVista laminaPelota;

public JPanel laminaBotones;

public JButton btnIniciar;

public JButton btnSalir;

public JButton btnDetener;

public LaminaPelotaVista getLaminaPelota() {

return laminaPelota;

}

public void setLaminaPelota(LaminaPelotaVista laminaPelota) {

this.laminaPelota = laminaPelota;

}

public Vista(){

setBounds(600,300,400,350);

setTitle ("Rebotes");

laminaBotones=new JPanel();

this.laminaPelota=new LaminaPelotaVista();

add(this.laminaPelota, BorderLayout.CENTER);

btnIniciar=new JButton();

btnSalir=new JButton();

btnDetener =new JButton();

add(laminaBotones, BorderLayout.SOUTH);

}

public void AñadirPelota(Pelota m){

laminaPelota.add(m);

}

}

public LaminaPelotaVista getLaminaPelota() {

return laminaPelota;

}

public void setLaminaPelota(LaminaPelotaVista laminaPelota) {

this.laminaPelota = laminaPelota;

}

public Vista(){

setBounds(600,300,400,350);

setTitle ("Rebotes");

laminaBotones=new JPanel();

this.laminaPelota=new LaminaPelotaVista();

add(this.laminaPelota, BorderLayout.CENTER);

btnIniciar=new JButton();

btnSalir=new JButton();

btnDetener =new JButton();

add(laminaBotones, BorderLayout.SOUTH);

}

public void AñadirPelota(Pelota m){

laminaPelota.add(m);

}

}

Cree otra clase Java.

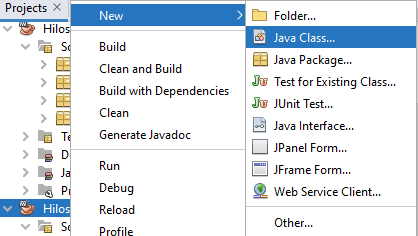


Figura 52. Creación de clase Java.

Le da el nombre de “LamilaPelotaVista”, y proceda a colocar el siguiente código para construir la segunda clase en el paquete de vista.

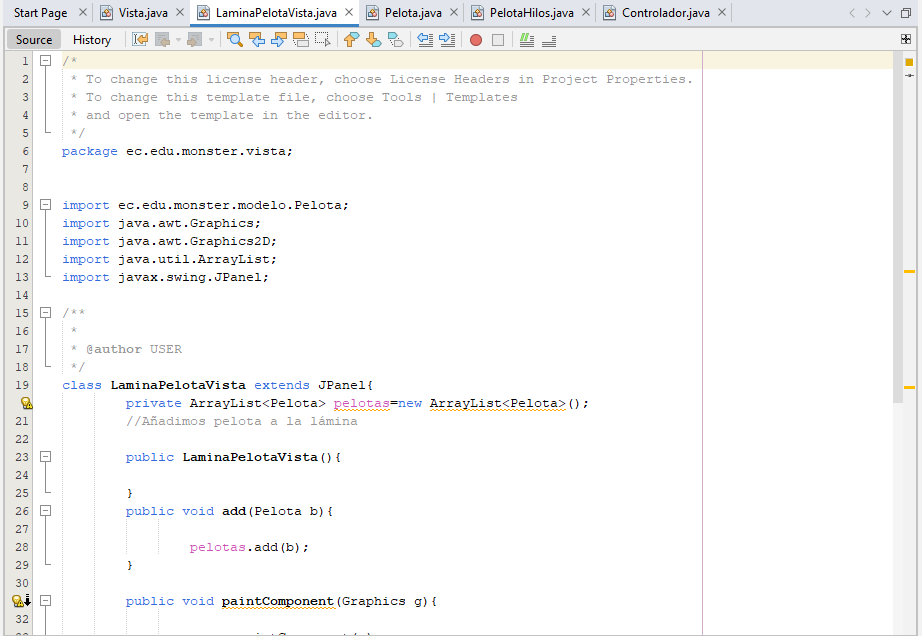


Figura 53. Codificación archivo Lamina Pelota Vista.

Tabla 15. Codificación del archivo Lamina Pelota Hilos.

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Pelota;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.util.ArrayList;

import javax.swing.JPanel;

/\*\*

\*

\* @author USER

\*/

class LaminaPelotaVista extends JPanel{

private ArrayList<Pelota> pelotas=new ArrayList<Pelota>();

//Añadimos pelota a la lámina

public LaminaPelotaVista(){

}

public void add(Pelota b){

pelotas.add(b);

}

public void paintComponent(Graphics g){

super.paintComponent(g);

Graphics2D g2=(Graphics2D)g;

for(Pelota b: pelotas){

g2.fill(b.getShape());

}

}

}

public void paintComponent(Graphics g){

super.paintComponent(g);

Graphics2D g2=(Graphics2D)g;

for(Pelota b: pelotas){

g2.fill(b.getShape());

}

}

}

}}

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

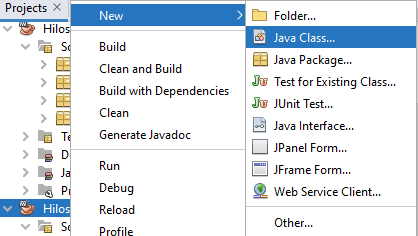


Figura 58. Creación de clase Java.

Le da el nombre de “Pelota”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

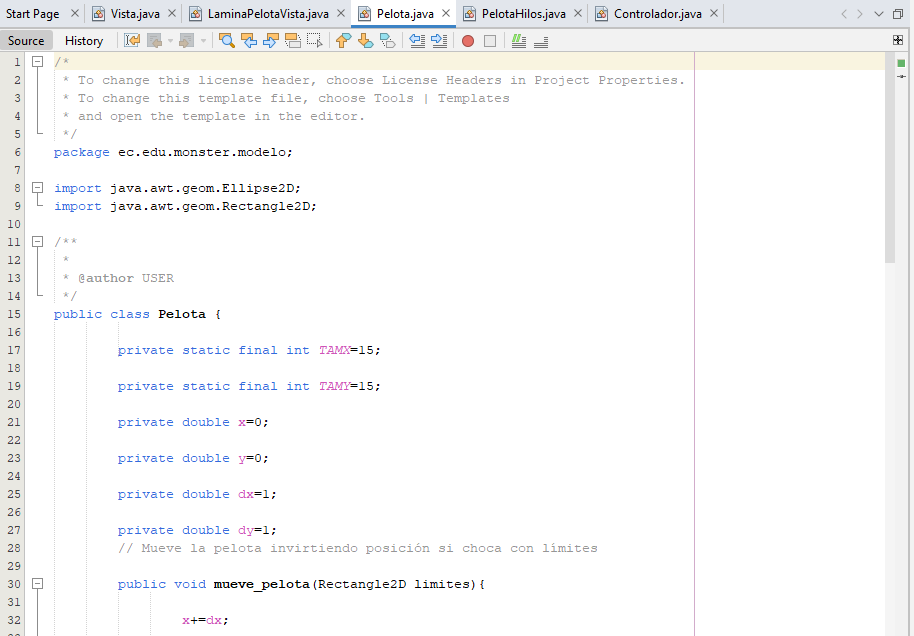


Figura 59. Codificación archivo Pelota.

Tabla 16. Codificación del archivo Pelota

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.modelo;

import java.awt.geom.Ellipse2D;

import java.awt.geom.Rectangle2D;

/\*\*

\*

\* @author USER

\*/

public class Pelota {

private static final int TAMX=15;

private static final int TAMY=15;

private double x=0;

private double y=0;

private double dx=1;

private double dy=1;

// Mueve la pelota invirtiendo posición si choca con límites

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

} <%= lenguajesSeleccionados[i]%>

</li>

<%}

}

%>

</ul>

</body>

</html>

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

}

Cree una nueva clase en el paquete del modelo.

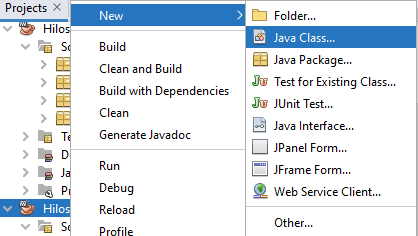


Figura 60. Creación de clase Java.

Le da el nombre de “PelotaHilos”, y proceda a colocar el siguiente código para construir su la segunda clase en el paquete de modelo.

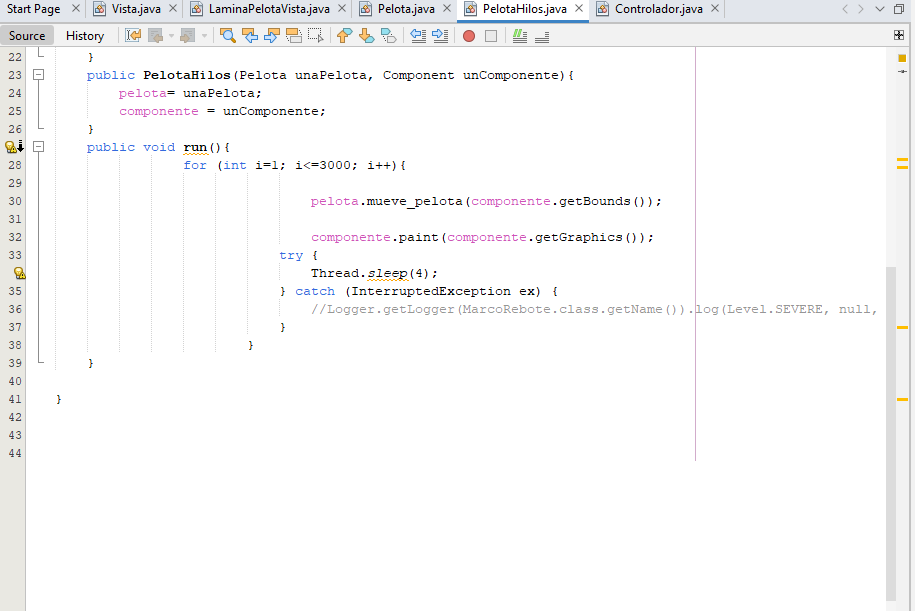


Figura 61. Codificación archivo Pelota Hilos.

Tabla 17. Codificación del archivo PelotaHilos

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package ec.edu.monster.modelo;

import java.awt.Component;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author USER

\*/

public class PelotaHilos implements Runnable {

private Pelota pelota;

private Component componente;

public PelotaHilos(){

}

public PelotaHilos(Pelota unaPelota, Component unComponente){

pelota= unaPelota;

componente = unComponente;

}

public void run(){

while (!Thread.currentThread().isInterrupted()){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

/\*for (int i=1; i<=3000; i++){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

/\*try {

Thread.sleep(4);

} catch (InterruptedException ex) {

//Logger.getLogger(MarcoRebote.class.getName()).log(Level.SEVERE, null, ex);

}

}\*/

}

}

}

while (!Thread.currentThread().isInterrupted()){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

/\*for (int i=1; i<=3000; i++){

pelota.mueve\_pelota(componente.getBounds());

componente.paint(componente.getGraphics());

/\*try {

Thread.sleep(4);

} catch (InterruptedException ex) {

//Logger.getLogger(MarcoRebote.class.getName()).log(Level.SEVERE, null, ex);

}

}\*/

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

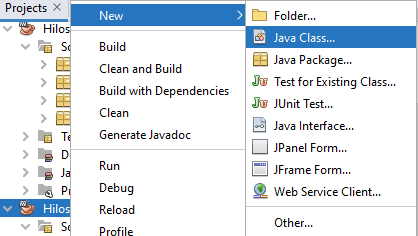


Figura 62. Creación de clase Java.

Le da el nombre de “Controlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

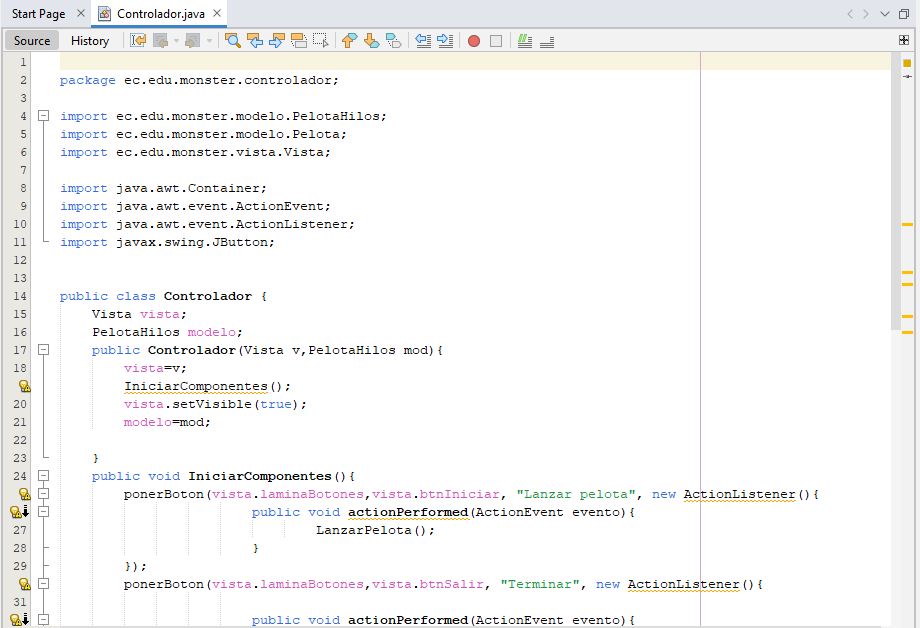


Figura 63. Codificación archivo controlador.

Tabla 18. Codificación del archivo Controlador

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.PelotaHilos;

import ec.edu.monster.modelo.Pelota;

import ec.edu.monster.vista.Vista;

import java.awt.Container;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

public class Controlador {

Vista vista;

PelotaHilos modelo;

Thread t;

public Controlador(Vista v,PelotaHilos mod){

vista=v;

IniciarComponentes();

vista.setVisible(true);

modelo=mod;

}

public void IniciarComponentes(){

ponerBoton(vista.laminaBotones,vista.btnIniciar, "Lanzar Pelota", new ActionListener(){

public void actionPerformed(ActionEvent evento){

LanzarPelota();

}

});

ponerBoton(vista.laminaBotones,vista.btnSalir, "Salir", new ActionListener(){

public void actionPerformed(ActionEvent evento){

System.exit(0);

}

});

ponerBoton(vista.laminaBotones,vista.btnDetener, "Parar pelota", new ActionListener(){

public void actionPerformed(ActionEvent evento){

detener();

}

});

}

public void ponerBoton(Container c,JButton btn, String titulo, ActionListener oyente){

btn.setText(titulo);

c.add(btn);

btn.addActionListener(oyente);

}

public void LanzarPelota (){

Pelota pelota=new Pelota();

vista.AñadirPelota(pelota);

Runnable r = new PelotaHilos(pelota, vista.laminaPelota);

t = new Thread(r);

t.start();

}

public void detener(){

//t.stop();

t.interrupt();

}

}

}

});

}

public void ponerBoton(Container c,JButton btn, String titulo, ActionListener accion){

btn.setText(titulo);

c.add(btn);

btn.addActionListener(accion);

}

public void LanzarPelota(){

Pelota pelota=new Pelota();

vista.AñadirPelota(pelota);

Runnable r = new PelotaHilos(pelota, vista.laminaPelota);

Thread t = new Thread(r);

t.start();

}

}

public void mueve\_pelota(Rectangle2D limites){

x+=dx;

y+=dy;

if(x<limites.getMinX()){

x=limites.getMinX();

dx=-dx;

}

if(x + TAMX>=limites.getMaxX()){

x=limites.getMaxX() - TAMX;

dx=-dx;

}

if(y<limites.getMinY()){

y=limites.getMinY();

dy=-dy;

}

if(y + TAMY>=limites.getMaxY()){

y=limites.getMaxY()-TAMY;

dy=-dy;

}

}

//Forma de la pelota en su posición inicial

public Ellipse2D getShape(){

return new Ellipse2D.Double(x,y,TAMX,TAMY);

}

} <%= lenguajesSeleccionados[i]%>

</li>

<%}

}

%>

</ul>

</body>

</html>

LanzarPelota();

}

});

ponerBoton(vista.laminaBotones,vista.btnSalir, "Salir", new ActionListener(){

public void actionPerformed(ActionEvent evento){

System.exit(0);

}

});

ponerBoton(vista.laminaBotones,vista.btnDetener, "Parar pelota", new ActionListener(){

public void actionPerformed(ActionEvent evento){

detener();

}

});

}

public void ponerBoton(Container c,JButton btn, String titulo, ActionListener oyente){

btn.setText(titulo);

c.add(btn);

btn.addActionListener(oyente);

}

public void LanzarPelota (){

Pelota pelota=new Pelota();

vista.AñadirPelota(pelota);

Runnable r = new PelotaHilos(pelota, vista.laminaPelota);

t = new Thread(r);

t.start();

}

public void detener(){

t.interrupt(); }

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código.

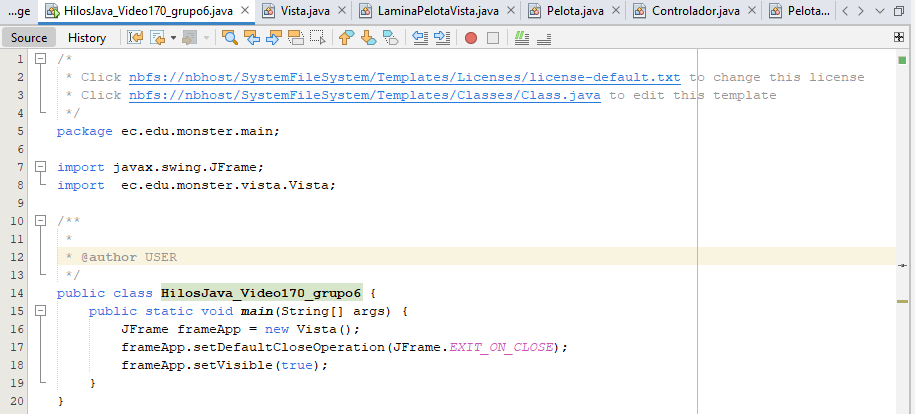


Figura 64. Codificación archivo principal del proyecto 170.

Tabla 19. Codificación del archivo Principal.

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Project/Maven2/JavaApp/src/main/java/${packagePath}/${mainClassName}.java to edit this template

\*/

package ec.edu.monster.main;

import ec.edu.monster.controlador.Controlador;

import ec.edu.monster.modelo.PelotaHilos;

import ec.edu.monster.vista.Vista;

/\*\*

\*

\* @author USER

\*/

public class HilosJava\_Video169\_grupo6 {

public static void main(String[] args) {

Vista vista=new Vista();

PelotaHilos modelo=new PelotaHilos();

Controlador controlador=new Controlador(vista,modelo);

}

}

### EJECUCIÓN DEL PROYECTO.

Ejecute el proyecto y podrá observar como puede lanzar varias pelotas, además de tener la capacidad de detener la pelota que se desee, esto se logra con las modificaciones realizadas en comparación al proyecto anterior.

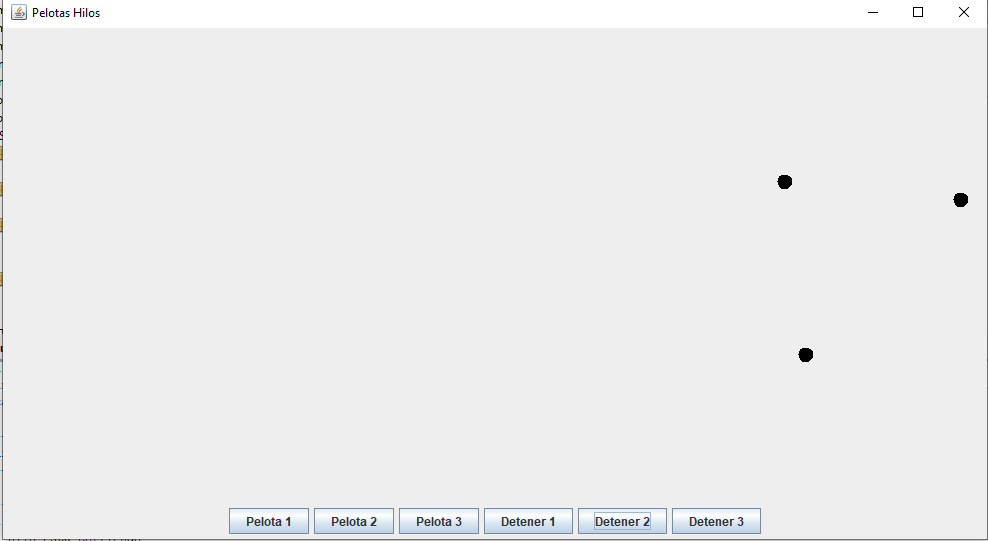


Figura 65. Resultado del proyecto 170.

## Video 171

### 2.1.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.

Imagen que contiene Forma

Descripción generada automáticamente

Figura 2. Logotipo identificativo de Netbeans

Primero dentro del disco C y cree una carpeta llamada DISTRIBUIDAS allí crea otra subcarpeta llamada FORMULARIOSWEB, dentro de esta crea otra subcarpeta con el nombre de HilosJava a su vez dentro de esta última cree 2 carpetas, una con el nombre de “aplicativo” y otra con el nombre de “documentación”. Una vez haya creado las carpetas abre Apache Netbeans y se sitúa en la Pestaña superior y selecciona File->New Project -> Java with Ant -> Java Application

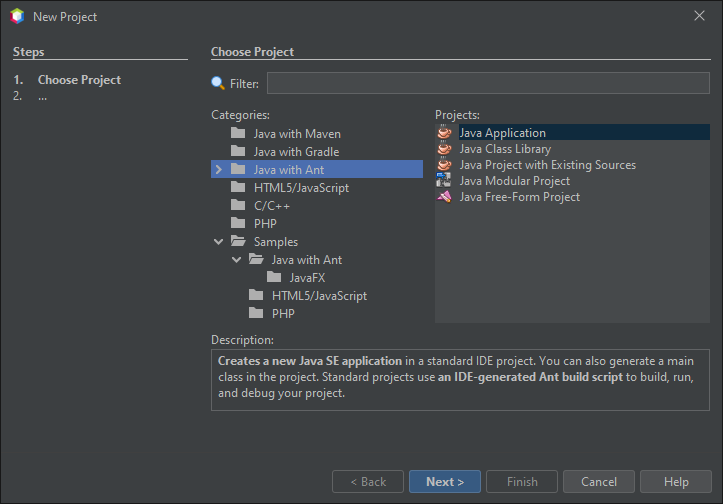


Figura 3. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre G, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “aplicativo” que fue creada con anterioridad, en el grupo Id “ec.edu.monster.main” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

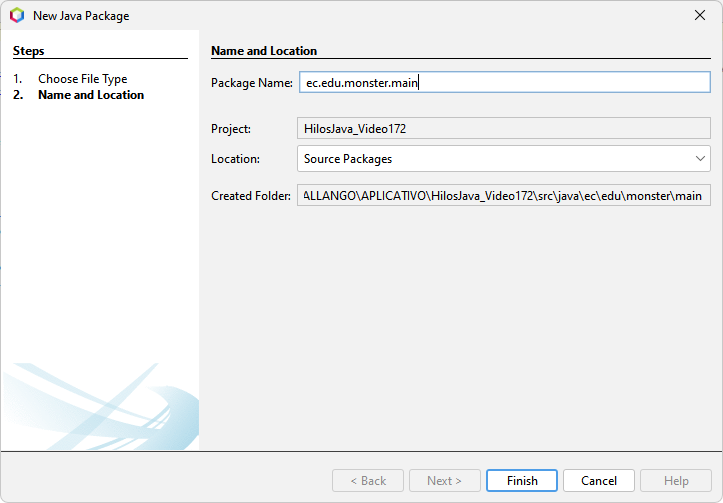


Figura 4. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

Texto

Descripción generada automáticamente

Figura 5. Estructura del proyecto generado

### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 6. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

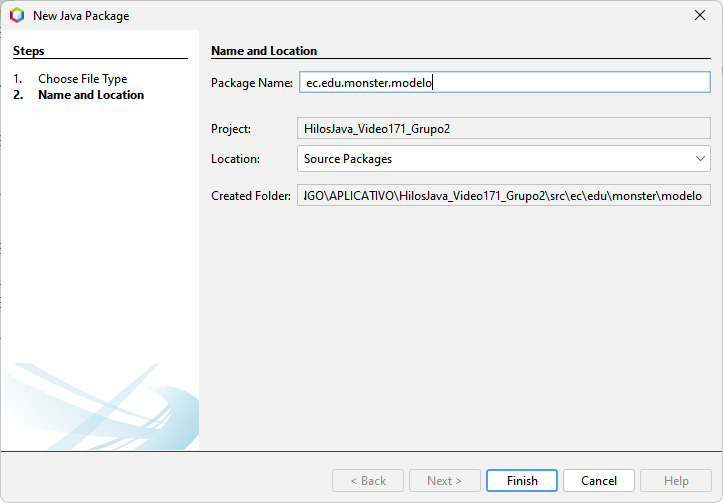


Figura 7. Selección del nombre de los paquetes del video 171

Repita el proceso hasta tener los 3 paquetes creados

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

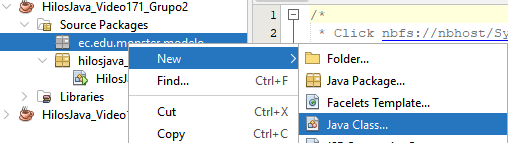


Figura 8. Creación de clase Java.

Le da el nombre de “HilosVista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista, que en este caso solo ocuparemos la consola para mostrar los resultados.

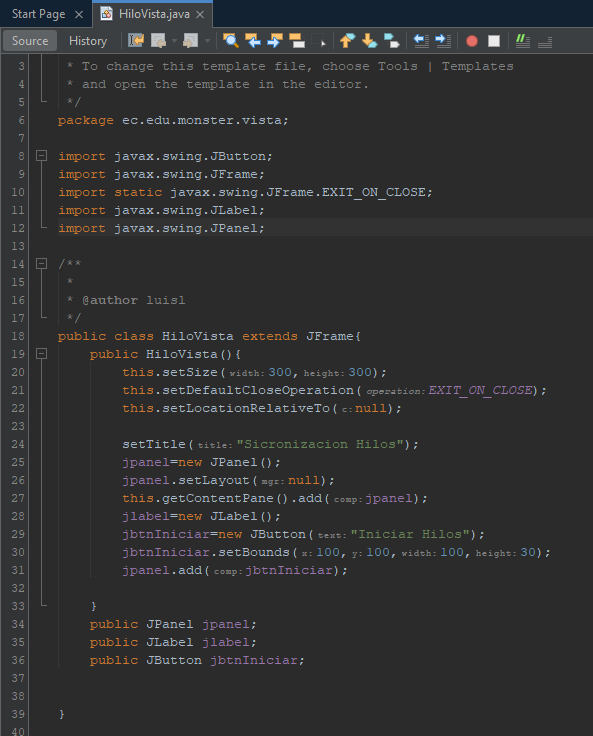


Figura 9. Codificación archivo HilosVista.

.

Tabla 2. Código de HilosVista.

package ec.edu.monster.vista;

import javax.swing.JButton;

import javax.swing.JFrame;

import static javax.swing.JFrame.EXIT\_ON\_CLOSE;

import javax.swing.JLabel;

import javax.swing.JPanel;

public class HiloVista extends JFrame{

public HiloVista(){

this.setSize(300,300);

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

this.setLocationRelativeTo(null);

setTitle("Sicronizacion Hilos");

jpanel=new JPanel();

jpanel.setLayout(null);

this.getContentPane().add(jpanel);

jlabel=new JLabel();

jbtnIniciar=new JButton("Iniciar Hilos");

jbtnIniciar.setBounds(100,100,100,30);

jpanel.add(jbtnIniciar);

}

public JPanel jpanel;

public JLabel jlabel;

public JButton jbtnIniciar;

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

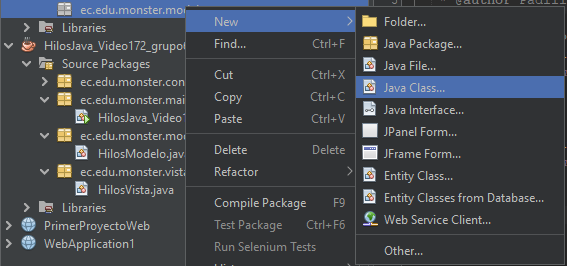


Figura 10. Creación de clase Java.

Le da el nombre de “HilosModelo”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

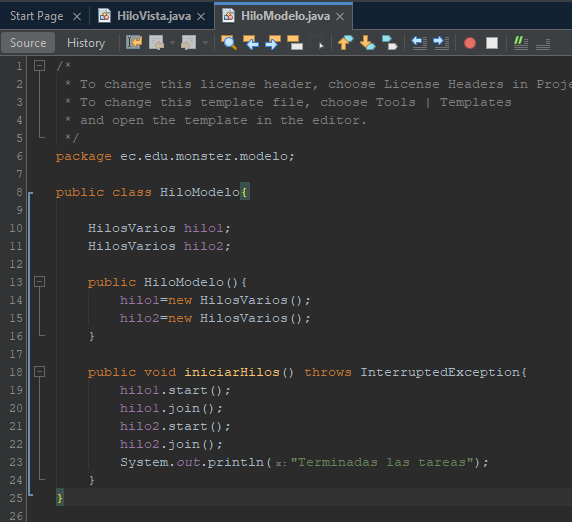


Figura 11. Codificación archivo HilosModelo.

Tabla 3. Codificación del archivo HilosModelo

package ec.edu.monster.modelo;

public class HiloModelo{

HilosVarios hilo1;

HilosVarios hilo2;

public HiloModelo(){

hilo1=new HilosVarios();

hilo2=new HilosVarios();

}

public void iniciarHilos() throws InterruptedException{

hilo1.start();

hilo1.join();

hilo2.start();

hilo2.join();

System.out.println("Terminadas las tareas");

}

}

Cree una nueva clase en el paquete del modelo.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 12. Creación de clase Java.

Le da el nombre de “HilosVarios”, y proceda a colocar el siguiente código para construir la clase en el paquete de controlador.

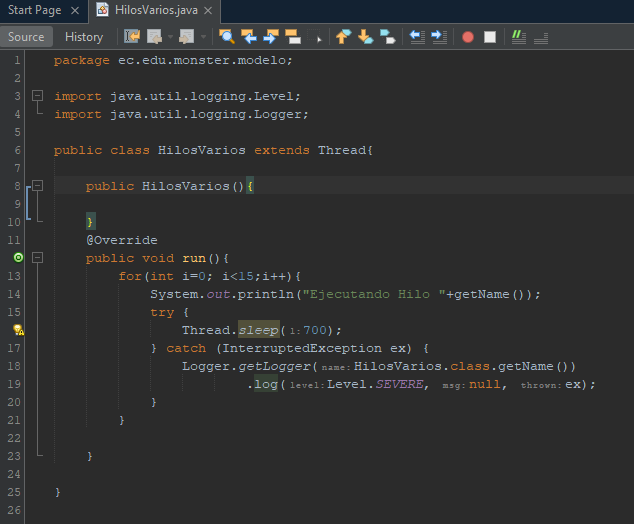


Figura 13. Codificación archivo HilosVarios

Tabla 4. Codificación del archivo HilosVarios

package ec.edu.monster.modelo;

import java.util.logging.Level;

import java.util.logging.Logger;

public class HilosVarios extends Thread{

public HilosVarios(){

}

@Override

public void run(){

for(int i=0; i<15;i++){

System.out.println("Ejecutando Hilo "+getName());

try {

Thread.sleep(700);

} catch (InterruptedException ex) {

Logger.getLogger(HilosVarios.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

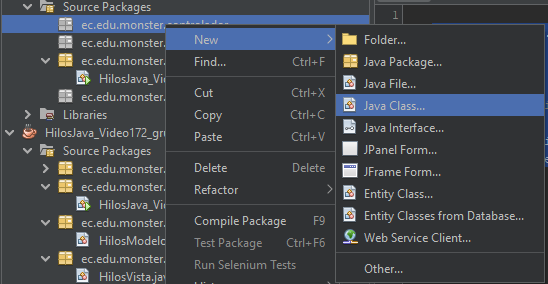


Figura 14. Creación de clase Java.

Le da el nombre de “HilosControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.



Figura 15.Codificación archivo HilosControlador.

Tabla 5. Codificación del archivo HilosControlador

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.HiloModelo;

import ec.edu.monster.vista.HiloVista;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.logging.Level;

import java.util.logging.Logger;

public class HiloControlador implements ActionListener {

HiloVista hiloVista;

HiloModelo hiloModelo;

public HiloControlador(HiloVista vista,HiloModelo modelo){

this.hiloVista=vista;

this.hiloModelo=modelo;

this.hiloVista.jbtnIniciar.addActionListener(this);

iniciarComponentes();

}

public void iniciarComponentes(){

hiloVista.jlabel.setText("Hola");

hiloVista.setVisible(true);

}

@Override

public void actionPerformed(ActionEvent e) {

if (e.getSource() == hiloVista.jbtnIniciar) {

try {

jbtnIniciarHilosActionPerformed();

} catch (InterruptedException ex) {

Logger.getLogger(HiloControlador.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

private void jbtnIniciarHilosActionPerformed() throws InterruptedException{

hiloModelo.iniciarHilos();

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

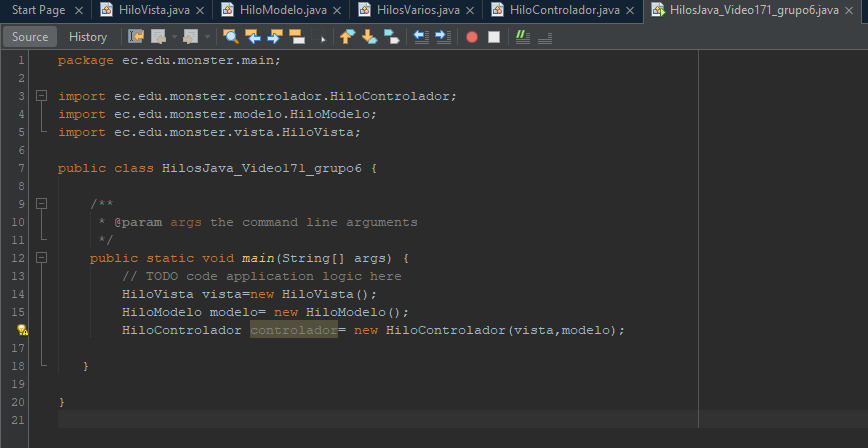


Figura 16. Codificación archivo principal.

Tabla 6.Codificación del archivo principal del proyecto

package ec.edu.monster.main;

import ec.edu.monster.controlador.HiloControlador;

import ec.edu.monster.modelo.HiloModelo;

import ec.edu.monster.vista.HiloVista;

public class HilosJava\_Video171\_grupo2 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

HiloVista vista=new HiloVista();

HiloModelo modelo= new HiloModelo();

HiloControlador controlador= new HiloControlador(vista,modelo);

}

}

### EJECUCIÓN DEL PROYECTO.

Ejecute el proyecto y podrá observar como puede lanzar varias pelotas en diferentes momentos, siendo cada pelota una representación de un hilo.

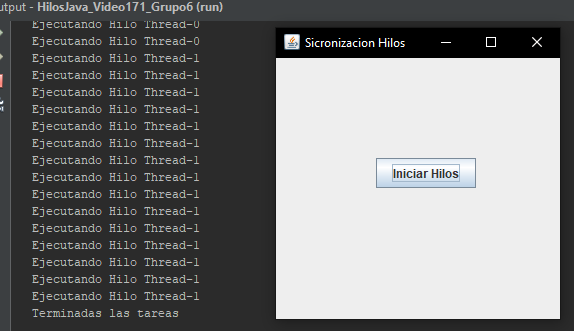


Figura 17. Resultado de la ejecución

## Video 172

### 2.1.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.



Figura 18. Logotipo identificativo de Netbeans

Primero dentro del disco C y cree una carpeta llamada DISTRIBUIDAS allí crea otra subcarpeta llamada FORMULARIOSWEB, dentro de esta crea otra subcarpeta con el nombre de HilosJava a su vez dentro de esta última cree 2 carpetas, una con el nombre de “aplicativo” y otra con el nombre de “documentación”. Una vez haya creado las carpetas abre Apache Netbeans y se sitúa en la Pestaña superior y selecciona File->New Project -> Java Application

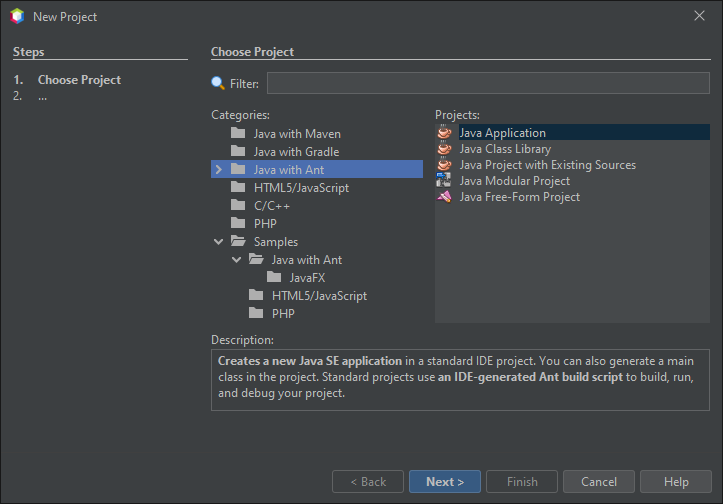


Figura 19. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre G, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “aplicativo” que fue creada con anterioridad, en el grupo Id “ec.edu.monster.main” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

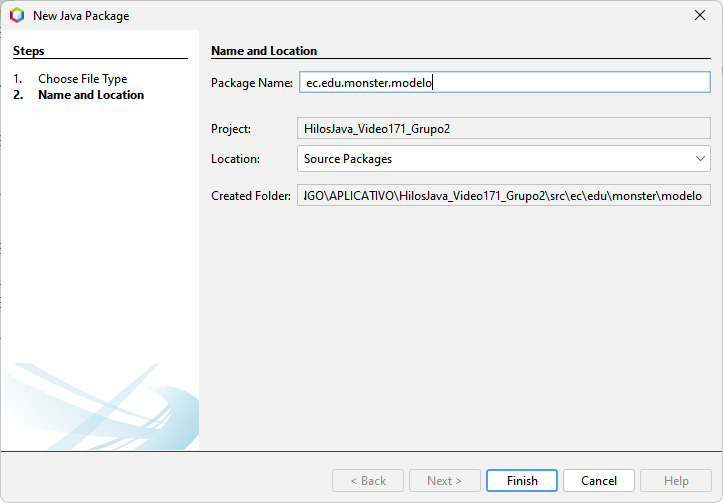


Figura 20. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

Texto

Descripción generada automáticamente

Figura 21. Estructura del proyecto generado.

### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

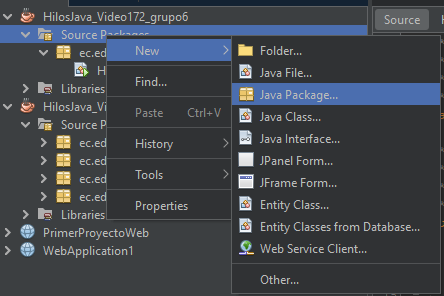


Figura 22. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

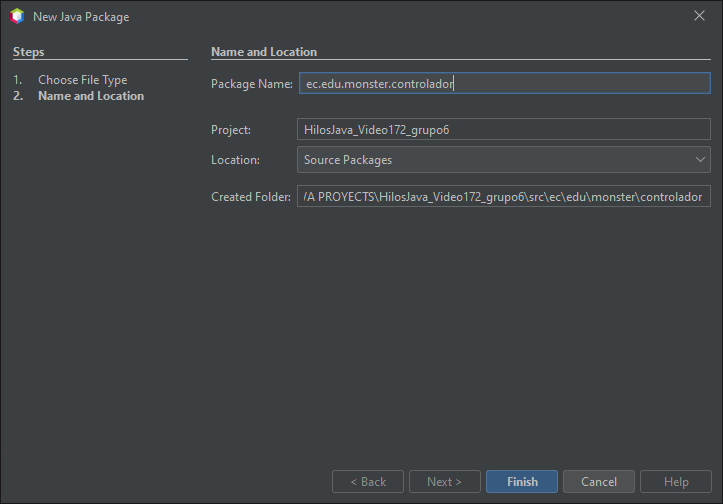


Figura 23. Selección del nombre de los paquetes del video 172

Repita el proceso hasta tener los 3 paquetes creados

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

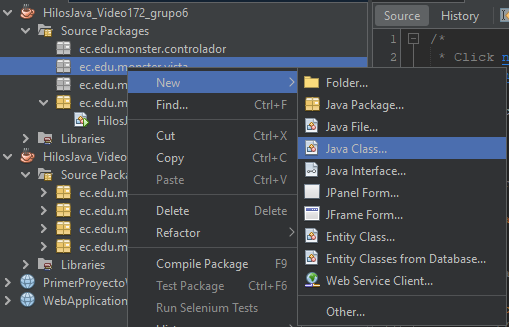


Figura 24. Creación de clase Java

Le da el nombre de “HilosVista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista, que en este caso solo ocuparemos la consola para mostrar los resultados.

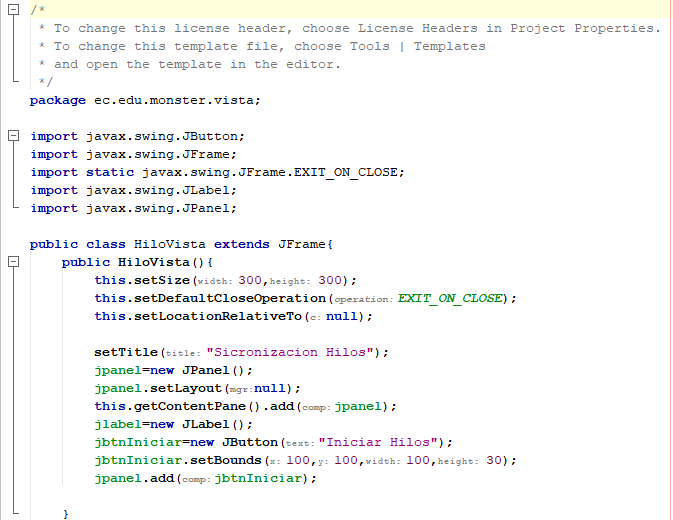


Figura 25. Codificación archivo HilosVista

Tabla 7. Código de HilosVista.

package ec.edu.monster.vista;

/\*\*

\*

\* @author Mosquera – Pallango – Sánchez

\*/

public class HilosVista {

public void imprimirInfoHilo(String nombreHilo) {

System.out.println("Ejecutando hilo..." + nombreHilo);

}

public void imprimirFinalizado() {

System.out.println("Terminadas las tareas...");

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 26. Creación de clase Java

Le da el nombre de “HilosModelo”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 27. Codificación archivo HilosModelo

Tabla 8. Codificación del archivo HilosModelo

package ec.edu.monster.modelo;

/\*\*

\*

\* @author Mosquera – Pallango – Sánchez

\*/

public class HilosModelo {

public Thread hilo1;

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

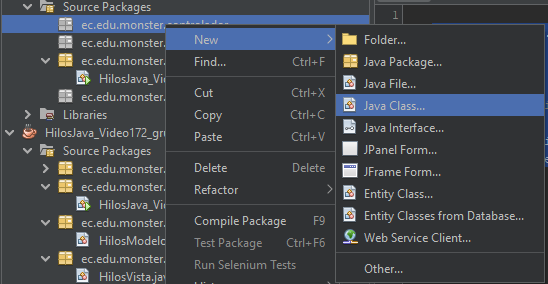


Figura 28. Creación de clase Java.

Le da el nombre de “HilosVariosControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.



Figura 29. Codificación archivo HilosVariosControlador

Tabla 9. Codificación del archivo Controlador

package ec.edu.monster.controlador;

import ec.edu.monster.vista.HilosVista;

public class HilosVariosControlador extends Thread {

public void run() {

HilosVista vista = new HilosVista();

for (int i = 0; i < 15; i++) {

vista.imprimirInfoHilo(getName());

try {

Thread.sleep(700);

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

Cree una nueva clase en el paquete del controlador.

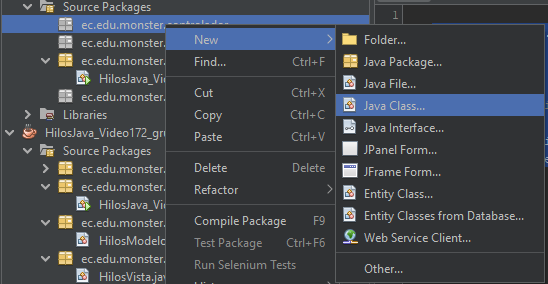


Figura 30. Creación de clase Java.

Le da el nombre de “HilosVarios2Controlador”, y proceda a colocar el siguiente código para construir la clase en el paquete de controlador.

Texto

Descripción generada automáticamente

Figura 31. Codificación archivo HilosVarios2Controlador

Tabla 10. Codificación del archivo HilosVarios2Controlador

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.HilosModelo;

import ec.edu.monster.vista.HilosVista;

public class HilosVarios2Controlador extends Thread {

private HilosModelo hiloModel;

public HilosVarios2Controlador(HilosModelo hiloModel) {

this.hiloModel = hiloModel;

}

public void run() {

HilosVista vista = new HilosVista();

try {

hiloModel.hilo1.join();

} catch (Exception e) {

e.printStackTrace();

}

for (int i = 0; i < 15; i++) {

vista.imprimirInfoHilo(getName());

try {

Thread.sleep(700);

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 32. Codificación archivo principal

Tabla 11. Codificación del archivo principal del proyecto

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Project/Maven2/JavaApp/src/main/java/${packagePath}/${mainClassName}.java to edit this template

\*/

package ec.edu.monster.main;

import ec.edu.monster.controlador.Controlador;

import ec.edu.monster.modelo.PelotaHilos;

import ec.edu.monster.vista.Vista;

/\*\*

\*

\* @author USER

\*/

public class HilosJava\_Video168\_grupo2 {

public static void main(String[] args) {

Vista vista=new Vista();

PelotaHilos modelo=new PelotaHilos();

Controlador controlador=new Controlador(vista,modelo);

}

}

### EJECUCIÓN DEL PROYECTO.

Ejecute el proyecto y podrá observar como puede lanzar varias pelotas en diferentes momentos, siendo cada pelota una representación de un hilo.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 33. Resultado de la ejecución.

# PARTE PRÁCTICA

## Video 173

### 2.1.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.



Figura 2. Logotipo identificativo de Netbeans

Primero, dentro del disco C cree una carpeta llamada DISTRIBUIDAS. Allí cree otra subcarpeta llamada HILOSJAVA y dentro de esta cree 2 carpetas, una con el nombre de “APLICATIVO” y otra con el nombre de “DOCUMENTACIÓN”. Con las carpetas creadas se abre Apache Netbeans y se sitúa en la Pestaña superior y donde se selecciona File->New Project -> Java with Ant -> Java Application

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 3. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre HilosJava\_Video173\_Grupo#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “APLICATIVO” que fue creada con anterioridad, se marca la opción Create Main Class y se coloca en el mismo el siguiente nombre “ec.edu.monster.main.HilosJava\_Video173\_Grupo6” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Figura 4. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Figura 5. Estructura del proyecto generado.

### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

Figura 6. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 7. Selección del nombre de los paquetes del video 173

Repita el proceso hasta tener los 3 paquetes creados

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

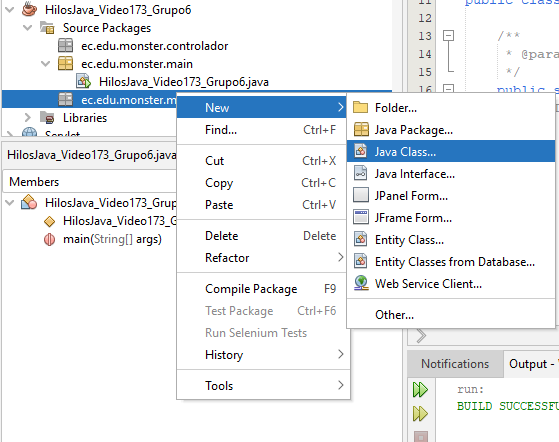


Figura 8. Creación de clase Java.

Le da el nombre de “Banco”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 9. Creación archivo Banco.

Proceda a colocar el siguiente código para construir su clase en el paquete de modelo.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Figura 10. Codificación archivo Banco

Tabla . Codificación del archivo Banco

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.modelo;

/\*\*

\*

\* @author Christian Novoa

\*/

public class Banco {

public double[] cuentas;

public Banco() {

this.cuentas = new double[100];

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 11. Creación de clase Java.

Le da el nombre de “BancoVista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 12. Creación archivo BancoVista.

Coloque el siguiente código para construir su clase en el paquete de vista.

**Texto

Descripción generada automáticamente**

Figura 13. Codificación archivo BancoVista

Tabla . Codificación del archivo BancoVista

package ec.edu.monster.vista;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoVista {

public void mostrarDetallesTransferencia(

int cuentaOrigen,

int cuentaDestino,

double cantidad

) {

System.out.println(Thread.currentThread() + "\n");

System.out.printf("%10.2f de %d para %d\n", cantidad, cuentaOrigen, cuentaDestino);

}

public void mostrarSaldoTotal(

double saldoTotal

) {

System.out.printf("Saldo Total: %10.2f\n", saldoTotal);

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 14. Creación de clase Java.

Le da el nombre de “BancoControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 15. Codificación archivo BancoControlador.

Tabla . Codificación del archivo BancoControlador

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

import ec.edu.monster.vista.BancoVista;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoControlador {

private BancoVista bancoView = new BancoVista();

public void acreditarSaldoCuentas(Banco banco, double saldo) {

double[] cuentasAcreditadas = new double[100];

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

cuentasAcreditadas[i] = saldo;

}

banco.cuentas = cuentasAcreditadas;

}

public void transferirEntreCuentas(Banco banco, int cuentaOrigen, int cuentaDestino, double cantidad) {

if (banco.cuentas[cuentaOrigen] < cantidad) {

return;

}

banco.cuentas[cuentaOrigen] -= cantidad;

bancoView.mostrarDetallesTransferencia(cuentaOrigen, cuentaDestino, cantidad);

banco.cuentas[cuentaDestino] += cantidad;

bancoView.mostrarSaldoTotal(obtenerSaldoTotal(banco));

}

public double obtenerSaldoTotal(Banco banco) {

double saldoTotal = 0;

for (double saldo : banco.cuentas) {

saldoTotal += saldo;

}

return saldoTotal;

}

}

## Video 174

### 2.2.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.



Figura 16. Logotipo identificativo de Netbeans

Abra Apache Netbeans y se sitúa en la Pestaña superior y donde se selecciona File->New Project -> Java with Ant -> Java Application.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 17. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre HilosJava\_Video174\_Grupo#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “APLICATIVO” que fue creada con anterioridad, se marca la opción Create Main Class y se coloca en el mismo el siguiente nombre “ec.edu.monster.main.HilosJava\_Video174\_Grupo6” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Figura 18. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Figura 19. Estructura del proyecto generado.

### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

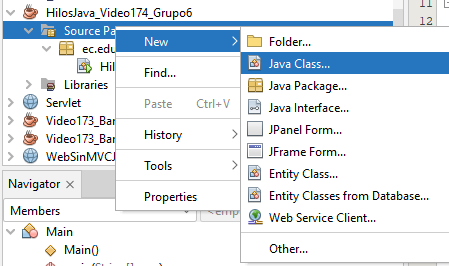


Figura 20. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 21. Selección del nombre de los paquetes del video 174

Repita el proceso hasta tener los 3 paquetes creados.

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 22. Creación de clase Java.

Le da el nombre de “Banco”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 23. Creación archivo Banco.

Proceda a colocar el siguiente código para construir su clase en el paquete de modelo.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Figura 24. Codificación archivo Banco

Tabla . Codificación del archivo Banco

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.modelo;

/\*\*

\*

\* @author Christian Novoa

\*/

public class Banco {

public double[] cuentas;

public Banco() {

this.cuentas = new double[100];

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 25. Creación de clase Java.

Le da el nombre de “BancoVista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 26. Creación archivo BancoVista.

Coloque el siguiente código para construir su clase en el paquete de vista.

**Texto, Correo electrónico

Descripción generada automáticamente**

Figura 27. Codificación archivo BancoVista

Tabla . Codificación del archivo BancoVista

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoVista {

public void mostrarDetallesTransferencia(

int cuentaOrigen,

int cuentaDestino,

double cantidad

) {

System.out.println(Thread.currentThread() + "\n");

System.out.printf("%10.2f de %d para %d\n", cantidad, cuentaOrigen, cuentaDestino);

}

public void mostrarSaldoTotal(

double saldoTotal

) {

System.out.printf("Saldo Total: %10.2f\n", saldoTotal);

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 28. Creación de clase Java.

Le da el nombre de “BancoControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 29. Codificación archivo BancoControlador.

Tabla . Codificación del archivo BancoControlador

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

import ec.edu.monster.vista.BancoVista;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoControlador {

private BancoVista bancoView = new BancoVista();

public void acreditarSaldoCuentas(Banco banco, double saldo) {

double[] cuentasAcreditadas = new double[100];

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

cuentasAcreditadas[i] = saldo;

}

banco.cuentas = cuentasAcreditadas;

}

public void transferirEntreCuentas(Banco banco, int cuentaOrigen, int cuentaDestino, double cantidad) {

if (banco.cuentas[cuentaOrigen] < cantidad) {

return;

}

banco.cuentas[cuentaOrigen] -= cantidad;

bancoView.mostrarDetallesTransferencia(cuentaOrigen, cuentaDestino, cantidad);

banco.cuentas[cuentaDestino] += cantidad;

bancoView.mostrarSaldoTotal(obtenerSaldoTotal(banco));

}

public double obtenerSaldoTotal(Banco banco) {

double saldoTotal = 0;

for (double saldo : banco.cuentas) {

saldoTotal += saldo;

}

return saldoTotal;

}

}

Para la creación del segundo archivo de controlador se da clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 30. Creación de clase Java.

Le da el nombre de “TransferenciasControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 31. Codificación archivo TransferenciasControlador.

Tabla . Codificación del archivo TransferenciasControlador

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class TransferenciasControlador implements Runnable {

private Banco banco;

private BancoControlador bancoControlador;

private int cuentaOrigen;

private double valorMaximo;

public TransferenciasControlador(Banco banco, BancoControlador bancoControlador, int cuentaOrigen, double valorMaximo) {

this.banco = banco;

this.bancoControlador = bancoControlador;

this.cuentaOrigen = cuentaOrigen;

this.valorMaximo = valorMaximo;

}

@Override

public void run() {

while (true) {

int cuentaDestino = (int) (100 \* Math.random());

double cantidad = this.valorMaximo \* Math.random();

this.bancoControlador.transferirEntreCuentas(

this.banco,

this.cuentaOrigen,

cuentaDestino,

cantidad

);

try {

Thread.sleep((int) (Math.random() \* 10));

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 32. Codificación archivo principal del video 174.

Tabla . Codificación del archivo principal del proyecto

package ec.edu.monster.main;

import ec.edu.monster.controlador.BancoControlador;

import ec.edu.monster.controlador.TransferenciasControlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class HilosJava\_Video174\_Grupo6 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Banco banco = new Banco();

BancoControlador bancoControlador = new BancoControlador();

bancoControlador.acreditarSaldoCuentas(banco, 2000);

for (int i = 0; i < 10; i++) {

TransferenciasControlador transferencias

= new TransferenciasControlador(banco, bancoControlador, i, 2000);

new Thread(transferencias).start();

}

}

}

### EJECUCIÓN DEL PROYECTO.

Al ejecutar el proyecto se puede observar por consola como se realizan por distintos hilos las transferencias infinitas desde la cantidad de dinero que pasa, la cuenta desde donde se realiza la transferencia y a la cuenta a la que llega, sin embargo, en ciertas transacciones el saldo total, que siempre debe ser el mismo, se pierde dinero. Esto se debe a que los hilos no están sincronizados y se corregirá con el código del siguiente video.

Tabla

Descripción generada automáticamente con confianza media

Figura 33. Resultado del video 174.

## Video 175

### 2.3.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.

Imagen que contiene Forma

Descripción generada automáticamente

Figura 34. Logotipo identificativo de Netbeans

Abra Apache Netbeans y se sitúa en la Pestaña superior y donde se selecciona File->New Project -> Java with Ant -> Java Application.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 35. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre HilosJava\_Video175\_Grupo#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “APLICATIVO” que fue creada con anterioridad, se marca la opción Create Main Class y se coloca en el mismo el siguiente nombre “ec.edu.monster.main.HilosJava\_Video175\_Grupo6” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Figura 36. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

**Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media**

Figura 37. Estructura del proyecto generado.



### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 38. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 39. Selección del nombre de los paquetes del video 175

Repita el proceso hasta tener los 3 paquetes creados.

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 40. Creación de clase Java.

Le da el nombre de “Banco”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 41. Creación archivo Banco.

Proceda a colocar el siguiente código para construir su clase en el paquete de modelo.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Figura 42. Codificación archivo Banco

Tabla . Codificación del archivo Banco

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.modelo;

/\*\*

\*

\* @author Christian Novoa

\*/

public class Banco {

public double[] cuentas;

public Banco() {

this.cuentas = new double[100];

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 43. Creación de clase Java.

Le da el nombre de “BancoVista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 44. Creación archivo BancoVista.

Coloque el siguiente código para construir su clase en el paquete de vista.

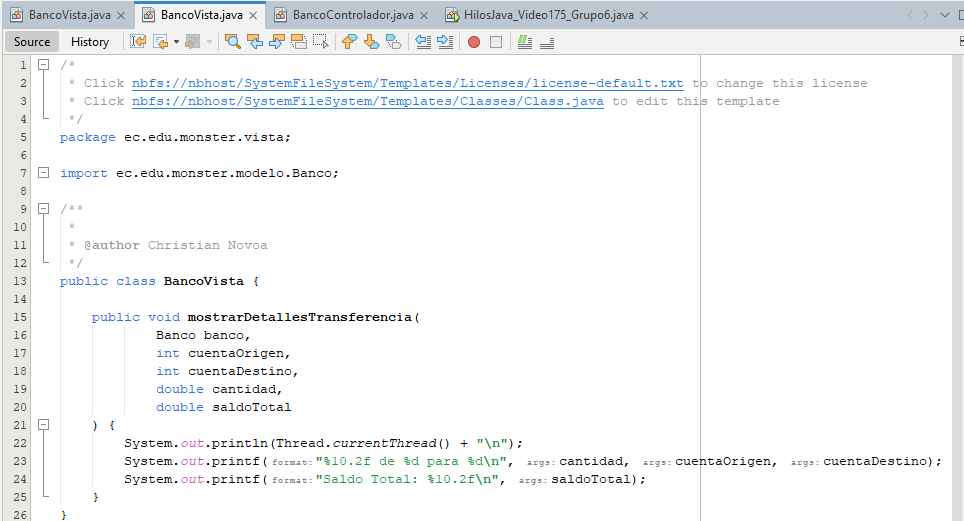
****

Figura 45. Codificación archivo BancoVista

Tabla . Codificación del archivo BancoVista

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoVista {

public void mostrarDetallesTransferencia(

Banco banco,

int cuentaOrigen,

int cuentaDestino,

double cantidad,

double saldoTotal

) {

System.out.println(Thread.currentThread() + "\n");

System.out.printf("%10.2f de %d para %d\n", cantidad, cuentaOrigen, cuentaDestino);

System.out.printf("Saldo Total: %10.2f\n", saldoTotal);

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 46. Creación de clase Java.

Le da el nombre de “BancoControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

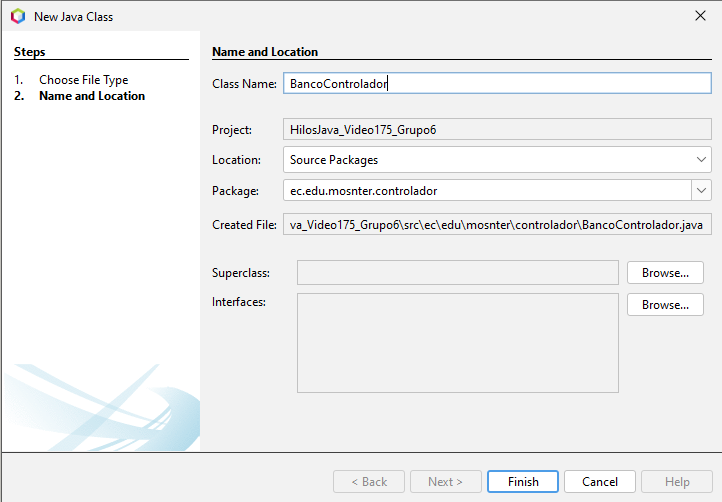


Figura 47. Codificación archivo BancoControlador.

En el código, a diferencia del anterior video, se incluyen los métodos unlock y lock de la clase ReentrantLock para impedir que varios hilos a la vez entren a ejecutar ciertas líneas de código y evitar así el error anterior que hacía que se pierda dinero del saldo total. Para poder utilizar las funciones de dicha clase es necesario añadir al código la importación del java.util.concurrent.locks como se muestra en el código a continuación y crear una instancia de la clase ReentrantLock.

Tabla . Codificación del archivo BancoControlador

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

import ec.edu.monster.vista.BancoVista;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoControlador {

private Lock cierreBanco = new ReentrantLock();

private BancoVista bancoView = new BancoVista();

public void acreditarSaldoCuentas(Banco banco, double saldo) {

double[] cuentasAcreditadas = new double[100];

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

cuentasAcreditadas[i] = saldo;

}

banco.cuentas = cuentasAcreditadas;

}

public void transferirEntreCuentas(Banco banco, int cuentaOrigen, int cuentaDestino, double cantidad) {

cierreBanco.lock();

try {

if (banco.cuentas[cuentaOrigen] < cantidad) {

return;

}

banco.cuentas[cuentaOrigen] -= cantidad;

banco.cuentas[cuentaDestino] += cantidad;

bancoView.mostrarDetallesTransferencia(banco, cuentaOrigen, cuentaDestino, cantidad, obtenerSaldoTotal(banco));

} finally {

cierreBanco.unlock();

}

}

public double obtenerSaldoTotal(Banco banco) {

double saldoTotal = 0;

for (double saldo : banco.cuentas) {

saldoTotal += saldo;

}

return saldoTotal;

}

}

Para la creación del segundo archivo de controlador se da clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 48. Creación de clase Java.

Le da el nombre de “TransferenciasControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 49. Codificación archivo TransferenciasControlador.

Tabla . Codificación del archivo TransferenciasControlador

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class TransferenciasControlador implements Runnable {

private Banco banco;

private BancoControlador bancoControlador;

private int cuentaOrigen;

private double valorMaximo;

public TransferenciasControlador(Banco banco, BancoControlador bancoControlador, int cuentaOrigen, double valorMaximo) {

this.banco = banco;

this.bancoControlador = bancoControlador;

this.cuentaOrigen = cuentaOrigen;

this.valorMaximo = valorMaximo;

}

@Override

public void run() {

while (true) {

int cuentaDestino = (int) (100 \* Math.random());

double cantidad = this.valorMaximo \* Math.random();

this.bancoControlador.transferirEntreCuentas(

this.banco,

this.cuentaOrigen,

cuentaDestino,

cantidad

);

try {

Thread.sleep((int) (Math.random() \* 10));

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.



Figura 50. Codificación archivo principal del video 175.

Tabla . Codificación del archivo principal del proyecto

package ec.edu.monster.main;

import ec.edu.monster.controlador.BancoControlador;

import ec.edu.monster.controlador.TransferenciasControlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class HilosJava\_Video175\_Grupo6 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Banco banco = new Banco();

BancoControlador bancoControlador = new BancoControlador();

bancoControlador.acreditarSaldoCuentas(banco, 2000);

for (int i = 0; i < 10; i++) {

TransferenciasControlador transferencias

= new TransferenciasControlador(banco, bancoControlador, i, 2000);

new Thread(transferencias).start();

}

}

}

### EJECUCIÓN DEL PROYECTO.

Al ejecutar el proyecto se puede observar por consola como se realizan por distintos hilos las transferencias infinitas desde la cantidad de dinero que pasa, la cuenta desde donde se realiza la transferencia y a la cuenta a la que llega. A diferencia del anterior video, en este ya no existen errores de pérdida de dinero en el saldo total al realizar transferencias, esto gracias al control del acceso de los hilos al código de transferencias, esto influye también en la impresión en consola de las transferencias pues ahora se imprimen de forma ordenada.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Figura 51. Resultado del video 175.

## Video 176

### 2.4.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.

Imagen que contiene Forma

Descripción generada automáticamente

Figura 52. Logotipo identificativo de Netbeans

Abra Apache Netbeans y se sitúa en la Pestaña superior y donde se selecciona File->New Project -> Java with Ant -> Java Application.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 53. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre HilosJava\_Video176\_Grupo#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “APLICATIVO” que fue creada con anterioridad, se marca la opción Create Main Class y se coloca en el mismo el siguiente nombre “ec.edu.monster.main.HilosJava\_Video176\_Grupo6” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

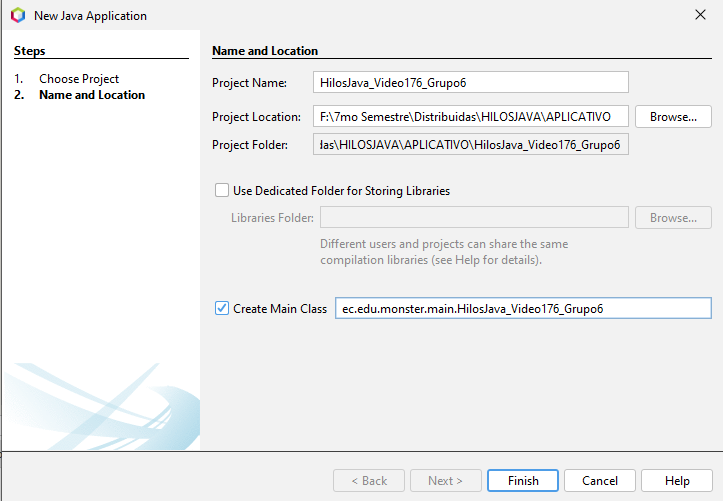
****

Figura 54. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

**Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media**

Figura 55. Estructura del proyecto generado.



### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 56. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 57. Selección del nombre de los paquetes del video 176

Repita el proceso hasta tener los 3 paquetes creados.

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 57. Creación de clase Java.

Le da el nombre de “Banco”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

Interfaz de usuario gráfica, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 58. Creación archivo Banco.

Proceda a colocar el siguiente código para construir su clase en el paquete de modelo.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Figura 59. Codificación archivo Banco

Tabla . Codificación del archivo Banco

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.modelo;

/\*\*

\*

\* @author Christian Novoa

\*/

public class Banco {

public double[] cuentas;

public Banco() {

this.cuentas = new double[100];

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 60. Creación de clase Java.

Le da el nombre de “BancoVista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 61. Creación archivo BancoVista.

Coloque el siguiente código para construir su clase en el paquete de vista.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Figura 62. Codificación archivo BancoVista

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 63. Continuación codificación archivo BancoVista

Tabla . Codificación del archivo BancoVista

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoVista {

public void mostrarTransferenciaError(Banco banco, int cuentaOrigen, double cantidad) {

System.out.println("---------CANTIDAD INSUFICIENTE---------");

System.out.println("\tCuenta: " + cuentaOrigen);

System.out.println("\tSaldo: " + banco.cuentas[cuentaOrigen]);

System.out.println("\tCantidad: " + cantidad);

}

public void mostrarTransferenciaExitosa(int cuentaOrigen) {

System.out.println("---------CANTIDAD OK---------");

System.out.println("\tCuenta: " + cuentaOrigen);

}

public void mostrarDetallesTransferencia(

Banco banco,

int cuentaOrigen,

int cuentaDestino,

double cantidad,

double saldoTotal

) {

System.out.println(Thread.currentThread() + "\n");

System.out.printf("%10.2f de %d para %d\n", cantidad, cuentaOrigen, cuentaDestino);

System.out.printf("Saldo Total: %10.2f\n", saldoTotal);

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

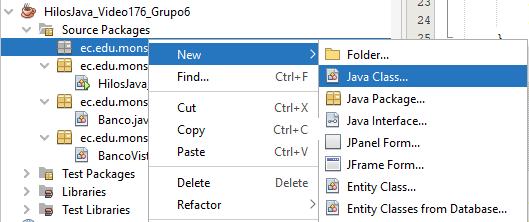


Figura 64. Creación de clase Java.

Le da el nombre de “BancoControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 65. Creación archivo BancoControlador.

Coloque el siguiente código para construir su clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 66. Codificación archivo BancoControlador.

Tabla . Codificación del archivo BancoControlador

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoControlador {

private Lock cierreBanco = new ReentrantLock();

public void acreditarSaldoCuentas(Banco banco, double saldo) {

double[] cuentasAcreditadas = new double[100];

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

cuentasAcreditadas[i] = saldo;

}

banco.cuentas = cuentasAcreditadas;

}

public void transferirEntreCuentas(Banco banco, int cuentaOrigen, int cuentaDestino, double cantidad) {

cierreBanco.lock();

try {

if (banco.cuentas[cuentaOrigen] < cantidad) {

return;

}

banco.cuentas[cuentaOrigen] -= cantidad;

banco.cuentas[cuentaDestino] += cantidad;

System.out.println(Thread.currentThread() + "\n");

System.out.printf("%10.2f de %d para %d\n", cantidad, cuentaOrigen, cuentaDestino);

System.out.printf("Saldo Total: %10.2f\n", obtenerSaldoTotal(banco));

} finally {

cierreBanco.unlock();

}

}

public double obtenerSaldoTotal(Banco banco) {

double saldoTotal = 0;

for (double saldo : banco.cuentas) {

saldoTotal += saldo;

}

return saldoTotal;

}

}

Para la creación del segundo archivo de controlador se da clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

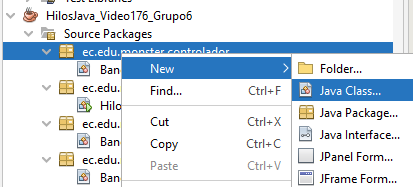


Figura 67. Creación de clase Java.

Le da el nombre de “TransferenciasControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 68. Creación de archivo TransferenciasControlador.

Texto

Descripción generada automáticamente con confianza media

Figura 69. Codificación archivo TransferenciasControlador.

Tabla . Codificación del archivo TransferenciasControlador

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class TransferenciasControlador implements Runnable {

private Banco banco;

private BancoControlador bancoControlador;

private int cuentaOrigen;

private double valorMaximo;

public TransferenciasControlador(Banco banco, BancoControlador bancoControlador, int cuentaOrigen, double valorMaximo) {

this.banco = banco;

this.bancoControlador = bancoControlador;

this.cuentaOrigen = cuentaOrigen;

this.valorMaximo = valorMaximo;

}

@Override

public void run() {

while (true) {

int cuentaDestino = (int) (100 \* Math.random());

double cantidad = this.valorMaximo \* Math.random();

this.bancoControlador.transferirEntreCuentas(

this.banco,

this.cuentaOrigen,

cuentaDestino,

cantidad

);

try {

Thread.sleep((int) (Math.random() \* 10));

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Texto

Descripción generada automáticamente

Figura 70. Codificación archivo principal del video 176.

Tabla . Codificación del archivo principal del proyecto

package ec.edu.monster.main;

import ec.edu.monster.controlador.BancoControlador;

import ec.edu.monster.controlador.TransferenciasControlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class HilosJava\_Video176\_Grupo6 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Banco banco = new Banco();

BancoControlador bancoControlador = new BancoControlador();

bancoControlador.acreditarSaldoCuentas(banco, 2000);

for (int i = 0; i < 10; i++) {

TransferenciasControlador transferencias

= new TransferenciasControlador(banco, bancoControlador, i, 2000);

new Thread(transferencias).start();

}

}

}

### EJECUCIÓN DEL PROYECTO.

Al ejecutar el proyecto se puede observar por consola como se realizan transferencias infinitas por distintos hilos desde la cantidad de dinero que pasa, la cuenta desde donde se realiza la transferencia y a la cuenta a la que llega. Además, se mostrará si la transferencia fue exitosa o no se pudo realizar por fondos en la cuenta insuficientes.

Tabla

Descripción generada automáticamente con confianza media

Figura 71. Resultado del video 176.

## Video 177

### 2.5.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.

Imagen que contiene Forma

Descripción generada automáticamente

Figura 72. Logotipo identificativo de Netbeans

Abra Apache Netbeans y se sitúa en la Pestaña superior y donde se selecciona File->New Project -> Java with Ant -> Java Application.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 73. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre HilosJava\_Video177\_Grupo#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “APLICATIVO” que fue creada con anterioridad, se marca la opción Create Main Class y se coloca en el mismo el siguiente nombre “ec.edu.monster.main.HilosJava\_Video177\_Grupo6” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Figura 74. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

**Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente**

Figura 75. Estructura del proyecto generado.



### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 76. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 77. Selección del nombre de los paquetes del video 176

Repita el proceso hasta tener los 3 paquetes creados.

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 78. Creación de clase Java.

Le da el nombre de “Banco”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

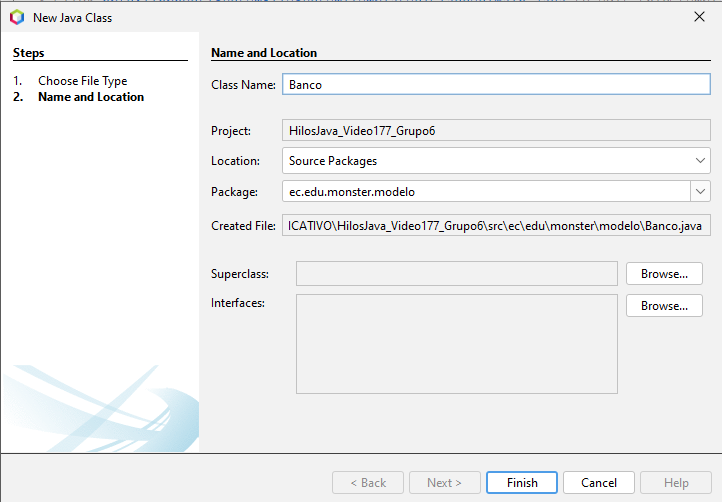


Figura 79. Creación archivo Banco.

Proceda a colocar el siguiente código para construir su clase en el paquete de modelo.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Figura 80. Codificación archivo Banco

Tabla . Codificación del archivo Banco

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.modelo;

/\*\*

\*

\* @author Christian Novoa

\*/

public class Banco {

public double[] cuentas;

public Banco() {

this.cuentas = new double[100];

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 81. Creación de clase Java.

Le da el nombre de “BancoVista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 82. Creación archivo BancoVista.

Coloque el siguiente código para construir su clase en el paquete de vista.

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Figura 83. Codificación archivo BancoVista

Tabla . Codificación del archivo BancoVista

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoVista {

public void mostrarTransferenciaError(Banco banco, int cuentaOrigen, double cantidad) {

System.out.println("---------CANTIDAD INSUFICIENTE---------");

System.out.println("\tCuenta: " + cuentaOrigen);

System.out.println("\tSaldo: " + banco.cuentas[cuentaOrigen]);

System.out.println("\tCantidad: " + cantidad);

}

public void mostrarTransferenciaExitosa(int cuentaOrigen) {

System.out.println("---------CANTIDAD OK---------");

System.out.println("\tCuenta: " + cuentaOrigen);

}

public void mostrarDetallesTransferencia(

Banco banco,

int cuentaOrigen,

int cuentaDestino,

double cantidad,

double saldoTotal

) {

System.out.println(Thread.currentThread() + "\n");

System.out.printf("%10.2f de %d para %d\n", cantidad, cuentaOrigen, cuentaDestino);

System.out.printf("Saldo Total: %10.2f\n", saldoTotal);

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 84. Creación de clase Java.

Le da el nombre de “BancoControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 85. Creación archivo BancoControlador.

Coloque el siguiente código para construir su clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 86. Codificación archivo BancoControlador.

Tabla . Codificación del archivo BancoControlador

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

import ec.edu.monster.vista.BancoVista;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoControlador {

private Condition saldoSuficiente;

private Lock cierreBanco = new ReentrantLock();

private BancoVista bancoView = new BancoVista();

public void acreditarSaldoCuentas(Banco banco, double saldo) {

double[] cuentasAcreditadas = new double[100];

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

cuentasAcreditadas[i] = saldo;

}

banco.cuentas = cuentasAcreditadas;

saldoSuficiente = cierreBanco.newCondition();

}

public void transferirEntreCuentas(Banco banco, int cuentaOrigen, int cuentaDestino, double cantidad)

throws InterruptedException {

cierreBanco.lock();

try {

while (banco.cuentas[cuentaOrigen] < cantidad) {

saldoSuficiente.await();

}

bancoView.mostrarTransferenciaExitosa(cuentaOrigen);

banco.cuentas[cuentaOrigen] -= cantidad;

banco.cuentas[cuentaDestino] += cantidad;

bancoView.mostrarDetallesTransferencia(banco, cuentaOrigen, cuentaDestino, cantidad,

obtenerSaldoTotal(banco));

saldoSuficiente.signalAll();

} finally {

cierreBanco.unlock();

}

}

public double obtenerSaldoTotal(Banco banco) {

double saldoTotal = 0;

for (double saldo : banco.cuentas) {

saldoTotal += saldo;

}

return saldoTotal;

}

}

Para la creación del segundo archivo de controlador se da clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 87. Creación de clase Java.

Le da el nombre de “TransferenciasControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 88. Creación de archivo TransferenciasControlador.

Texto

Descripción generada automáticamente con confianza media

Figura 89. Codificación archivo TransferenciasControlador.

Tabla . Codificación del archivo TransferenciasControlador

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class TransferenciasControlador implements Runnable {

private Banco banco;

private BancoControlador bancoControlador;

private int cuentaOrigen;

private double valorMaximo;

public TransferenciasControlador(Banco banco, BancoControlador bancoControlador, int cuentaOrigen, double valorMaximo) {

this.banco = banco;

this.bancoControlador = bancoControlador;

this.cuentaOrigen = cuentaOrigen;

this.valorMaximo = valorMaximo;

}

@Override

public void run() {

try {

while (true) {

int cuentaDestino = (int) (100 \* Math.random());

double cantidad = this.valorMaximo \* Math.random();

this.bancoControlador.transferirEntreCuentas(

this.banco,

this.cuentaOrigen,

cuentaDestino,

cantidad

);

Thread.sleep((int) (Math.random() \* 10));

}

} catch (InterruptedException e1) {

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Tabla

Descripción generada automáticamente

Figura 90. Codificación archivo principal del video 177.

Tabla . Codificación del archivo principal del proyecto

package ec.edu.monster.main;

import ec.edu.monster.controlador.BancoControlador;

import ec.edu.monster.controlador.TransferenciasControlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class HilosJava\_Video177\_Grupo6 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Banco banco = new Banco();

BancoControlador bancoControlador = new BancoControlador();

bancoControlador.acreditarSaldoCuentas(banco, 200000);

for (int i = 0; i < 100; i++) {

TransferenciasControlador transferencias

= new TransferenciasControlador(banco, bancoControlador, i, 200000);

new Thread(transferencias).start();

}

}

}

### EJECUCIÓN DEL PROYECTO.

Al ejecutar el proyecto se puede observar por consola como se realizan transferencias por distintos hilos desde la cantidad de dinero que pasa, la cuenta desde donde se realiza la transferencia y a la cuenta a la que llega. En esta versión todos los hilos que entran terminan realizando una transacción, es decir, no hay hilos que mueran sin realizar su tarea.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Figura 91. Resultado del video 177.

## Video 178

### 2.6.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.

Imagen que contiene Forma

Descripción generada automáticamente

Figura 92. Logotipo identificativo de Netbeans

Abra Apache Netbeans y se sitúa en la Pestaña superior y donde se selecciona File->New Project -> Java with Ant -> Java Application.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 93. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre HilosJava\_Video178\_Grupo#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “APLICATIVO” que fue creada con anterioridad, se marca la opción Create Main Class y se coloca en el mismo el siguiente nombre “ec.edu.monster.main.HilosJava\_Video178\_Grupo6” para tener un elemento principal que invoque inicialmente a los demás elementos presentes en el modelo de arquitectura MVC.

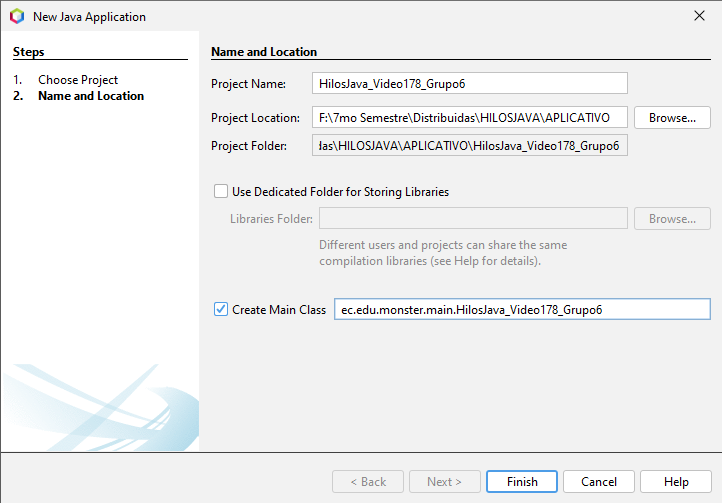
****

Figura 94. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

**Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media**

Figura 95. Estructura del proyecto generado.



### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 96. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 97. Selección del nombre de los paquetes del video 176

Repita el proceso hasta tener los 3 paquetes creados.

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 98. Creación de clase Java.

Le da el nombre de “Banco”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 99. Creación archivo Banco.

Proceda a colocar el siguiente código para construir su clase en el paquete de modelo.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

Figura 100. Codificación archivo Banco

Tabla . Codificación del archivo Banco

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.modelo;

/\*\*

\*

\* @author Christian Novoa

\*/

public class Banco {

public double[] cuentas;

public Banco() {

this.cuentas = new double[100];

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente clase Java.

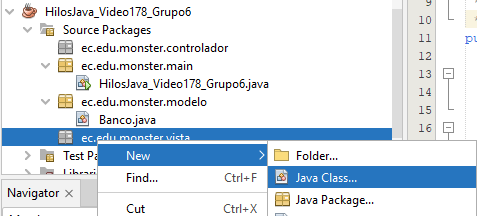


Figura 101. Creación de clase Java.

Le da el nombre de “BancoVista”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 102. Creación archivo BancoVista.

Coloque el siguiente código para construir su clase en el paquete de vista.

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Figura 103. Codificación archivo BancoVista

Tabla . Codificación del archivo BancoVista

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.vista;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoVista {

public void mostrarTransferenciaError(Banco banco, int cuentaOrigen, double cantidad) {

System.out.println("---------CANTIDAD INSUFICIENTE---------");

System.out.println("\tCuenta: " + cuentaOrigen);

System.out.println("\tSaldo: " + banco.cuentas[cuentaOrigen]);

System.out.println("\tCantidad: " + cantidad);

}

public void mostrarTransferenciaExitosa(int cuentaOrigen) {

System.out.println("---------CANTIDAD OK---------");

System.out.println("\tCuenta: " + cuentaOrigen);

}

public void mostrarDetallesTransferencia(

Banco banco,

int cuentaOrigen,

int cuentaDestino,

double cantidad,

double saldoTotal

) {

System.out.println(Thread.currentThread() + "\n");

System.out.printf("%10.2f de %d para %d\n", cantidad, cuentaOrigen, cuentaDestino);

System.out.printf("Saldo Total: %10.2f\n", saldoTotal);

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 104. Creación de clase Java.

Le da el nombre de “BancoControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 105. Creación archivo BancoControlador.

Coloque el siguiente código para construir su clase en el paquete de controlador.

Imagen que contiene Tabla

Descripción generada automáticamente

Figura 106. Codificación archivo BancoControlador.

Tabla . Codificación del archivo BancoControlador

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

import ec.edu.monster.vista.BancoVista;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Christian Novoa

\*/

public class BancoControlador {

private Condition saldoSuficiente;

private Lock cierreBanco = new ReentrantLock();

private BancoVista bancoView = new BancoVista();

public void acreditarSaldoCuentas(Banco banco, double saldo) {

double[] cuentasAcreditadas = new double[100];

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

cuentasAcreditadas[i] = saldo;

}

banco.cuentas = cuentasAcreditadas;

saldoSuficiente = cierreBanco.newCondition();

}

public void transferirEntreCuentas(Banco banco, int cuentaOrigen, int cuentaDestino, double cantidad)

throws InterruptedException {

cierreBanco.lock();

try {

while (banco.cuentas[cuentaOrigen] < cantidad) {

saldoSuficiente.await();

}

bancoView.mostrarTransferenciaExitosa(cuentaOrigen);

banco.cuentas[cuentaOrigen] -= cantidad;

banco.cuentas[cuentaDestino] += cantidad;

bancoView.mostrarDetallesTransferencia(banco, cuentaOrigen, cuentaDestino, cantidad,

obtenerSaldoTotal(banco));

saldoSuficiente.signalAll();

} finally {

cierreBanco.unlock();

}

}

public double obtenerSaldoTotal(Banco banco) {

double saldoTotal = 0;

for (double saldo : banco.cuentas) {

saldoTotal += saldo;

}

return saldoTotal;

}

}

Para la creación del segundo archivo de controlador se da clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 107. Creación de clase Java.

Le da el nombre de “TransferenciasControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 108. Creación de archivo TransferenciasControlador.

Texto

Descripción generada automáticamente con confianza media

Figura 109. Codificación archivo TransferenciasControlador.

Tabla . Codificación del archivo TransferenciasControlador

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class TransferenciasControlador implements Runnable {

private Banco banco;

private BancoControlador bancoControlador;

private int cuentaOrigen;

private double valorMaximo;

public TransferenciasControlador(Banco banco, BancoControlador bancoControlador, int cuentaOrigen, double valorMaximo) {

this.banco = banco;

this.bancoControlador = bancoControlador;

this.cuentaOrigen = cuentaOrigen;

this.valorMaximo = valorMaximo;

}

@Override

public void run() {

try {

while (true) {

int cuentaDestino = (int) (100 \* Math.random());

double cantidad = this.valorMaximo \* Math.random();

this.bancoControlador.transferirEntreCuentas(

this.banco,

this.cuentaOrigen,

cuentaDestino,

cantidad

);

Thread.sleep((int) (Math.random() \* 10));

}

} catch (InterruptedException e1) {

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN

Abra el archivo que fue creado por defecto dentro del paquete main y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Texto

Descripción generada automáticamente

Figura 110. Codificación archivo principal del video 178.

Tabla . Codificación del archivo principal del proyecto

package ec.edu.monster.main;

import ec.edu.monster.controlador.BancoControlador;

import ec.edu.monster.controlador.TransferenciasControlador;

import ec.edu.monster.modelo.Banco;

/\*\*

\*

\* @author Christian Novoa

\*/

public class HilosJava\_Video178\_Grupo6 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Banco banco = new Banco();

BancoControlador bancoControlador = new BancoControlador();

bancoControlador.acreditarSaldoCuentas(banco, 200000);

for (int i = 0; i < 100; i++) {

TransferenciasControlador transferencias

= new TransferenciasControlador(banco, bancoControlador, i, 200000);

new Thread(transferencias).start();

}

}

}

### EJECUCIÓN DEL PROYECTO.

Al ejecutar el proyecto se puede observar por consola como se realizan transferencias por distintos hilos desde la cantidad de dinero que pasa, la cuenta desde donde se realiza la transferencia y a la cuenta a la que llega. En esta versión el programa funciona de la misma forma que la anterior con el cambio de que en esta se utiliza la palabra reservada “Synchronized” para el bloqueo de la sincronización de hilos.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 111. Resultado del video 178.

# CONCLUSIONES

* Los Hilos en java nos permite generar programas capaces de realizar múltiples tareas al mismo tiempo que pueden aprovechar todas las capacidades de procesamiento de un computador.
* La sincronización de hilos es esencial para garantizar un comportamiento correcto y predecible en aplicaciones concurrentes. El desarrollo de un programa que implementa técnicas de sincronización adecuadas puede evitar problemas de concurrencia y mejorar el rendimiento general. Sin embargo, es importante tener en cuenta que la sincronización también puede introducir cierta sobrecarga y complejidad adicional.
* Java nos permite generar proyectos con la capacidad de utilizar hilos de manera fácil y rápida ya que incluye métodos y bibliotecas propias.
* Cuando se utilizan hilos las capacidades de nuestros proyectos aumentan ya que pueden hacer más tareas a la vez.

# RECOMENDACIONES

* Se recomienda renombrar el paquete principal del proyecto ya que esto nos ayuda a tener una mejor organización de nuestro código.
* Utilizar el modelo MVC ya que de esta manera podemos realizar modificaciones de nuestros programas sin la necesidad de cambiar todo el código.
* Para futuras mejoras del programa, se sugiere considerar técnicas de sincronización más avanzadas, como monitores o semáforos, dependiendo de las necesidades específicas. Además, es recomendable realizar pruebas exhaustivas en diferentes escenarios de carga y evaluar el rendimiento en diferentes configuraciones de hardware.

# BIBLIOGRAFIA

|  |  |
| --- | --- |
| [1] | ORACLE CORPORATION, «JAVA INTRODUCCION,» ORACLE CORPORATION, 2017. [En línea]. Available: https://aws.amazon.com/es/what-is/java/. [Último acceso: 05 11 2022]. |
| [2] | Netbeans, «Welcome to Apache NetBeans,» Netbeans, [En línea]. Available: https://netbeans.apache.org/. [Último acceso: 05 11 2022]. |
| [3] | theastrology, «theastrology,» theastrology, 2018. [En línea]. Available: https://es.theastrologypage.com/concurrency. [Último acceso: 26 11 2022]. |
| [4] | EDTeam, «EDTeam,» EDTeam, 2018. [En línea]. Available: https://ed.team/blog/como-funcionan-los-hilos-en-programacion. [Último acceso: 26 11 2022]. |
| [5] | Universidad de Alicante, «Universidad de Alicante,» Universidad de Alicante, 2012. [En línea]. Available: http://www.jtech.ua.es/dadm/restringido/java/sesion05-apuntes.html#:~:text=En%20Java%20los%20hilos%20est%C3%A1n,definir%20el%20m%C3%A9todo%20run().. [Último acceso: 26 11 2022]. |