

**Tema**

MANUAL

TALLER HILOS

**Autores**

JOHAO ALEJANDRO MORALES PISCO

MAYCOL ESTALIN TITUAÑA TUPIZA

ALEX DARIO VELASTEGUI SOLIS

**Tutor**

Ing. Eduardo Mauricio Campana Ortega

MIS.MDU.CCNA.CCIA.

PhD.(c) Ingeniería de Software PhD.(c) Seguridad Información

**Fecha**

08/07/2023

**TABLA DE CONTENIDOS**

[INTRODUCCIÓN 4](#_Toc139716801)

[OBJETIVO 4](#_Toc139716802)

[MARCO TEÓRICO 4](#_Toc139716803)

[JAVA 4](#_Toc139716804)

[PROGRAMAS MONOTAREA 5](#_Toc139716805)

[PROGRAMAS MULTITAREA 5](#_Toc139716806)

[PROGRAMACIÓN CONCURRENTE 5](#_Toc139716807)

[PATRÓN MODELO VISTA CONTROLADOR 6](#_Toc139716808)

[MODELO 7](#_Toc139716809)

[VISTA 7](#_Toc139716810)

[AWT 7](#_Toc139716811)

[SWING 8](#_Toc139716812)

[CONTROLADOR 9](#_Toc139716813)

[HILOS 9](#_Toc139716814)

[CICLO DE VIDA DE UN HILO 10](#_Toc139716815)

[OTROS MÉTODOS UTILIZADOS EN THREADS 11](#_Toc139716816)

[INTERFAZ RUNNABLE 11](#_Toc139716817)

[USO DE EXCEPCIONES 12](#_Toc139716818)

[INTERFAZ ACTIONLISTENER 13](#_Toc139716819)

[DESCRIPCIÓN DEL TALLER DE HILOS 13](#_Toc139716820)

[INSTALACIÓN DE LAS HERRAMIENTAS 14](#_Toc139716821)

[INSTALACIÓN DE NETBEANS 14](#_Toc139716822)

[DESARROLLO DE LA APLICACIÓN 25](#_Toc139716823)

[CREACIÓN DEL PROYECTO 25](#_Toc139716824)

[MODELO 32](#_Toc139716825)

[Creación del modelo Banco 32](#_Toc139716826)

[CONTROLADOR 36](#_Toc139716827)

[Creación del controlador EjecucionTransferencias 36](#_Toc139716828)

[CÓDIGO FUENTE 41](#_Toc139716829)

[MAIN 41](#_Toc139716830)

[EJECUCIÓN DEL PROYECTO 43](#_Toc139716831)

[BANCO 43](#_Toc139716832)

[CONCLUSIONES 44](#_Toc139716833)

[RECOMENDACIONES 44](#_Toc139716834)

[BIBLIOGRAFÍA 45](#_Toc139716835)

**ÍNDICE DE FIGURAS**

[Figura 1 Logo de Java 6](#_Toc139719770)

[Figura 2 Esquema del patrón MVC 7](#_Toc139719771)

[Figura 3 Esquema de componentes de la librería AWT 9](#_Toc139719772)

[Figura 4 Esquema de componentes de la librería Swing 10](#_Toc139719773)

[Figura 5 Esquema del ciclo de vida de un hilo 12](#_Toc139719774)

[Figura 6 Ejemplo de uso de la interfaz Runnable 13](#_Toc139719775)

[Figura 7 Ejemplo de manejo de excepciones en Java 14](#_Toc139719776)

[Figura 8 Página descarga NetBeans 15](#_Toc139719777)

[Figura 9 Página descarga imagen Java 16](#_Toc139719778)

[Figura 10 Guardar archivo en computador 16](#_Toc139719779)

[Figura 11 Ventana instalador java 16](#_Toc139719780)

[Figura 12 Selección características JDK, JRE 17](#_Toc139719781)

[Figura 13 Selección carpeta destino Java 17](#_Toc139719782)

[Figura 14 Finalizar la instalación Java 18](#_Toc139719783)

[Figura 15 Instalación NetBeans 19](#_Toc139719784)

[Figura 16 Guardar el archivo 20](#_Toc139719785)

[Figura 17 Configuración del instalador NetBeans 20](#_Toc139719786)

[Figura 18 Características a instalar NetBeans 21](#_Toc139719787)

[Figura 19 Selección de paquetes y servidores a instalar 21](#_Toc139719788)

[Figura 20 Aceptación de términos 22](#_Toc139719789)

[Figura 21 Selección de carpeta de instalación 23](#_Toc139719790)

[Figura 22 Instalación de paquetes seleccionados 23](#_Toc139719791)

[Figura 23 Finalización de la instalación 24](#_Toc139719792)

[Figura 24 Crear un nuevo proyecto 24](#_Toc139719793)

[Figura 25 Selección del tipo de proyecto a crear 25](#_Toc139719794)

[Figura 26 Asignación de nombre al proyecto 26](#_Toc139719795)

[Figura 27 Creación de una nueva clase 27](#_Toc139719796)

[Figura 28 Crear un nuevo paquete 28](#_Toc139719797)

[Figura 29 Creación del paquete modelo 29](#_Toc139719798)

[Figura 30 Creación del paquete vista 30](#_Toc139719799)

[Figura 31 Estructura del proyecto MVC 30](#_Toc139719800)

[Figura 32 Creación de nueva clase 31](#_Toc139719801)

[Figura 33 Atributos de la clase Banco 31](#_Toc139719802)

[Figura 34 Inicialización de 100 cuentas en constructor 32](#_Toc139719803)

[Figura 35 Creación de función getSaldoTotal 32](#_Toc139719804)

[Figura 36 Creación de función transferencia 33](#_Toc139719805)

[Figura 37 Código del modelo Banco 33](#_Toc139719806)

[Figura 38 Creación de nueva clase 35](#_Toc139719807)

[Figura 39 Asignación del nombre del controlador 36](#_Toc139719808)

[Figura 40 Atributos del controlador 37](#_Toc139719809)

[Figura 41 Función run 37](#_Toc139719810)

[Figura 42 Creación del constructor de la clase del controlador 37](#_Toc139719811)

[Figura 43 Código del controlador 38](#_Toc139719812)

[Figura 44 Aplicación del Banco 41](#_Toc139719813)

# INTRODUCCIÓN

Los hilos son otra forma de crear la posibilidad de concurrencia de actividades; sin embargo, la gran diferencia es que los hilos comparten el código y el acceso a algunos datos en forma similar a como un objeto tiene acceso a otros objetos.

En Java un hilo es un objeto con capacidad de correr en forma concurrente el método run(). En cierta manera es como tener dos "program counters" para un mismo código. Una diferencia con los procesos es que carece de sentido y no es posible en este enfoque hacer mutar un proceso con algo similar a exec().

# OBJETIVO

Desarrollar e implementar una aplicación que utilice hilos mediante JAVA para realizar un programa animado sobre unas pelotas que se mueven por toda la pantalla de la aplicación con la ayuda del IDE de desarrollo Apache Netbeans.

# MARCO TEÓRICO

## JAVA

Java es un lenguaje de programación creado por Sun Microsystems en 1995. Ha evolucionado desde sus comienzos hasta ser una gran parte del mundo digital actual, debido a que es una plataforma fiable en la que se crean gran cantidad de aplicaciones y servicios [1].

Java es un lenguaje que posee varias características, siendo estas:

* Simple y potente: Java ofrece un lenguaje potente, derivada de C y C++, pero sin las características más complicadas (y menos usadas) de estos.
* Orientado a objetos: el enfoque de Java es Orientado a Objetos (OO), es decir, posee los cuatro principios de la programación OO, siendo estos: herencia, encapsulamiento, polimorfismo y abstracción.
* Distribuido: Java proporciona de manera estándar bibliotecas y herramientas para que las aplicaciones desarrolladas sean distribuidas.
* Portable: los programas desarrollados en Java son independientes de la plataforma.
* Multihilo: Java puede llevar a cabo varias tareas de forma simultánea dentro del mismo programa, permitiendo mayor rendimiento y velocidad en la ejecución.



Figura 1 Logo de Java

## PROGRAMAS MONOTAREA

Los programas o sistemas monotarea son aquellos que solo permiten realizar una tarea a la vez por usuario [2]. Existen caso de sistemas monotarea, pero que son multiusuarios, en el cual se admiten varios usuarios al mismo tiempo, pero cada uno de ellos puede realizar únicamente una tarea a la vez [2].

## PROGRAMAS MULTITAREA

Un sistema o programa multitarea es aquel que permite que los usuarios realicen varias tareas al mismo tiempo [2]. Por ejemplo, muchos sistemas operativos son multitarea, ya que permiten escribir código de un programa, mientras se recibe un correo electrónico, y se consulta algo en internet. En estos tipos de programa es común encontrar interfaces gráficas que utilicen menús, lo cual permite al usuario un rápido cambio entre las varias tareas que se encuentre realizando [2].

## PROGRAMACIÓN CONCURRENTE

La programación concurrente es una forma de programa implementada con el fin de resolver varios problemas (realizar varias actividades) de forma concurrente, lo que significa que se ejecutan múltiples labores de programación al mismo tiempo, y no de forma secuencial (una actividad a la vez) [3].

Una de las características de la programación concurrente es que se clasifica como un método de computación modular, lo que se refiere que un cálculo entero tiene la posibilidad de dividirse en muchos subcálculos, que pueden ejecutarse de manera concurrente [3].

Este tipo de programación también se caracteriza por funcionar bajo el concepto de procesos; estos son ejecutados en el mismo momento, sin embargo, no se realizan de forma paralela, es decir, no se realizan todos al mismo tiempo [3]. En este tipo de programación, si un proceso no logra realizar su ejecución en el tiempo que le correspondía, el mismo sistema se encargará de ponerlo en pausa [3].

## PATRÓN MODELO VISTA CONTROLADOR

Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores.

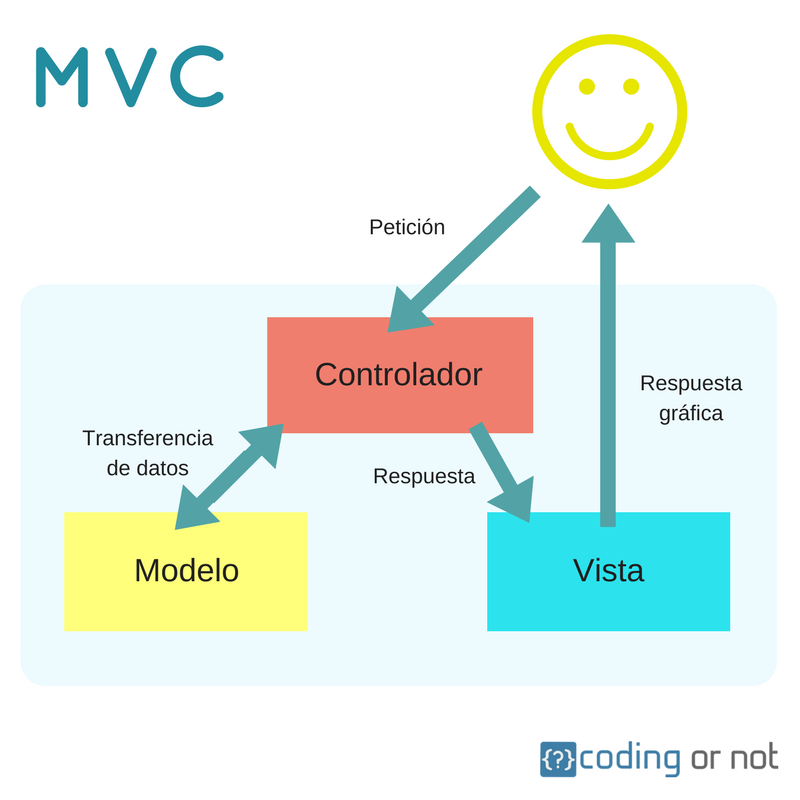


Figura 2 Esquema del patrón MVC

## MODELO

Es la capa donde se trabaja con los datos, por tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que nos permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

## VISTA

Las vistas, como su nombre nos hacen entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite mostrar la salida.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generarán la salida, tal como nuestra aplicación requiera.

### AWT

Java AWT (Abstract Window Toolkit) es una librería que permite desarrollar interfaces gráficas de usuario (GUI) en Java [4].

Los componentes de Java AWT dependen de la plataforma, es decir, que los componentes se muestran de acuerdo con la vista del sistema operativo [4]. AWT es pesado, ya que utiliza los recursos del sistema operativo subyacente [4].

AWT tiene clases como: TextField, Label, TextArea, Choice, RadioButton, etc., las cuales se encuentran dentro del paquete java.awt [4].

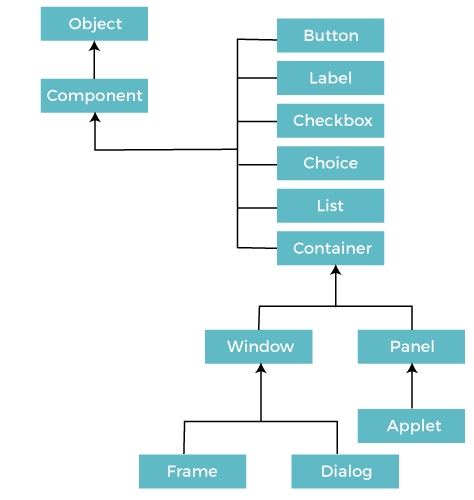


Figura 3 Esquema de componentes de la librería AWT

### SWING

Swing es una librería o paquete que es utilizado para crear aplicaciones gráficas que interactúen con el usuario [5]. Esta librería se encuentra disponible desde la versión 1.2 de Java [5].

Esta librería incluye todos los recursos necesarios para construir interfaces gráficas, varios que ha heredado de su predecesor AWT, y otros que han sido adicionados por la misma librería [5].

Su uso puede ser algo complejo, sin embargo, presenta la gran ventaja de que el código desarrollado puede ejecutarse en cualquier sistema operativo, manteniendo la portabilidad, que es una de las características que Java sigue ofreciendo [5].

Swing cuenta con clases como: JMenu, JCheckbox, JButton, JTextField, etc., las cuales se encuentran dentro del paquete javax.swing [5].

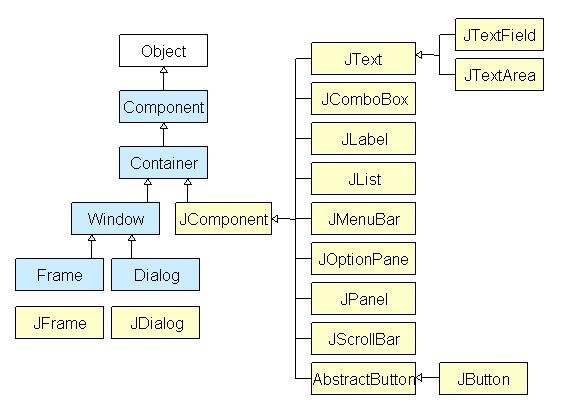


Figura 4 Esquema de componentes de la librería Swing

## CONTROLADOR

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad, es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

### HILOS

Un hilo es un flujo de control dentro de un programa [6]. Si se crean varios hilos se pueden realizar varias tareas simultáneamente [6]. Cada hilo tiene su contexto de ejecución (contador de programa, pila de ejecución) [6].

En Java los hilos están encapsulados en la clase Thread, y pueden crearse utilizando algunas de las siguientes dos posibilidades:

1. Heredar de la clase Thread, redefiniendo el método run() [6].
2. Que la clase a utiliza implemente la interfaz Runnable, la cual obliga a definir el método run() [6].

Es preferible utilizar el segundo método, debido a que no existe herencia múltiple en Java, por tanto, si se hereda de Thread, no se podrá heredar de ninguna otra clase; lo cual no sucede si se implementa la interfaz Runnable, debido a que una clase puede implementar varias interfaces.

El método run será el que contenga el código del hilo, y será quien se ejecute cuando se inicie la ejecución del hilo [6]. Desde dentro de este método se puede llamar a cualquier otro método de cualquier objeto [6]. El hilo terminará su ejecución cuando se termine de ejecutar el método run [6].

Un hilo se ejecuta llamando al método start, ya que, con esto, comenzará a ejecutarse el método run [6].

### CICLO DE VIDA DE UN HILO

Un hilo pasará por varios estados, desde su construcción hasta su destrucción. Una vez instanciado el objeto del hilo, se dirá que el hilo se encuentra en el estado “Nuevo hilo” [6].

Tras invocar al método start (se ejecuta el correspondiente método run), el hilo pasará a ser un hilo “vivo” [6]. Tras salir de este método el hilo pasará a ser un hilo “muerto” [6]. La única forma de parar un hilo es que salga de forma natural de su método run, esto puede realizarse esperando a que concluya el método run, o en su defecto con alguna condición lógica. Las funciones para parar, pausar y reanudar hilos están desaprobadas en las versiones actuales de Java [6].

Mientras un hilo se encuentre “vivo”, podrá encontrase en uno de los siguientes estados: “ejecutable” o “no ejecutable” [6]. El hilo pasará de “ejecutable” a “no ejecutable” en los siguientes casos:

* Cuando el hilo se encuentre dormido por haber llamado al método sleep, pasará al estado “no ejecutable” durante la cantidad de milisegundos indicada en el método [6].
* Cuando el hilo se encuentre bloqueado por haber llamado al método wait, pasará al estado “no ejecutable” hasta que otro hilo lo desbloquee llamando al método notify o notifyAll [6].
* El hilo puede pasar a “no ejecutable” debido a una petición E/S, y permanecerá es ese estado hasta que se complete la operación E/S [6].

Se puede conocer si un hilo se encuentra vivo o muerto, llamando al método isAlive.



Figura 5 Esquema del ciclo de vida de un hilo

### OTROS MÉTODOS UTILIZADOS EN THREADS

* stop(): este método es utilizado para detener la ejecución del thread. Ya casi no se utiliza.
* interrupt(): los objetos de la clase Thread cuentan con el método interrupt, lo que permite que el hilo sea interrumpido. El método stop dejó de ser utilizado por este.
* currentThread(): este método informa acerca del hilo que se encuentra en ejecución en un momento determinado.
* isInterrupted(): este método devuelve true si el hilo ha sido interrumpido, caso contrario devuelve false.

### INTERFAZ RUNNABLE

La interfaz Runnable proporciona un método alternativo a la utilización de la clase Thread; es utilizado para los casos en los que no sea posible hacer que la clase herede de la clase Thread, esto ocurre debido a que en Java no existe herencia múltiple, y en caso de que ya se esté heredando de alguna otra clase, será imposible crear hilos heredando de Thread [7].

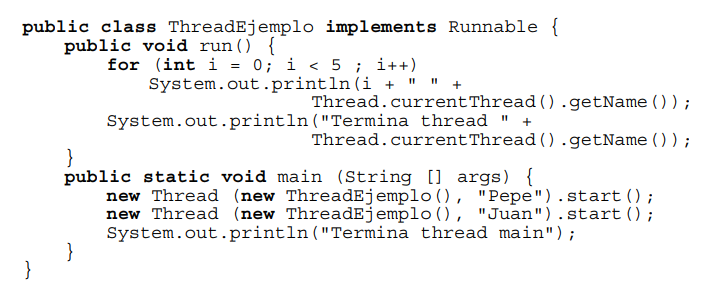


Figura 6 Ejemplo de uso de la interfaz Runnable

### USO DE EXCEPCIONES

Java proporciona el uso de excepciones para dar posibilidades de manejo de errores a los programas creados [8]. Una excepción es un evento que se produce cuando se ejecuta el programa de forma que interrumpe el flujo normal de instrucciones [8].

El sistema de ejecución Java y muchas clases de paquetes Java arrojan excepciones en determinadas circunstancias utilizando la sentencia throw [8].

Java permite manejar las excepciones a los programadores con el bloque try – catch. Dentro del bloque try, se coloca el código que va a ejecutarse y que es propenso a errores; y en el bloque catch, se especifica el tipo de error que puede arrojar el código encerrado en el bloque try, y se coloca las sentencias a ejecutar en caso de que el error se produzca.

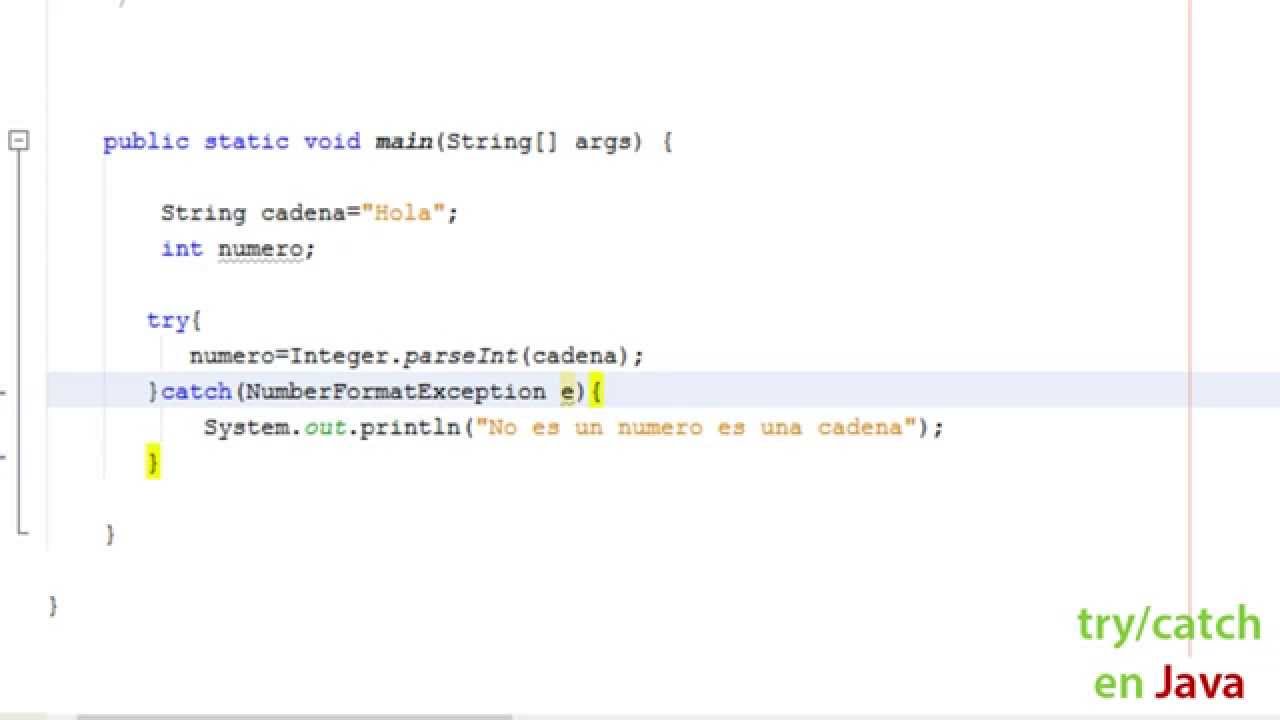


Figura 7 Ejemplo de manejo de excepciones en Java

### INTERFAZ ACTIONLISTENER

ActionListener es una interfaz de Java que se encuentra dentro del paquete java.awt.event. ActionLister pertenece al grupo de Listeners (escuchadores); posee un único método llamado: actionPerformed(ActionEvent e), en el que se especifican las acciones a realizar en caso de que un evento ocurra.

# DESCRIPCIÓN DEL TALLER DE HILOS

Realizar un taller práctico utilizando hilos con la ayuda del lenguaje de programación de JAVA en el IDE de desarrollo de NEATBEANS.

Además, en este se hará una simulación del rebote de unas pelotas saltarinas que será posible con la creación de varios hilos que simularan ser pelotas.

# INSTALACIÓN DE LAS HERRAMIENTAS

## INSTALACIÓN DE NETBEANS

1. Para instalar NetBeans, es necesario instalar java JDK y JRE, para esto se procede mediante la instalación de java SE que proporciona estos dos, de lo que cabe recalcar que la instalación deberá ser de la versión 8u211, se conseguirá en la siguiente página:

* <https://www.oracle.com/technetwork/es/java/javase/downloads/index.html>



Figura 8 Página descarga NetBeans

1. En esta página se da clic en Down load con la imagen de java.

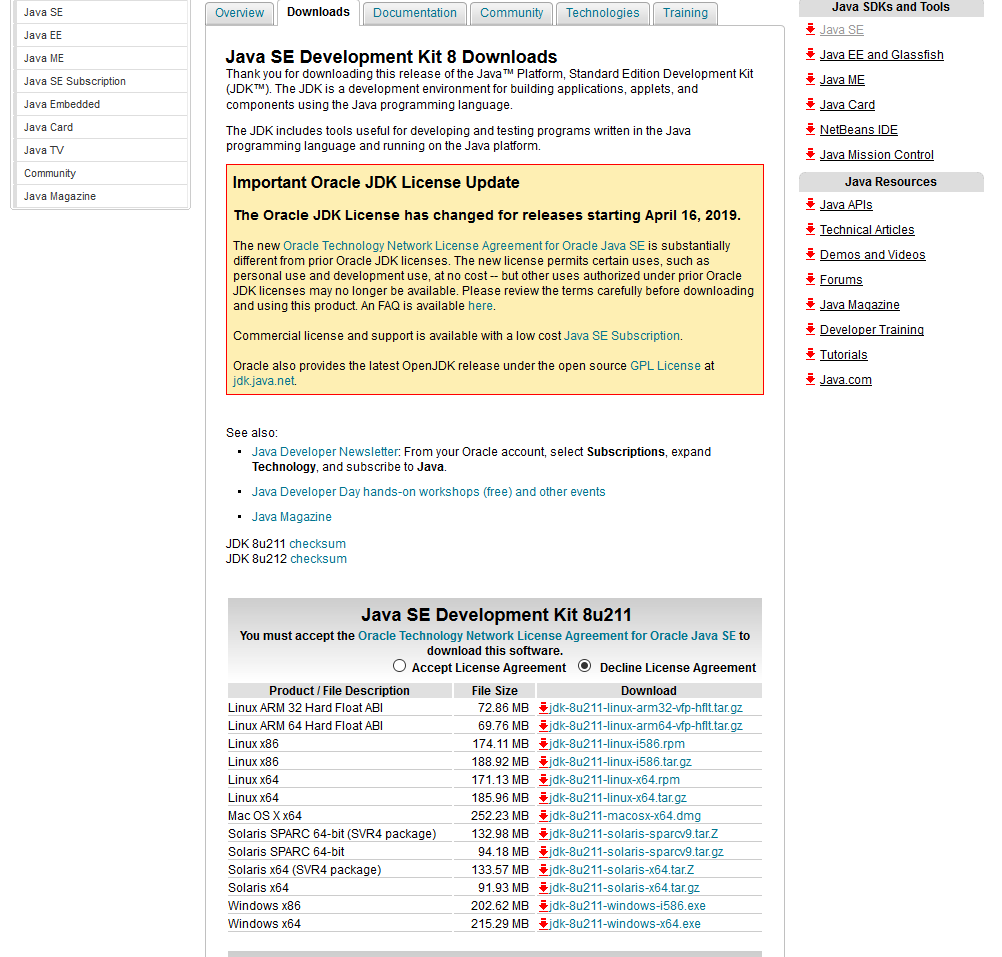


Figura 9 Página descarga imagen Java

1. Posteriormente se aceptan los acuerdos de licencia y se da clic en la versión del sistema operativo con el que se cuente, en este caso se utiliza Windows de 64 bits.

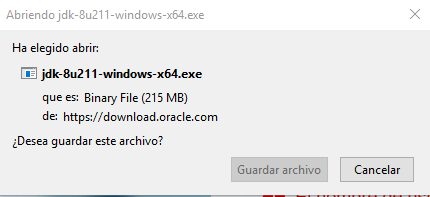


Figura 10 Guardar archivo en computador

1. Se guarda el archivo en el computador. Se ejecuta y en caso de requerirse se conceden permisos de administrador.

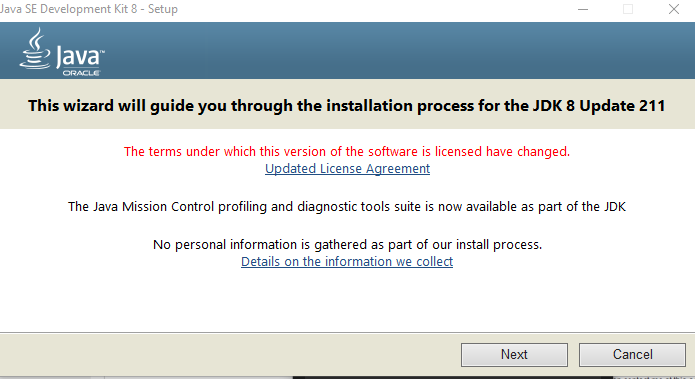


Figura 11 Ventana instalador java

1. Se presenta el instalador de java y se procede dando clic en siguiente.

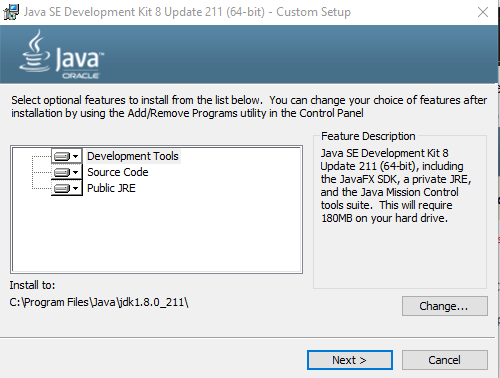


Figura 12 Selección características JDK, JRE

1. Seleccionan las características que irán con el JDK, al ser JSE también se instalará el JRE como se evidencia en la imagen anterior, se procede dando clic en siguiente.

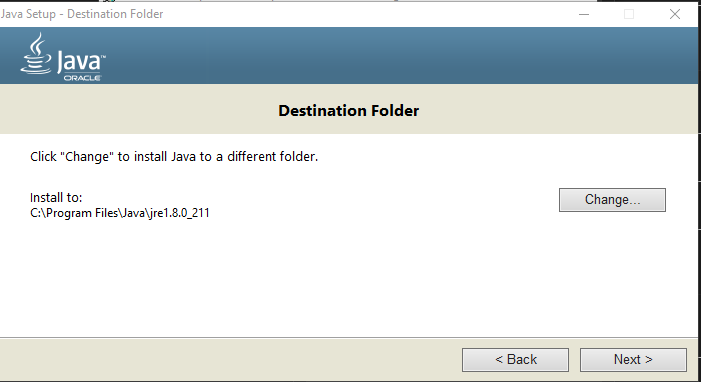


Figura 13 Selección carpeta destino Java

1. A la mitad de la instalación se instalará el JRE correspondiente, se procede dando clic en siguiente.

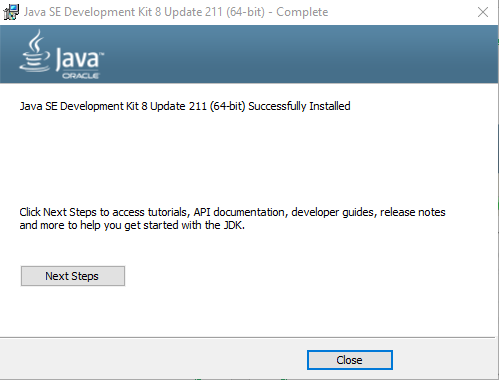


Figura 14 Finalizar la instalación Java

1. Al finalizar la instalación se finaliza la misma dando clic en cerrar.

Una vez se cuente con java instalado se procede con la instalación de NetBeans, su descarga se encuentra en la siguiente url:

* <https://netbeans.org/downloads/8.2/>

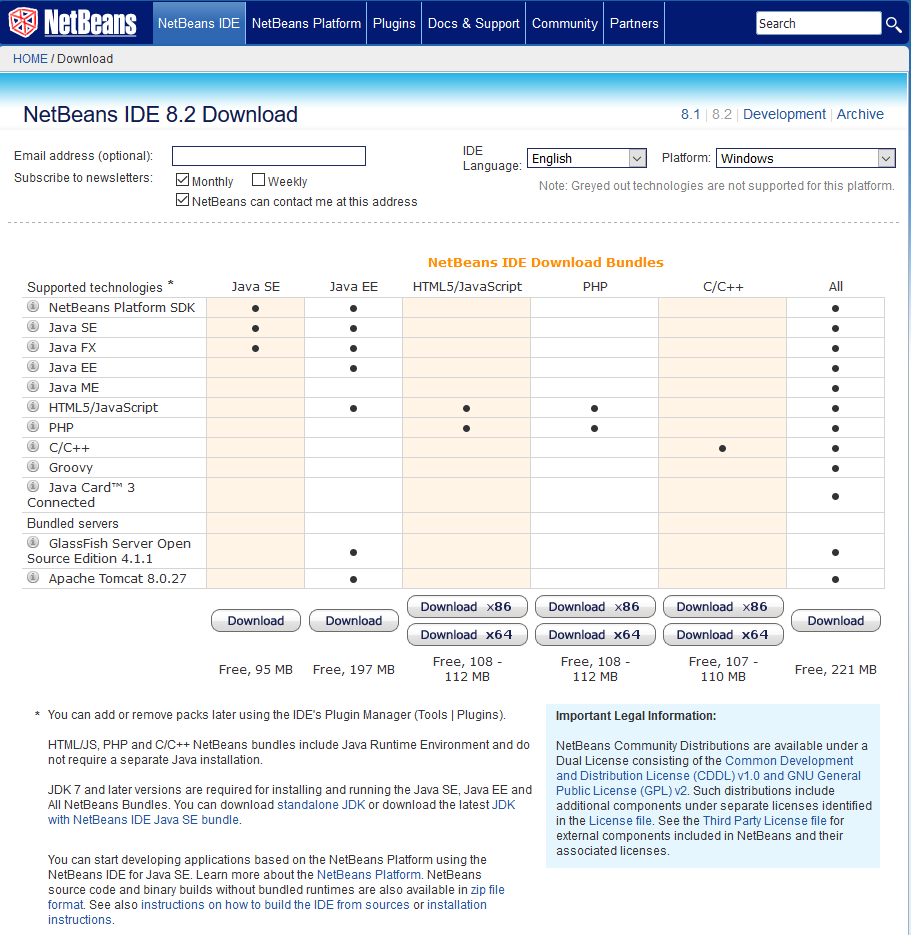


Figura 15 Instalación NetBeans

1. Dentro de esta página se selecciona el idioma del IDE por defecto en inglés, la plataforma, en la que se va a instalar por defecto Windows, y debe seleccionarse la versión completa del IDE, se encuentra en el campo All y se procede dando clic en Down load.

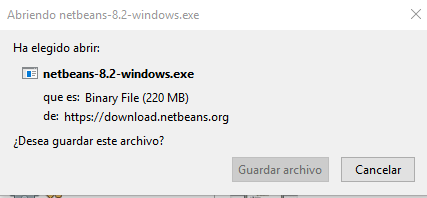


Figura 16 Guardar el archivo

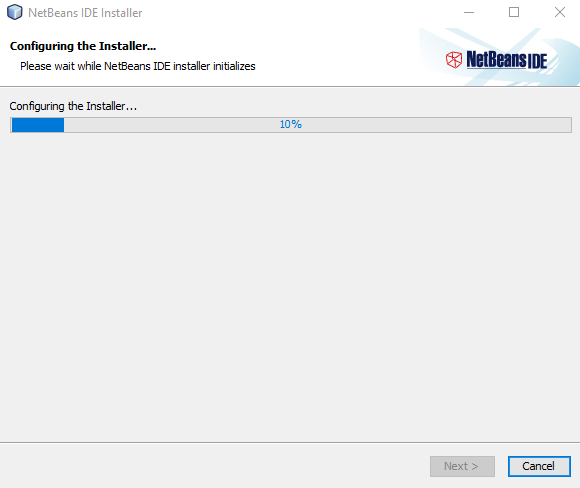


Figura 17 Configuración del instalador NetBeans

1. Al abrirse, NetBeans realiza una configuración del instalador.

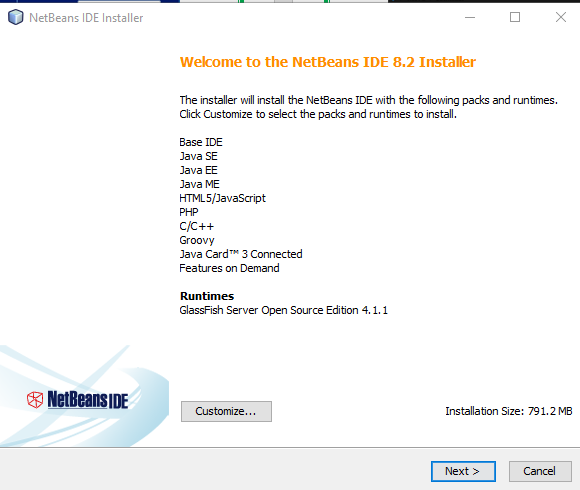


Figura 18 Características a instalar NetBeans

1. Posteriormente mostrará las características a instalar, en lo que se selecciona personalizar.

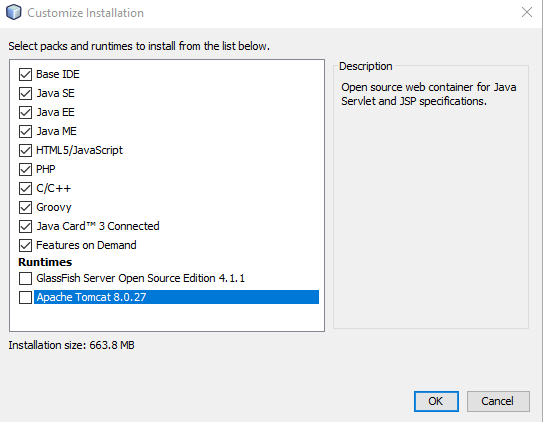


Figura 19 Selección de paquetes y servidores a instalar

1. En esta ventana se seleccionan los paquetes y servidores que se utilizaron en cuanto a paquetes, se seleccionan todos los disponibles de java y no se selecciona ningún servidor, puesto a que estos se instalarán posteriormente a la instalación de NetBeans, se da clic en ok y posteriormente en siguiente.

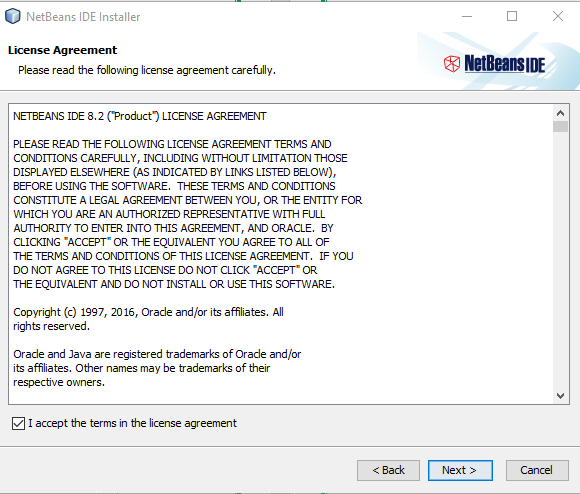


Figura 20 Aceptación de términos

1. En esta ventana se aceptan los términos y acuerdos de licencia y se da clic en siguiente.

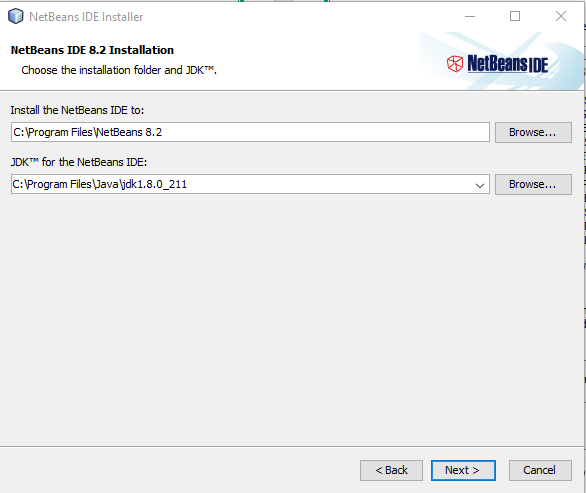


Figura 21 Selección de carpeta de instalación

1. Se seleccionan las carpetas de instalación de NetBeans y carpeta del JDK, esta última se cargará por defecto, pero en caso de que no deberá hacérselo, una vez puestas las carpetas se procede dando clic en siguiente.

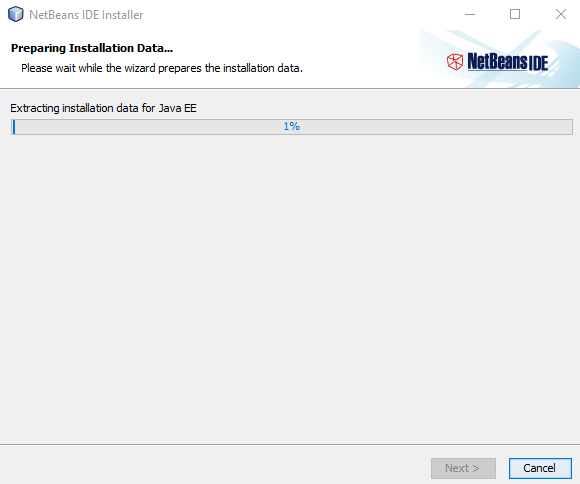


Figura 22 Instalación de paquetes seleccionados

1. Se instalarán todos los paquetes seleccionados previamente.

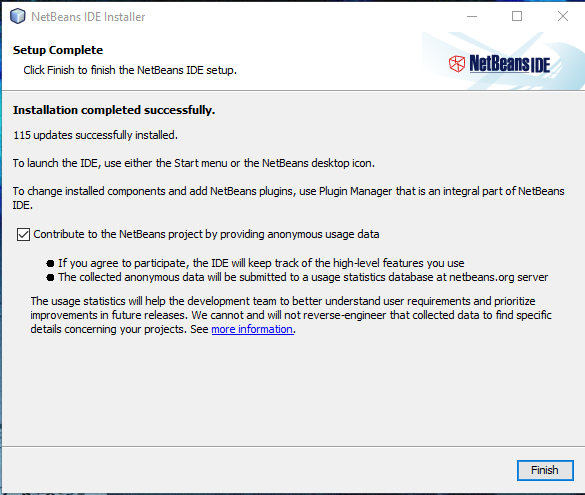


Figura 23 Finalización de la instalación

Al acabar la instalación solo se da clic en finalizar y listo, se ha concluido con la instalación de NetBeans.

# DESARROLLO DE LA APLICACIÓN

## CREACIÓN DEL PROYECTO

1. Crear un nuevo proyecto en NetBeans dando clic en “New Project”.

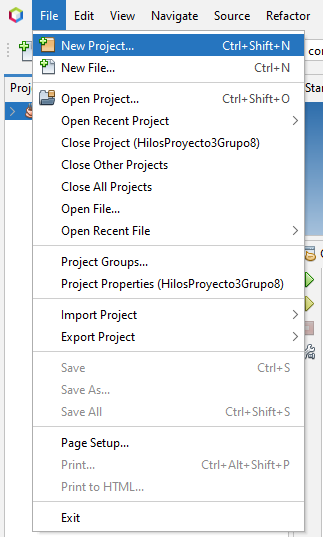


Figura 24 Crear un nuevo proyecto

1. Seleccionamos la opción "Java with Ant" y, a continuación, elegimos "Java application". Luego, procedemos a hacer clic en el botón "Siguiente".

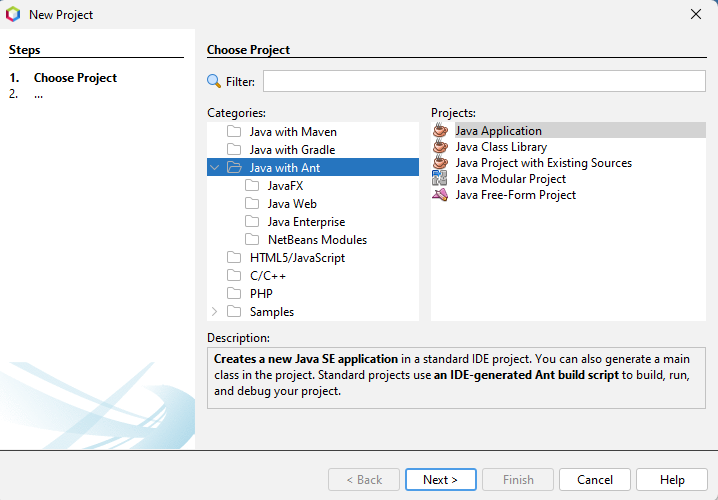


Figura 25 Selección del tipo de proyecto a crear

1. Finalmente creamos el proyecto y le damos el nombre de “Sincronizacion\_Hilos\_Banco”

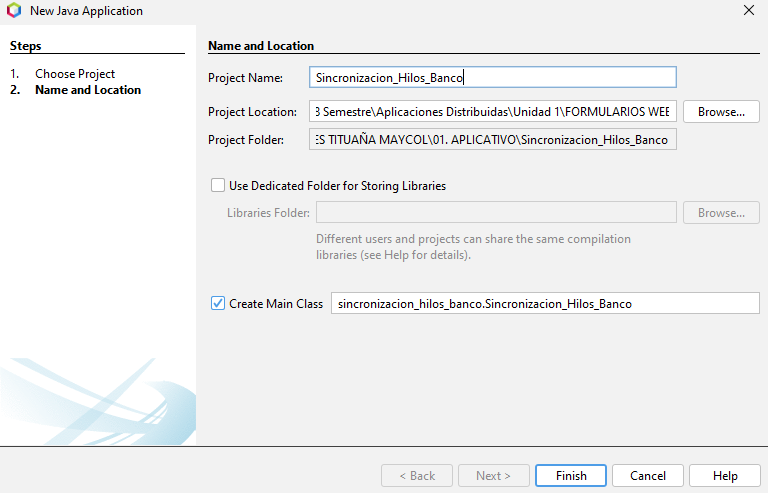


Figura 26 Asignación de nombre al proyecto

**CREACION DE PAQUETES**

1. Primero damos clic derecho en Sourcee Packages, new y por ultimo clic en Java Package.

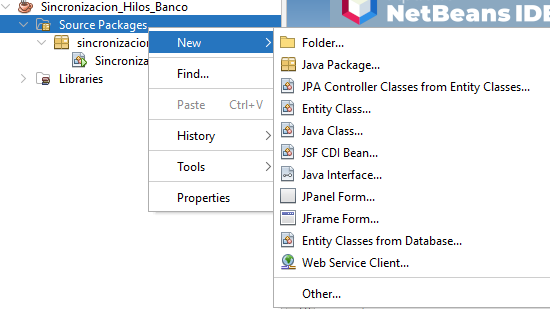


Figura 27 Creación de una nueva clase

1. Creamos un paquete llamado ec.edu.monster.controlador

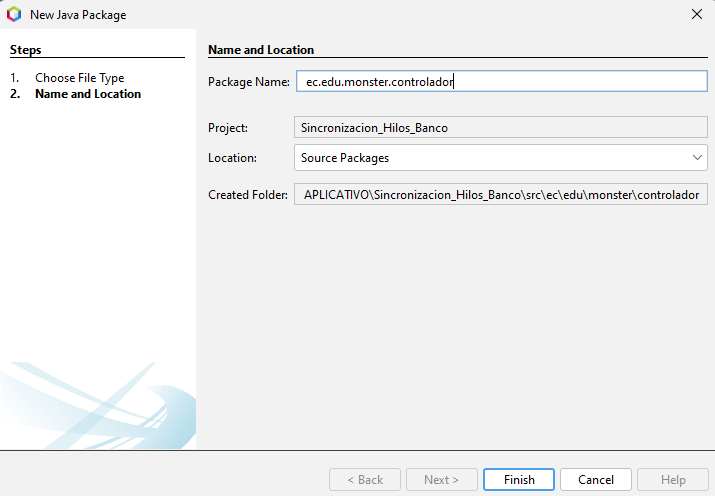
****

Figura 28 Crear un nuevo paquete

1. Creamos un paquete ec.edu.monster.modelo

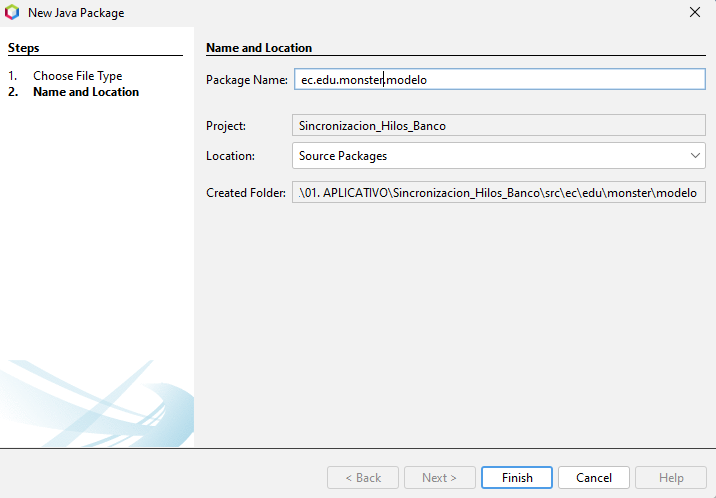


Figura 29 Creación del paquete modelo

1. En este proyecto en particular, no se implementarán vistas ya que se trata de un proyecto de consola, por lo tanto, toda la información se mostrará a través de la consola. Sin embargo, se crea el paquete correspondiente para mantener una estructura adecuada para el proyecto.

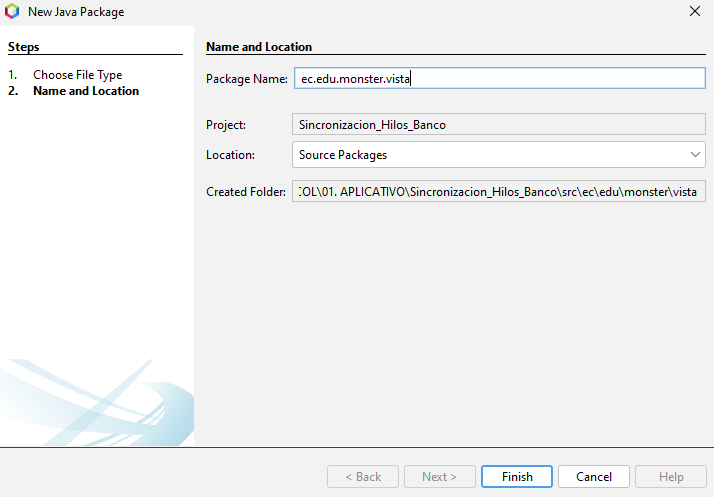


Figura 30 Creación del paquete vista

1. Realizaremos la creación de los paquetes necesarios para alcanzar la estructura. A continuación, se muestra el directorio final obtenido. En este caso, los paquetes se denominan "ec.edu.monster" y se aplica la estructura del patrón MVC:

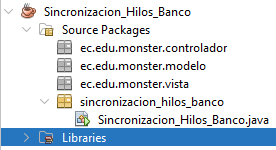


Figura 31 Estructura del proyecto MVC

# MODELO

## Creación del modelo Banco

Para generar el modelo en Apache NetBeans, basta con hacer clic derecho en la carpeta del modelo dentro del proyecto y seleccionar la opción "Agregar" (Add), seguida de "Clase" (Class). Esto abrirá un cuadro de diálogo donde se podrá ingresar el nombre de la nueva clase y definir sus atributos y métodos necesarios. La creación de esta clase es esencial para implementar la lógica y funcionalidad requerida en el modelo del proyecto en Apache NetBeans.

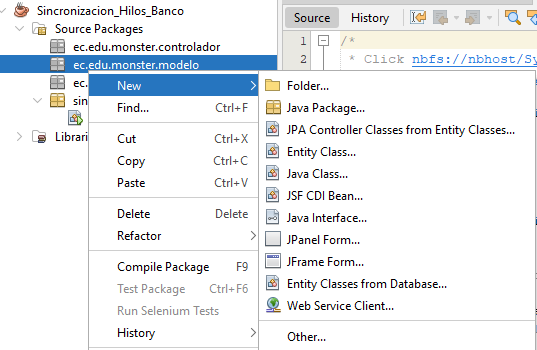


Figura 32 Creación de nueva clase

Luego de crear el modelo procedemos a atributo de tipo double para las cuentas



Figura 33 Atributos de la clase Banco

Además, hemos realizado modificaciones en el constructor del modelo de Banco para que ahora pueda crear 100 cuentas. Cada una de estas cuentas se inicializará con un saldo de 2000 dólares .

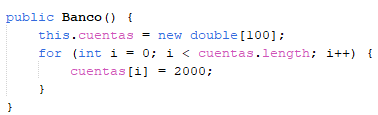


Figura 34 Inicialización de 100 cuentas en constructor

Con el fin de calcular el saldo total de las 100 cuentas creadas en el constructor, hemos implementado la función “getSaldoTotal”. Esta función recorre todas las cuentas y suma los saldos individuales de cada una de ellas.

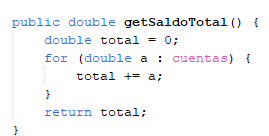


Figura 35 Creación de función getSaldoTotal

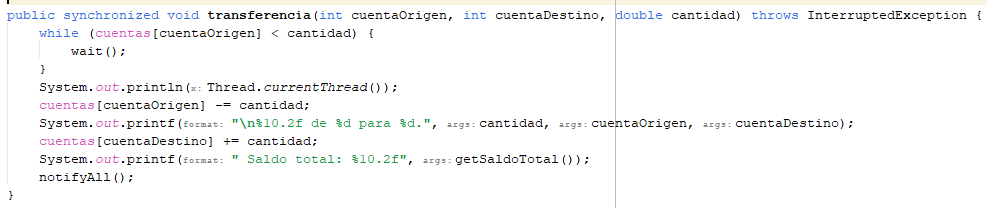
Dentro del método de transferencia, se lleva a cabo una simulación en la cual se realiza una transferencia de una cuenta de origen a una cuenta de destino. Se descuenta un monto especificado por parámetro y se muestra en pantalla el monto retirado, así como el saldo total de la cuenta de origen. Es fundamental tener en cuenta que estas cuentas se toman del arreglo creado en el constructor del modelo. 

Figura 36 Creación de función transferencia

Finalmente, tendríamos el código del modelo, de la siguiente manera.



Figura 37 Código del modelo Banco

A continuación, en la tabla 1, se presenta el código completo del modelo de Banco, incluyendo sus atributos y las funciones mencionadas anteriormente. Estas funciones son de gran utilidad para comprender la sincronización de los hilos de ejecución.

**Tabla 1**

Código del modelo 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;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

 \*

 \* @author Maycol

 \*/

public class Banco {

    private final double[] cuentas;

    private Lock cierreBanco = new ReentrantLock();

    public Banco() {

        this.cuentas = new double[100];

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

            cuentas[i] = 2000;

        }

    }

    public double getSaldoTotal() {

        double total = 0;

        for (double a : cuentas) {

            total += a;

        }

        return total;

    }

    public synchronized void transferencia(int cuentaOrigen, int cuentaDestino, double cantidad) throws InterruptedException {

        while (cuentas[cuentaOrigen] < cantidad) {

            wait();

        }

        System.out.println(Thread.currentThread());

        cuentas[cuentaOrigen] -= cantidad;

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

        cuentas[cuentaDestino] += cantidad;

        System.out.printf(" Saldo total: %10.2f", getSaldoTotal());

        notifyAll();

    }

}

# CONTROLADOR

## Creación del controlador EjecucionTransferencias

Para crear el controlador, se debe seguir el siguiente procedimiento: haciendo clic derecho en la carpeta de controladores y luego seleccionando la opción "Agregar" y posteriormente "Clase". Esto abrirá una ventana o diálogo donde se podrá especificar el nombre de la nueva clase y configurar sus atributos y métodos según sea necesario. La creación de esta nueva clase permitirá implementar la lógica y funcionalidad requerida para el controlador en el proyecto. Cabe mencionar que los pasos exactos pueden variar dependiendo del entorno de desarrollo o la herramienta utilizada, pero el objetivo es hacer clic derecho en la carpeta de controladores y seleccionar la opción para agregar una nueva clase.

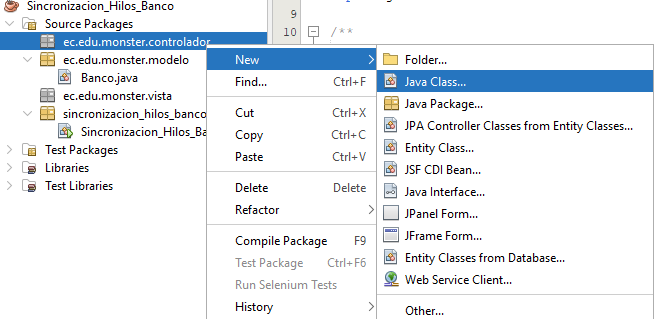


Figura 38 Creación de nueva clase

En este caso, nombraremos a la clase del controlador como “EjecucionTransferencias”. Para hacerlo, seleccionamos la opción “Clase” y, finalmente, damos clic en “Agregar”.

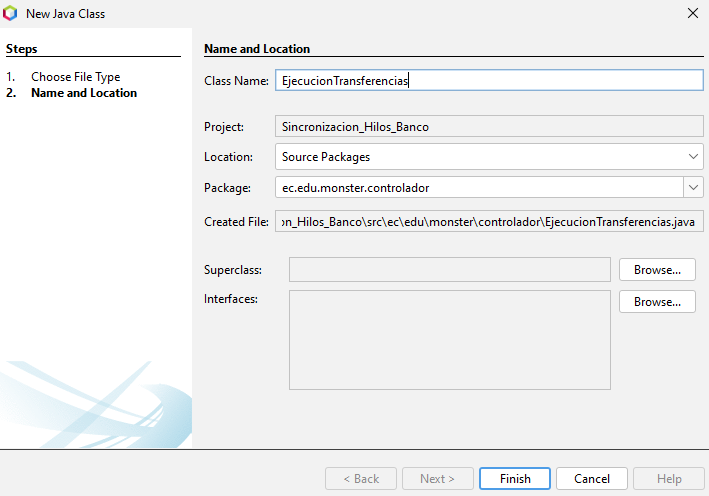
****

Figura 39 Asignación del nombre del controlador

El controlador "EjecucionTransferencias" contará con varios atributos. Estos incluirán un objeto de tipo Banco, que nos permitirá interactuar con las cuentas bancarias, un monto máximo para las transferencias, la cuenta de origen desde la cual se realizará la transferencia y el hilo con el cual el controlador trabajará.

Estos atributos son fundamentales para el funcionamiento del controlador, ya que nos proporcionan la información necesaria para llevar a cabo las transferencias bancarias de manera adecuada.

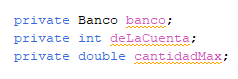


Figura 40 Atributos del controlador

El código mostrado implementa el método `run()` de un hilo en Java. Dentro del bucle while, genera valores aleatorios para la cuenta destino y la cantidad de dinero a transferir. Luego, utiliza el objeto `banco` para realizar la transferencia desde `deLaCuenta` hacia `paraLaCuenta` con el monto especificado. Después de cada transferencia, el hilo se duerme durante un tiempo aleatorio antes de repetir el proceso. En caso de que se produzca una interrupción (InterruptedException), se maneja sin realizar ninguna acción específica.

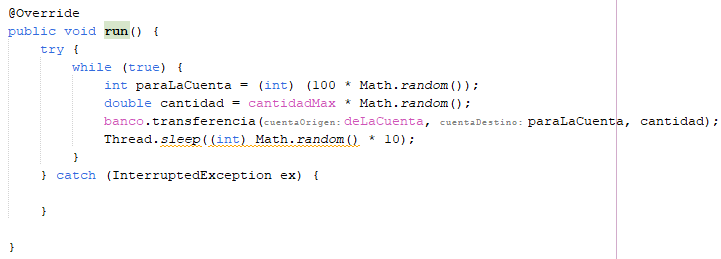


Figura 41 Función run

En el constructor del controlador, vamos a instanciar, tanto las variable de deLaCuenta, cantidadMax y banco, pero también vamos a instanciar el hilo con la función comenzar, para que de esta manera, cuando el hilo ejecute el método Start, esta función se ejecute.

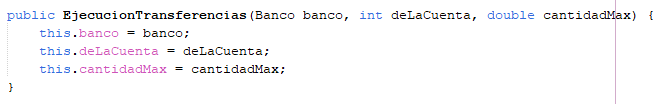


Figura 42 Creación del constructor de la clase del controlador

De esta manera, tenemos el código del controlador.



Figura 43 Código del controlador

**Tabla 2**

Código del controlador

/\*

 \* 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.espe.monster.controlador;

import ec.edu.espe.monster.modelo.Banco;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

 \*

 \* @author Erick

 \*/

public class EjecucionTransferencias implements Runnable {

    private Banco banco;

    private int deLaCuenta;

    private double cantidadMax;

    public EjecucionTransferencias(Banco banco, int deLaCuenta, double cantidadMax) {

        this.banco = banco;

        this.deLaCuenta = deLaCuenta;

        this.cantidadMax = cantidadMax;

    }

    @Override

    public void run() {

        try {

            while (true) {

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

                double cantidad = cantidadMax \* Math.random();

                banco.transferencia(deLaCuenta, paraLaCuenta, cantidad);

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

            }

        } catch (InterruptedException ex) {

        }

    }

}

## CÓDIGO FUENTE

### MAIN

En el archivo principal el método `main()` crea una instancia de la clase `Banco` llamada `b`. Luego, se ejecuta un bucle for que se repite 50 veces. En cada iteración, se crea una instancia de la clase `EjecucionTransferencias` pasando como argumentos el objeto `b`, el índice de iteración `i` y el monto máximo de transferencia de 2000. Luego, se crea un nuevo hilo (`Thread`) utilizando la instancia de `EjecucionTransferencias` y se inicia el hilo llamando al método `start()`. Esto crea 50 hilos simultáneos que ejecutarán la lógica de transferencia de manera concurrente en el objeto `b` del banco.

A continuación se muestra el código del main:

/\*

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

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

 \*/

package sincronizacion\_hilos\_banco;

import ec.edu.monster.controlador.EjecucionTransferencias;

import ec.edu.monster.modelo.Banco;

/\*\*

 \*

 \* @author Maycol

 \*/

public class Sincronizacion\_Hilos\_Banco {

    /\*\*

     \* @param args the command line arguments

     \*/

    public static void main(String[] args) {

        Banco b=new Banco();

        for (int i = 0; i < 50; i++) {

            EjecucionTransferencias r=new EjecucionTransferencias(b,i,2000);

            Thread t=new Thread(r);

            t.start();

        }

    }

}

# EJECUCIÓN DEL PROYECTO

## BANCO

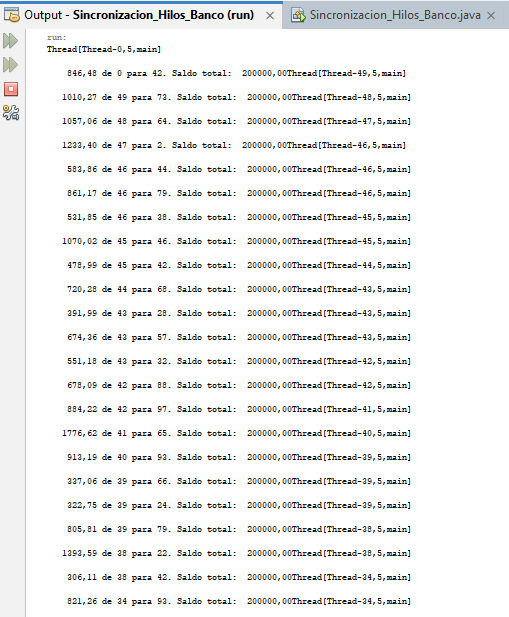


Figura 44 Aplicación del Banco

# 

# CONCLUSIONES

* El uso de la palabra clave `synchronized` es altamente recomendable cuando se necesita garantizar una única condición o restricción en un bloque de código compartido. Esto ayuda a evitar inconsistencias y conflictos entre hilos al acceder a recursos compartidos. Los hilos, por otro lado, mejoran la eficiencia de la comunicación entre programas en ejecución, ya que en la mayoría de los sistemas la comunicación entre procesos requiere la intervención del núcleo para proteger los recursos y facilitar la comunicación en sí misma. Al utilizar hilos, es posible lograr una comunicación más eficiente y una mejor utilización de los recursos del sistema.Los hilos aumentan la eficiencia de la comunicación entre programas en ejecución. En la mayoría de los sistemas en la comunicación entre procesos debe intervenir el núcleo para ofrecer protección de los recursos y realizar la comunicación misma.
* En este trabajo se ha desarrollado un modelo de Banco utilizando el lenguaje de programación Java. Este modelo incluye la creación de cuentas, transferencias y el cálculo del saldo total, brindando una base sólida para aplicaciones bancarias.
* La aplicación de conceptos de programación orientada a objetos, como la encapsulación, la herencia y el polimorfismo, ha facilitado la organización y el modularidad del código, mejorando su mantenibilidad y reutilización.
* El uso de herramientas como Apache NetBeans ha agilizado el desarrollo del proyecto, proporcionando un entorno de desarrollo integrado con características útiles para la codificación, depuración y gestión de proyectos Java.

# RECOMENDACIONES

* Se recomienda realizar pruebas exhaustivas del modelo para asegurarse de que funcione correctamente en diferentes escenarios y condiciones, validando tanto los casos normales como los casos límite y de error.
* Es importante aplicar buenas prácticas de programación, como utilizar nombres descriptivos, seguir convenciones de estilo de código y documentar adecuadamente el código, para facilitar su comprensión y mantenimiento.
* Tener cuidado del número de hilos que queramos crear ya que si creamos demasiados podríamos afectar nuestro computador
* Es recomendable nombrar la carpeta con el nombre especificado al inicio de la materia para así tener un orden.
* Se recomienda ver el video provisto por el docente para un mayor entendimiento del programa.

# BIBLIOGRAFÍA

[1] Java, "¿Qué es la tecnología Java y por qué la necesito?". Java. Disponible en: https://www.java.com/es/download/help/whatis\_java.html. [Último acceso: 27 noviembre 2022].

[2] M. "MONOTAREA Y MULTITAREA". Wordpress. Disponible en: https://marianogm17.wordpress.com/monotarea-y-multitarea/. [Último acceso: 27 noviembre 2022].

[3] KeepCoding. "¿Qué es la programación concurrente?". KeepCoding, 11 octubre 2022. Disponible en: https://keepcoding.io/blog/que-es-la-programacion-concurrente/. [Último acceso: 27 noviembre 2022].

[4] JavaTPoint. "Tutorial de Java AWT". JavaTPoint. Disponible en: https://www.javatpoint.com/java-awt. [Último acceso: 27 noviembre 2022].

[5] Open Bootcamp. "Introducción a Swing en Java". Open Bootcamp. Disponible en: https://open-bootcamp.com/cursos/java/introduccion-a-swing. [Último acceso: 27 noviembre 2022].

[6] Universidad de Alicante. "Hilos". Experto Java. Disponible en: http://www.jtech.ua.es/dadm/restringido/java/sesion05-apuntes.html#:~:text=Un%20hilo%20es%20un%20flujo,programa%2C%20pila%20de%20ejecuci%C3%B3n). [Último acceso: 27 noviembre 2022].

[7] Departamento de Informática de la Universidad de Valladolid. "Departamento de Informática de la Universidad de Valladolid". Disponible en: https://www.infor.uva.es/~fdiaz/sd/doc/hilos#:~:text=La%20interface%20Runnable%20proporciona%20un,deba%20extender%20alguna%20otra%20clase.. [Último acceso: 27 noviembre 2022].

[8] IBM. "Excepciones Java". IBM, 08 marzo 2021. Disponible en: https://www.ibm.com/docs/es/i/7.1?topic=driver-java-exceptions. [Último acceso: 27 noviembre 2022].