

**Tema**

HILOS EN JAVA NETBEANS

**Estudiantes**

Johao Morales

Maycol Tituaña

Alex Velasteguí

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS.MDU.CCNA.CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

08/07/2023

CONTENIDO

[INTRODUCCIÓN 4](#_Toc139749654)

[OBJETIVO 4](#_Toc139749655)

[MARCO TEÓRICO 5](#_Toc139749656)

[¿QUÉ ES UNA APLICACIÓN? 5](#_Toc139749657)

[MÁQUINA VIRTUAL DE JAVA 5](#_Toc139749658)

[¿QUÉ ES EL BYTECODE? 6](#_Toc139749659)

[¿QUÉ ES LA SERIALIZACIÓN? 7](#_Toc139749660)

[JAVA 7](#_Toc139749661)

[MVC 8](#_Toc139749662)

[HILOS 9](#_Toc139749663)

[**HILOS EN JAVA** 10](#_Toc139749664)

[**ESTADOS DE LOS HILOS JAVA** 10](#_Toc139749665)

[**SINCRONIZACIÓN DE HILOS** 11](#_Toc139749666)

[DESARROLLO E IMPLEMENTACIÓN DEL PROYECTO 11](#_Toc139749667)

[ENTORNO DE DESARROLLO 11](#_Toc139749668)

[CREACIÓN E IMPLEMENTACIÓN DEL PROYECTO 12](#_Toc139749669)

[CONCLUSIONES 26](#_Toc139749670)

[RECOMENDACIONES 26](#_Toc139749671)

[BIBLIOGRAFÍA 27](#_Toc139749672)

**ÍNDICE DE IMÁGENES**

[Figura 1.. Máquina virtual de Java 6](#_Toc139798680)

[Figura 2. Serialización de Java 7](#_Toc139798681)

[Figura 3. Patrón MVC 9](#_Toc139798682)

[Figura 4. Diagrama de transición de estado de un Hilo 11](#_Toc139798683)

[Figura 5. Entorno de Desarrollo 11](#_Toc139798684)

[Figura 6. Creación de un Nuevo Proyecto 12](#_Toc139798685)

[Figura 7.Selección de creación de un Nuevo Proyecto 12](#_Toc139798686)

[Figura 8. Nombre y Ubicación para la creación de un Nuevo Proyecto 13](#_Toc139798687)

[Figura 9. Creación de Paquetes 13](#_Toc139798688)

[Figura 10. Creación de Paquete Controlador 14](#_Toc139798689)

[Figura 11. Creación de Clase en el Paquete Controlador 14](#_Toc139798690)

[Figura 12. Nombrar la Clase a crear en el Paquete Controlador 15](#_Toc139798691)

[Figura 13.Implementación de la Clase Controlador 15](#_Toc139798692)

[Figura 14. Creación de Paquete Vista 17](#_Toc139798693)

[Figura 15. Creación de Clase en el Paquete Vista 17](#_Toc139798694)

[Figura 16. Nombrar la Clase a crear en el Paquete Vista 17](#_Toc139798695)

[Figura 17. Implementación de la Clase Vista 18](#_Toc139798696)

[Figura 18. Creación de Paquete Main 19](#_Toc139798697)

[Figura 19. Creación de Clase en el Paquete Main 19](#_Toc139798698)

[Figura 20. Implementación de la Clase Main 20](#_Toc139798699)

[Figura 21. Creación de Paquete Modelo 21](#_Toc139798700)

[Figura 22. Creación de Clase en el Paquete Modelo 21](#_Toc139798701)

[Figura 23. Nombrar la Clase como Hilos Varios 22](#_Toc139798702)

[Figura 24. Implementación de la Clase Hilos Varios 22](#_Toc139798703)

[Figura 25. Nombrar la Clase como Hilos Varios2 23](#_Toc139798704)

[Figura 26. Implementación de la Clase Hilos Varios 24](#_Toc139798705)

[Figura 27.Implementación de la Clase Hilos Varios 25](#_Toc139798706)

[Figura 28. Implementación de la Clase Hilos Varios 26](#_Toc139798707)

**ÍNDICE DE TABLAS**

[Tabla 1. Codificación de la Clase Controlador 15](#_Toc139798670)

[Tabla 2. Codificación de la Clase Vista 17](#_Toc139798671)

[Tabla 3. Codificación de la Clase Main 19](#_Toc139798672)

[Tabla 4. Codificación de la Clase Hilos Varios 22](#_Toc139798673)

[Tabla 5.Codificación de la Clase Hilos Varios2 24](#_Toc139798674)

# INTRODUCCIÓN

En el campo de la informática, a medida que los ordenadores se han vuelto más poderosos, ha surgido la necesidad de ejecutar tareas concurrentemente para aprovechar al máximo su capacidad. En lugar de ejecutar instrucciones de programación de forma individual y secuencial, se ha vuelto crucial poder realizar múltiples tareas al mismo tiempo.

La sincronización de hilos es una técnica fundamental para lograr la ejecución concurrente de tareas en un sistema informático. Los hilos son unidades de ejecución que representan secuencias de instrucciones que pueden ejecutarse de forma independiente y simultánea dentro de un programa. Al utilizar hilos, podemos descomponer nuestro programa en subtareas más pequeñas y ejecutarlas en paralelo, lo que resulta en un aumento significativo del rendimiento.

Cuando empaquetamos nuestro trabajo en serie, es decir, cuando ejecutamos todas las tareas de forma secuencial, estamos desaprovechando el potencial de los ordenadores modernos. Si todas nuestras tareas se ejecutan de esta manera, es probable que el rendimiento sea reducido y que no estemos utilizando el computador al máximo de sus capacidades.

La capacidad de un sistema informático para realizar multitareas nos permite considerar nuestro programa como una colección de subtareas. Cada una de estas subtareas puede ser ejecutada en un hilo separado, lo que permite que los procesos se ejecuten simultáneamente en un único procesador. Esto resulta en un aumento significativo del rendimiento, ya que las tareas pueden avanzar en paralelo y no tienen que esperar a que se completen otras tareas.

Sin embargo, la sincronización de hilos presenta desafíos adicionales. Cuando múltiples hilos comparten recursos comunes, como memoria o variables, es necesario garantizar que las operaciones se realicen de manera ordenada y coherente. De lo contrario, pueden ocurrir condiciones de carrera, donde el resultado final depende del orden en que los hilos accedan y modifiquen los recursos compartidos.

Para resolver estos problemas de sincronización, se utilizan técnicas como los semáforos, los monitores, los cerrojos y las variables condicionales. Estos mecanismos permiten establecer reglas y restricciones en el acceso a los recursos compartidos, asegurando que los hilos se sincronicen adecuadamente y evitando condiciones de carrera.

En resumen, la sincronización de hilos es una técnica esencial en la programación concurrente, que nos permite aprovechar al máximo la capacidad de los ordenadores modernos. Al descomponer nuestro programa en subtareas ejecutadas en hilos separados y sincronizar su acceso a los recursos compartidos, podemos lograr un mayor rendimiento y una ejecución más eficiente de nuestras aplicaciones.

# OBJETIVO

El objetivo de esta práctica de laboratorio es aplicar los conceptos de hilos en Java y el patrón de arquitectura MVC (Modelo-Vista-Controlador) utilizando el entorno de desarrollo NetBeans. A través de esta práctica, se pretende lograr una mejor comprensión de cómo utilizar hilos para ejecutar tareas concurrentes en un programa Java y cómo implementar la estructura MVC para separar la lógica de negocio, la presentación y la interacción en un proyecto.

Al finalizar esta práctica, se espera haber adquirido las siguientes habilidades:

1. Crear y administrar hilos en Java utilizando las clases Thread o implementando la interfaz Runnable.
2. Comprender y aplicar los estados de los hilos y las técnicas de sincronización para evitar problemas de concurrencia.
3. Diseñar y desarrollar la arquitectura MVC en una aplicación utilizando NetBeans como IDE de desarrollo.
4. Separar y organizar la lógica de negocio, la presentación y la interacción en componentes independientes del modelo, la vista y el controlador.
5. Implementar la comunicación y la actualización de datos entre el modelo, la vista y el controlador, siguiendo las reglas y principios del patrón MVC.

Al completar esta práctica, los participantes tendrán una comprensión más sólida de la programación con hilos y la implementación del patrón MVC en Java, lo que les permitirá desarrollar aplicaciones más eficientes, modulares y mantenibles.

# MARCO TEÓRICO

## ¿QUÉ ES UNA APLICACIÓN?

Una aplicación, en el contexto tecnológico, es un programa de software diseñado para realizar tareas específicas en dispositivos electrónicos como teléfonos inteligentes, tabletas, computadoras y otros dispositivos. Las aplicaciones, también conocidas como apps, pueden tener diferentes propósitos y funcionalidades, desde permitir la comunicación, la productividad, el entretenimiento, hasta brindar servicios específicos como el acceso a noticias, la gestión financiera, la edición de fotos, entre muchas otras posibilidades.

Las aplicaciones se desarrollan utilizando lenguajes de programación y frameworks adecuados para la plataforma en la que se ejecutarán, como iOS, Android, Windows, macOS, entre otros. Estas aplicaciones se distribuyen a través de tiendas en línea o se pueden descargar directamente desde sitios web.

En resumen, una aplicación es un programa informático diseñado para realizar funciones específicas en dispositivos electrónicos, brindando a los usuarios una amplia gama de utilidades y servicios.

## MÁQUINA VIRTUAL DE JAVA

La máquina virtual de Java™ (JVM, por sus siglas en inglés) es un entorno de ejecución que permite ejecutar aplicaciones escritas en el lenguaje de programación Java. Actúa como una capa de abstracción entre el código fuente de Java y el sistema operativo subyacente, lo que proporciona portabilidad y compatibilidad multiplataforma para las aplicaciones Java.

La JVM interpreta y ejecuta el bytecode generado por el compilador de Java, que es un código intermedio producido a partir del código fuente Java. La JVM es responsable de gestionar la memoria, realizar la administración de recursos y ejecutar el programa Java de manera eficiente.

Además de la ejecución de aplicaciones, la JVM también incluye otros componentes esenciales, como el recolector de basura (garbage collector), que se encarga de liberar la memoria utilizada por objetos no referenciados, y el cargador de clases (class loader), que carga dinámicamente las clases necesarias durante la ejecución del programa.

En resumen, una máquina virtual de Java es un entorno de ejecución que interpreta y ejecuta aplicaciones escritas en el lenguaje de programación Java, proporcionando portabilidad y compatibilidad multiplataforma a través del uso de bytecode y gestionando aspectos cruciales como la memoria y la carga dinámica de clases.

El cargador de clases y el gestor de seguridad, que forman parte del entorno de tiempo de ejecución Java, aíslan el código que proviene de otra plataforma. También pueden restringir a qué recursos del sistema puede acceder cada una de las clases que se cargan (IBM, 2021).

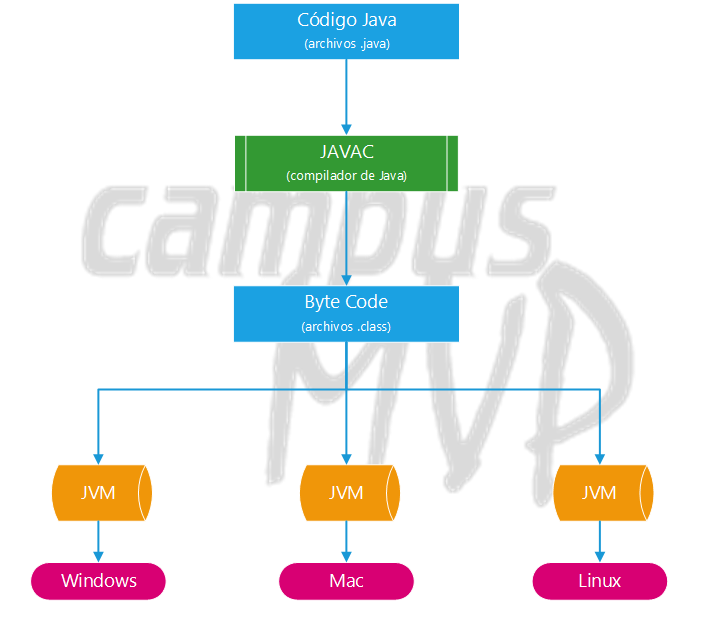


Figura 1.. Máquina virtual de Java

## ¿QUÉ ES EL BYTECODE?

El bytecode es un código intermedio generado por el compilador de un lenguaje de programación como Java. Es un conjunto de instrucciones de bajo nivel que se encuentra entre el código fuente legible por humanos y el código máquina ejecutable por la computadora.

Cuando se escribe un programa en un lenguaje como Java, se utiliza un compilador para traducir el código fuente a bytecode en lugar de traducirlo directamente a código máquina. El bytecode está diseñado para ser independiente de la plataforma, lo que significa que se puede ejecutar en cualquier sistema operativo o arquitectura de hardware que tenga una máquina virtual capaz de interpretarlo.

La máquina virtual de Java (JVM) es responsable de interpretar el bytecode y ejecutar el programa Java correspondiente. La JVM es específica de cada plataforma y se encarga de traducir el bytecode en instrucciones específicas del sistema operativo y la arquitectura de hardware subyacente.

El uso del bytecode en Java proporciona varias ventajas, como la portabilidad de las aplicaciones y la capacidad de ejecutar programas en diferentes entornos sin necesidad de recompilar el código fuente original. Además, el bytecode ofrece un nivel de abstracción que facilita el desarrollo y la depuración de aplicaciones.

En resumen, el bytecode es un código intermedio generado por el compilador de Java que se encuentra entre el código fuente y el código máquina. Es interpretado por la JVM y proporciona portabilidad y abstracción en el desarrollo y la ejecución de aplicaciones Java.

## ¿QUÉ ES LA SERIALIZACIÓN?

La serialización es el proceso de convertir un objeto en una secuencia de bytes para que pueda ser almacenado, transmitido o guardado en un medio de almacenamiento persistente. En el contexto de la programación, se refiere a la capacidad de convertir objetos en una forma que se pueda almacenar o transmitir y luego restaurarlos a su estado original.

Durante la serialización, los objetos se convierten en una secuencia de bytes que representan su estado actual, incluyendo sus datos y estructura interna. Esta secuencia de bytes se puede guardar en un archivo, transmitir a través de una red o almacenar en una base de datos. Posteriormente, durante la deserialización, la secuencia de bytes se convierte nuevamente en un objeto con su estado original.

La serialización es utilizada en diferentes escenarios, como la persistencia de datos en aplicaciones, el intercambio de información entre sistemas distribuidos o la comunicación entre procesos. Algunos lenguajes de programación, como Java, proporcionan mecanismos integrados para la serialización de objetos, facilitando el proceso.

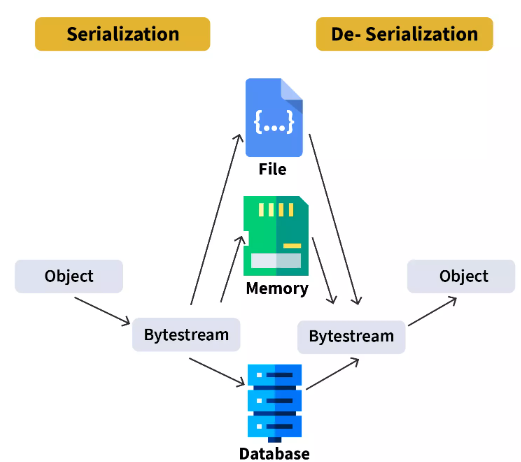


Figura 2. Serialización de Java

## JAVA

Java es un lenguaje de programación de alto nivel y orientado a objetos que fue creado por Sun Microsystems (ahora propiedad de Oracle) en la década de 1990. Es uno de los lenguajes más populares y ampliamente utilizados en la industria del desarrollo de software.

Java fue diseñado con el objetivo de ser un lenguaje portátil, seguro y robusto. Su principal característica es la "Write once, run anywhere" (escribir una vez, ejecutar en cualquier lugar), lo que significa que el código escrito en Java puede ejecutarse en diferentes plataformas sin necesidad de realizar modificaciones significativas. Esto es posible gracias al uso de la máquina virtual de Java (JVM), que interpreta el bytecode generado por el compilador de Java y lo ejecuta en el sistema operativo subyacente.

Además de ser un lenguaje versátil, Java también se destaca por su enfoque en la programación orientada a objetos. Proporciona características como herencia, encapsulamiento, polimorfismo y abstracción, lo que facilita la creación de aplicaciones modulares, extensibles y fáciles de mantener.

Java se utiliza en una amplia variedad de aplicaciones, desde desarrollo de aplicaciones de escritorio y web, hasta aplicaciones móviles, sistemas embebidos, juegos y aplicaciones empresariales. También es ampliamente utilizado en el desarrollo de software para servidores y en la creación de aplicaciones empresariales de gran escala.

## MVC

MVC es un acrónimo que se refiere al patrón de diseño arquitectónico Modelo-Vista-Controlador (Model-View-Controller, en inglés). Es ampliamente utilizado en el desarrollo de aplicaciones de software para separar y organizar la lógica de la aplicación en componentes independientes y facilitar el mantenimiento, la escalabilidad y la reutilización del código.

El patrón MVC divide una aplicación en tres componentes principales:

Modelo (Model): Representa los datos y la lógica de negocio de la aplicación. El modelo es responsable de gestionar los datos, realizar operaciones de validación y cálculo, y proporcionar interfaces para acceder y modificar los datos.

Vista (View): Es la capa de presentación de la aplicación. La vista se encarga de mostrar los datos al usuario y proporciona una interfaz gráfica o una representación visual de la información contenida en el modelo. La vista no contiene lógica de negocio, sino que se limita a presentar los datos de una manera comprensible para el usuario.

Controlador (Controller): Actúa como intermediario entre el modelo y la vista. El controlador recibe las interacciones del usuario a través de la vista y coordina las acciones correspondientes en el modelo. También se encarga de actualizar la vista cuando los datos del modelo cambian, asegurando así una comunicación bidireccional entre el modelo y la vista.

La separación de responsabilidades entre el modelo, la vista y el controlador permite una mayor modularidad y facilita el desarrollo colaborativo. Además, este enfoque hace que la aplicación sea más fácil de mantener, ya que los cambios en una parte del MVC no afectarán directamente a las otras partes.

El patrón MVC se utiliza en numerosos frameworks y tecnologías de desarrollo de software, como Ruby on Rails, ASP.NET MVC, Django y Spring MVC, entre otros.

En resumen, el patrón de diseño MVC es una forma organizada de estructurar y separar las diferentes responsabilidades de una aplicación, dividiéndola en los componentes Modelo, Vista y Controlador. Esta separación mejora la modularidad, la reutilización y la mantenibilidad del código.

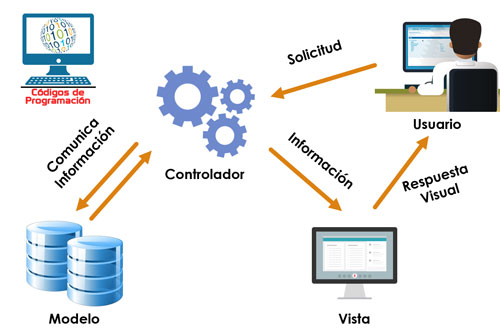


Figura 3. Patrón MVC

## HILOS

Un hilo en el contexto del software se refiere a una unidad básica de ejecución que representa un flujo de control independiente dentro de un programa. Es un concepto ampliamente utilizado en programación concurrente y paralela para lograr la ejecución simultánea de tareas.

Un hilo se puede considerar como una secuencia de instrucciones que se ejecutan en paralelo con otros hilos en un programa. Cada hilo tiene su propia pila de ejecución, que almacena variables locales y registros de ejecución. Los hilos comparten el mismo espacio de direcciones y recursos del proceso, lo que les permite comunicarse y cooperar entre sí.

La utilización de hilos en un programa puede brindar varios beneficios, como:

1. Multitarea: Los hilos permiten que diferentes partes de un programa se ejecuten simultáneamente, lo que puede mejorar la capacidad de respuesta y la eficiencia en la ejecución de tareas.

2. Concurrencia: Los hilos permiten que múltiples tareas se realicen en paralelo, lo que puede ser útil en situaciones donde se requiere una interacción rápida y simultánea con el usuario o con otros componentes del sistema.

3. Paralelismo: Al utilizar hilos en sistemas multiprocesador o multinúcleo, se puede lograr el procesamiento paralelo de tareas, lo que puede acelerar la ejecución de programas que se pueden dividir en subprocesos independientes.

Sin embargo, la programación con hilos también puede presentar desafíos, como condiciones de carrera (race conditions) y problemas de sincronización, que requieren un manejo cuidadoso mediante técnicas adecuadas de sincronización y exclusión mutua.

### **HILOS EN JAVA**

En Java, un hilo (thread, en inglés) es una unidad básica de ejecución que representa un flujo independiente de control dentro de un programa. Un hilo permite que un programa realice múltiples tareas simultáneamente, dividiendo la ejecución en fragmentos más pequeños y ejecutándolos en paralelo.

Cada programa Java tiene al menos un hilo principal, que es el hilo desde el cual se inicia la ejecución del programa. Sin embargo, es posible crear hilos adicionales para realizar tareas simultáneas.

La creación y gestión de hilos en Java se logra mediante la utilización de la clase Thread o mediante la implementación de la interfaz Runnable. La clase Thread proporciona métodos para iniciar, detener, suspender y reanudar hilos, así como para coordinar su ejecución y comunicación entre ellos.

Los hilos en Java se ejecutan en concurrencia y pueden compartir recursos y datos. Por lo tanto, es importante implementar mecanismos adecuados de sincronización y exclusión mutua para evitar problemas como las condiciones de carrera (race conditions) y garantizar la consistencia de los datos compartidos entre hilos.

La programación con hilos en Java permite aprovechar el potencial de los sistemas multiprocesador y multinúcleo, así como mejorar la capacidad de respuesta y el rendimiento de las aplicaciones al realizar tareas concurrentes.

### **ESTADOS DE LOS HILOS JAVA**

En Java, los hilos (threads) pueden estar en varios estados diferentes durante su ciclo de vida. Los estados de los hilos en Java son:

1. Nuevo (New): El hilo ha sido creado pero aún no se ha iniciado su ejecución.

2. Ejecutable (Runnable): El hilo está listo para ser ejecutado y está compitiendo por el tiempo de CPU disponible. Puede estar en ejecución o en espera para ser planificado por el planificador del sistema operativo.

3. Bloqueado (Blocked): El hilo está esperando ciertos recursos o condiciones para continuar su ejecución. Puede estar bloqueado debido a operaciones de entrada/salida (I/O), adquisición de bloqueos (locks) u otras situaciones en las que no puede continuar su ejecución.

4. Esperando (Waiting): El hilo está en espera hasta que se cumpla una condición específica. Puede ser debido a una llamada al método wait(), donde el hilo está esperando que otro hilo le notifique para continuar, o debido a una espera en una cola o en un objeto monitor.

5. Esperando tiempo (Timed Waiting): Similar al estado de Esperando, pero con un tiempo de espera definido. El hilo permanecerá en este estado hasta que expire el tiempo especificado o se cumpla una condición para su continuación.

6. Terminado (Terminated): El hilo ha finalizado su ejecución, ya sea porque ha completado su tarea o porque se ha producido una excepción que terminó su ejecución de forma anormal.

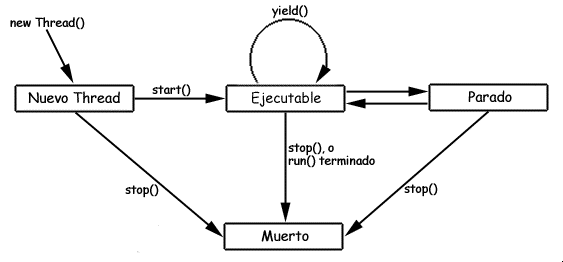


Figura 4. Diagrama de transición de estado de un Hilo

### **SINCRONIZACIÓN DE HILOS**

La sincronización de hilos en Java se refiere al mecanismo que permite controlar el acceso y la manipulación de datos compartidos por múltiples hilos. La sincronización garantiza que solo un hilo pueda acceder a un recurso compartido en un momento dado, evitando condiciones de carrera y asegurando la consistencia de los datos.

La sincronización se logra mediante el uso de bloques sincronizados o métodos sincronizados que establecen un bloqueo (lock) en un objeto compartido. Cuando un hilo adquiere el bloqueo, se le concede acceso exclusivo al recurso compartido. Los demás hilos que intentan acceder al recurso bloqueado deben esperar hasta que el bloqueo se libere.

El uso adecuado de la sincronización evita problemas de concurrencia, como la lectura y escritura simultánea de datos compartidos, y garantiza la atomicidad de las operaciones críticas. Además, la sincronización también implica la visibilidad de los cambios realizados por un hilo a otros hilos, lo que asegura que los cambios en los datos sean reflejados correctamente en todos los hilos.

Es importante tener en cuenta que un uso incorrecto o excesivo de la sincronización puede llevar a problemas de rendimiento y a situaciones de bloqueo (deadlock) si no se manejan adecuadamente.

# DESARROLLO E IMPLEMENTACIÓN DEL PROYECTO

## ENTORNO DE DESARROLLO

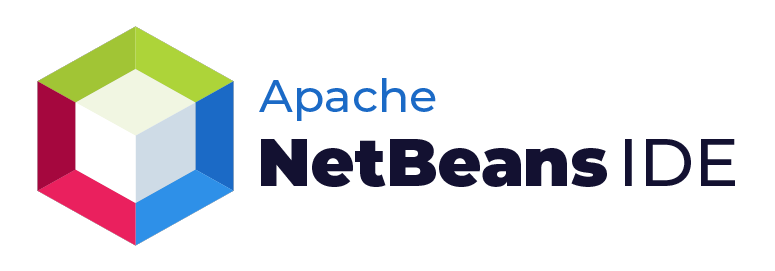


Figura 5. Entorno de Desarrollo

Apache NetBeans es un entorno de desarrollo integrado de código abierto que proporciona herramientas y funcionalidades para facilitar el desarrollo de aplicaciones Java y otros lenguajes. Su arquitectura modular y su conjunto de características potentes hacen que sea una opción popular para los desarrolladores que buscan un entorno de desarrollo eficiente y personalizable.

## CREACIÓN E IMPLEMENTACIÓN DEL PROYECTO

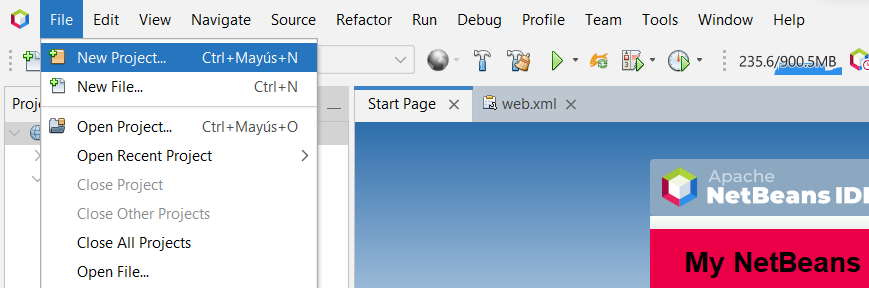
Primero Ingresas al IDE de Desarrollo APACHE NETBEANS y creas un nuevo proyecto:  


Figura 6. Creación de un Nuevo Proyecto

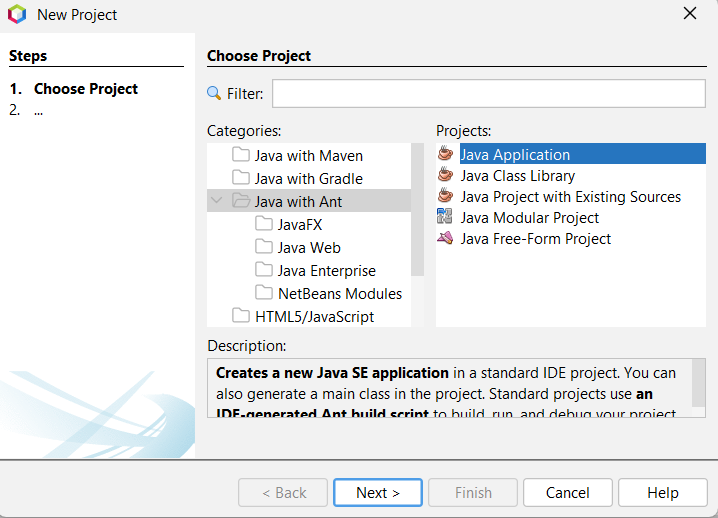
Después, dentro de la interfaz que ofrece la opción de crear un nuevo proyecto seleccionas la categoría Java with Ant y en Proyectos Java Application, para así proceder a dar Next:  


Figura 7.Selección de creación de un Nuevo Proyecto

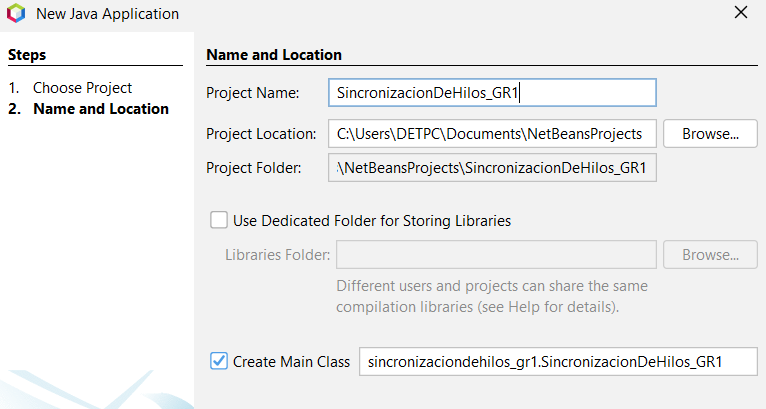
Asignas un nombre al proyecto y seleccionas la Ubicación en donde quieress que se cree el proyecto y lo generas:  


Figura 8. Nombre y Ubicación para la creación de un Nuevo Proyecto

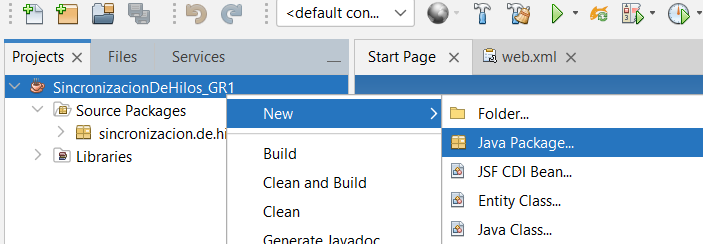
Dentro de la interfaz del proyecto creas los paquetes en base al Patrón MVC y un paquete extra llamado main, dentro de estos vas a crear las clases necesarias para continuar con el desarrollo del proyecto así mismo se presentará el código necesario para cada clase  


Figura 9. Creación de Paquetes

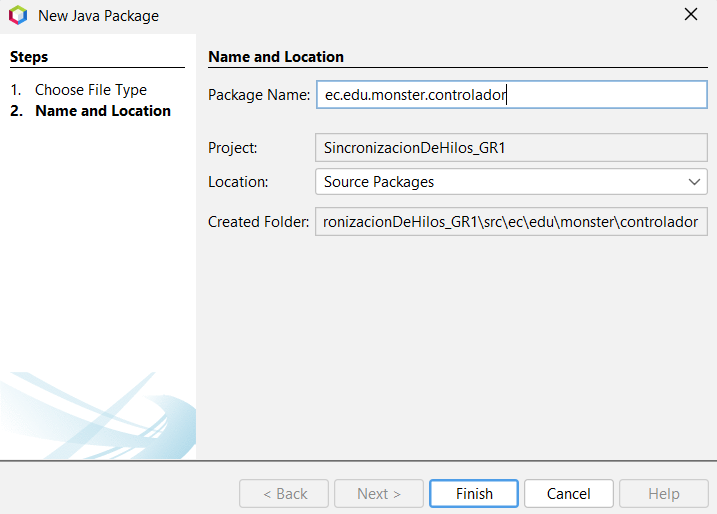
Creas el paquete controlador:  


Figura 10. Creación de Paquete Controlador

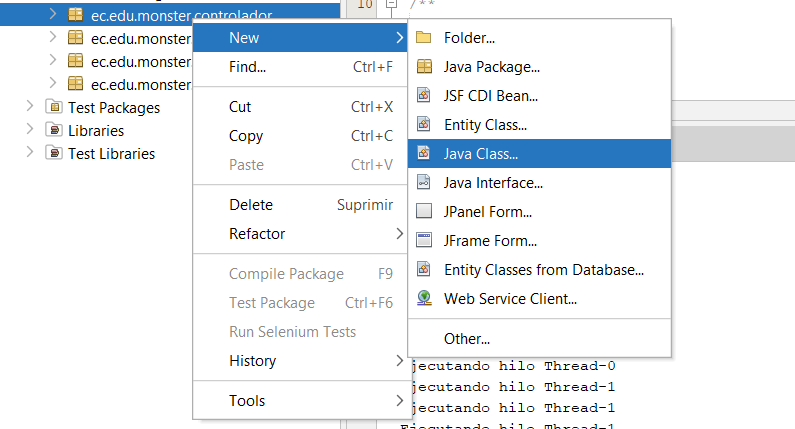
Dentro de este creas un JAVA CLASS, al que vas a llamar con el mismo nombre del paquete:  


Figura 11. Creación de Clase en el Paquete Controlador

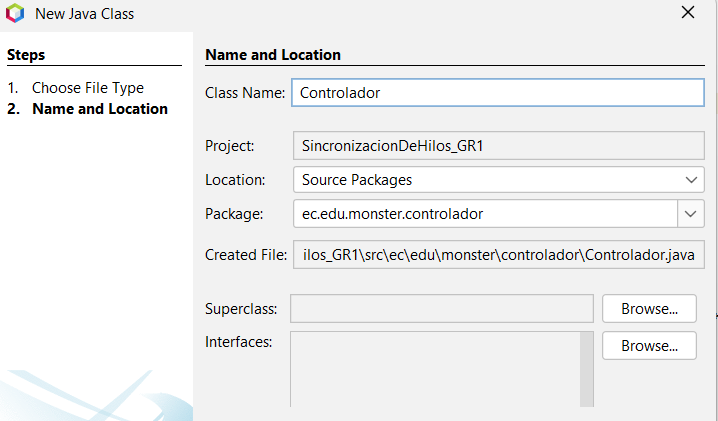


Figura 12. Nombrar la Clase a crear en el Paquete Controlador

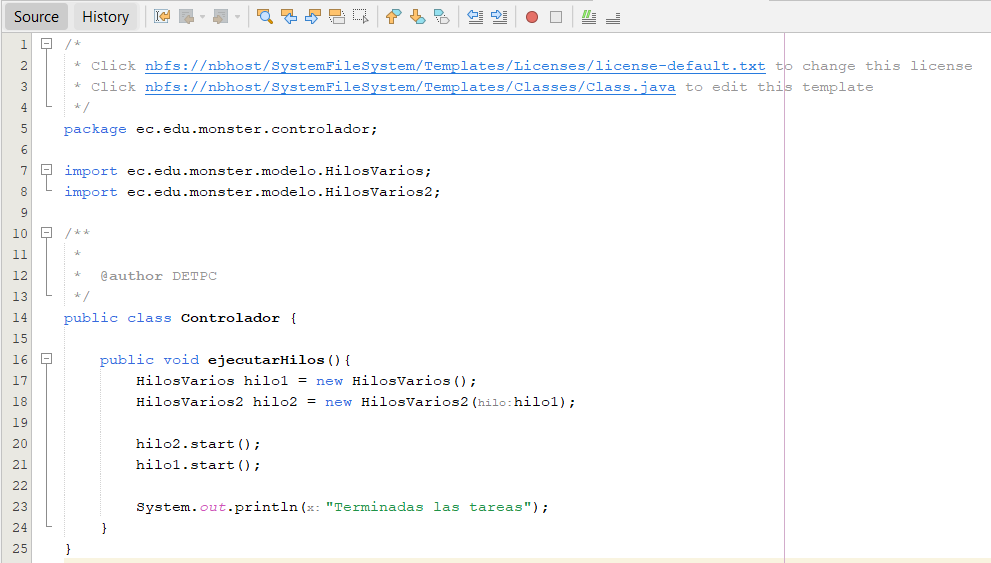
Por fines didácticos se adjuntará una captura del código necesario para esta clase, como también un cuadro de texto con el código para que pueda ser copiado con mayor facilidad:  


Figura 13.Implementación de la Clase Controlador

Código de la Clase 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.monster.controlador;

**import** **ec.edu.monster.modelo.HilosVarios**;

**import** **ec.edu.monster.modelo.HilosVarios2**;

/\*\*

\*

\* @author DETPC

\*/

**public** **class** **Controlador** {

**public** **void** **ejecutarHilos**(){

HilosVarios hilo1 = **new** HilosVarios();

HilosVarios2 hilo2 = **new** HilosVarios2(hilo1);

hilo2.start();

hilo1.start();

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

}

}

Tabla 1. Codificación de la Clase Controlador

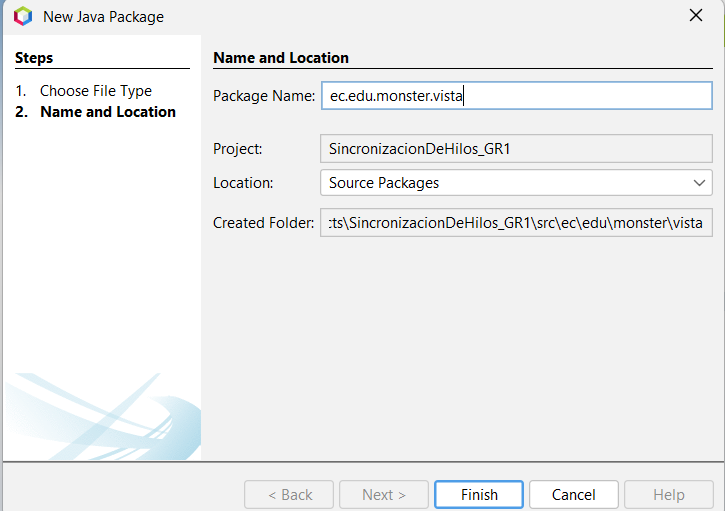
Creas el paquete Vista:  


Figura 14. Creación de Paquete Vista

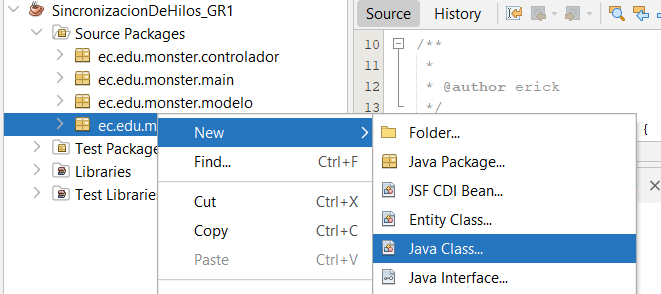
Dentro de este creas un JAVA CLASS, al que vas a llamar con el mismo nombre del paquete:  


Figura 15. Creación de Clase en el Paquete Vista

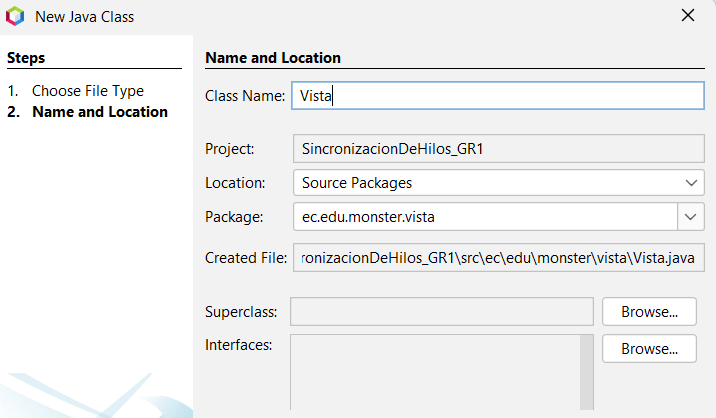


Figura 16. Nombrar la Clase a crear en el Paquete Vista

Por fines didácticos se adjuntará una captura del código necesario para esta clase, como también un cuadro de texto con el código para que pueda ser copiado con mayor facilidad:  


Figura 17. Implementación de la Clase Vista

Código de la Clase Vista:

/\*

\* 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** **java.util.logging.Level**;

**import** **java.util.logging.Logger**;

/\*\*

\*

\* @author DETPC

\*/

**public** **class** **Vista** **extends** Thread{

**public** **void** **monstarHilos**(String nombreHilo) {

**for** (**int** i = **0**; i < **8**; i++) {

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

**try** {

Thread.sleep(**400**);

} **catch** (InterruptedException ex) {

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

}

}

}

}

Tabla 2. Codificación de la Clase Vista

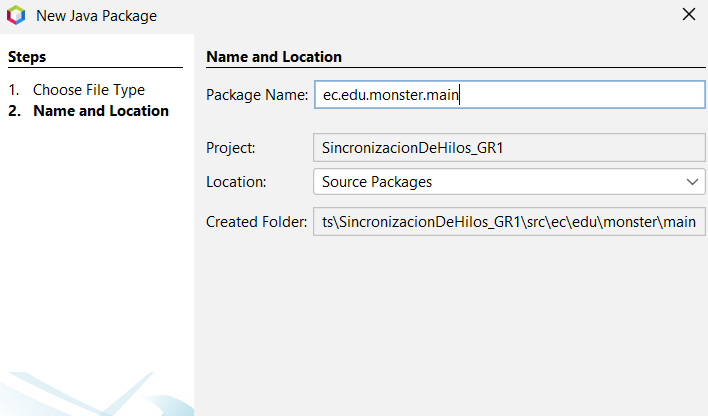
Creamos el paquete Main:  


Figura 18. Creación de Paquete Main

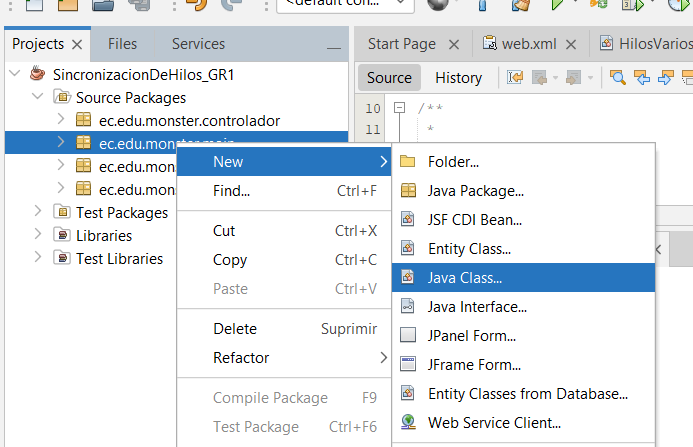
Dentro de este creas un JAVA CLASS, al que vamos a llamar con el mismo nombre del paquete:  


Figura 19. Creación de Clase en el Paquete Main

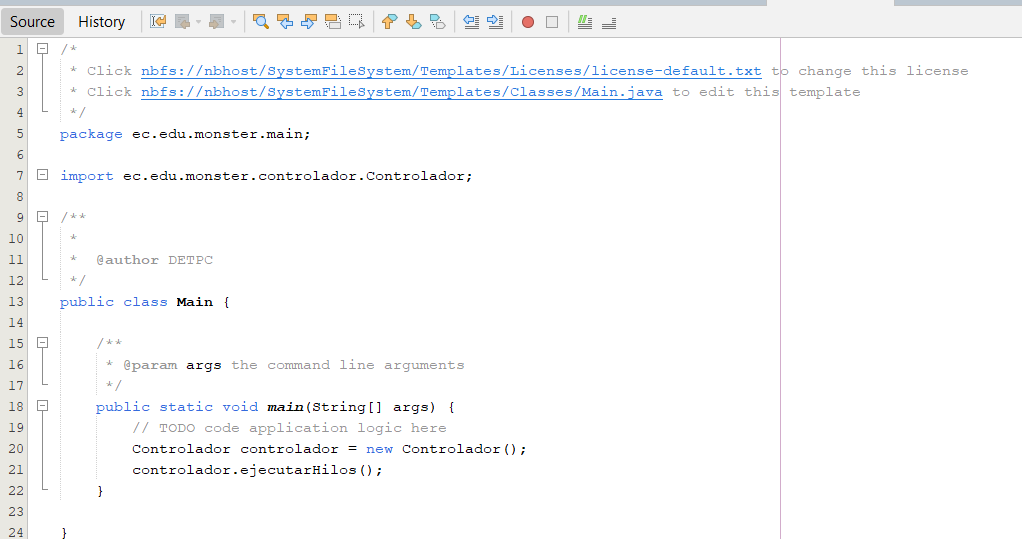
Por fines didácticos se adjuntará una captura del código necesario para esta clase, como también un cuadro de texto con el código para que pueda ser copiado con mayor facilidad:  


Figura 20. Implementación de la Clase Main

Código de la Clase 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** ec.edu.monster.main;

**import** **ec.edu.monster.controlador.Controlador**;

/\*\*

\*

\* @author DETPC

\*/

**public** **class** **Main** {

/\*\*

\* @param args the command line arguments

\*/

**public** **static** **void** **main**(String[] args) {

// TODO code application logic here

Controlador controlador = **new** Controlador();

controlador.ejecutarHilos();

}

}

Tabla 3. Codificación de la Clase Main

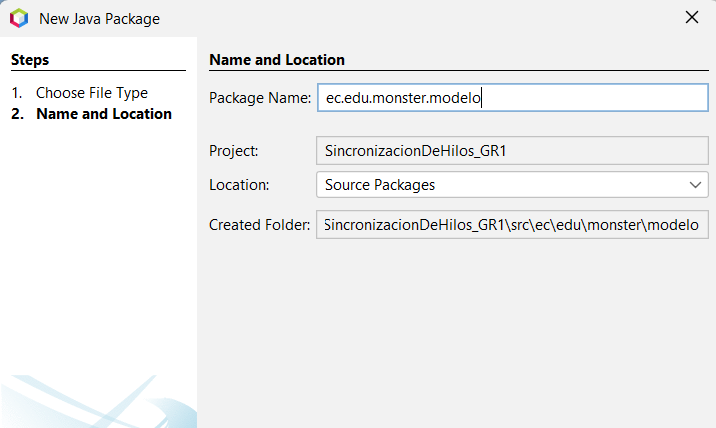
Creas el paquete Modelo:  


Figura 21. Creación de Paquete Modelo

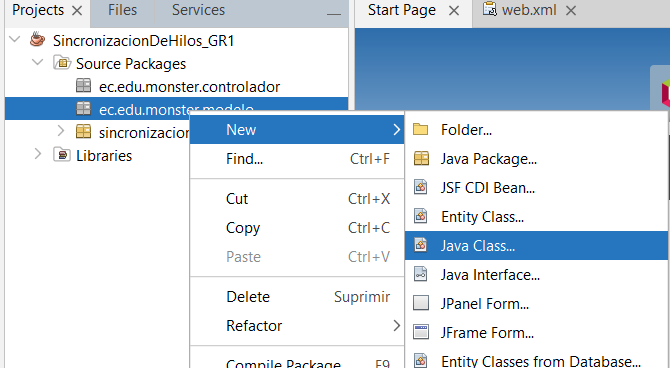
Dentro de este creas un JAVA CLASS, al que vas a llamar con el mismo nombre del paquete:  


Figura 22. Creación de Clase en el Paquete Modelo

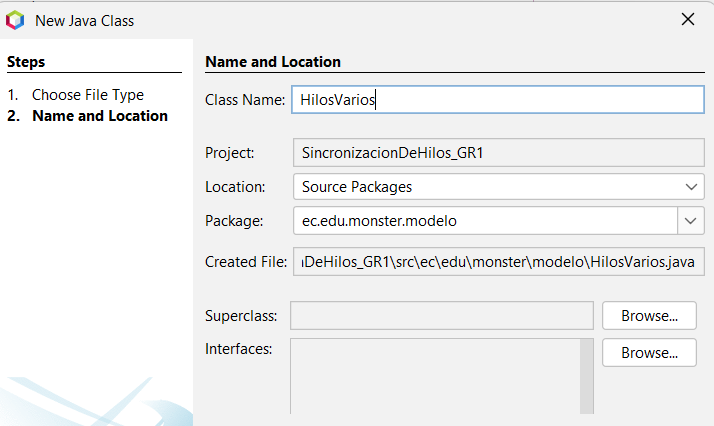


Figura 23. Nombrar la Clase como Hilos Varios

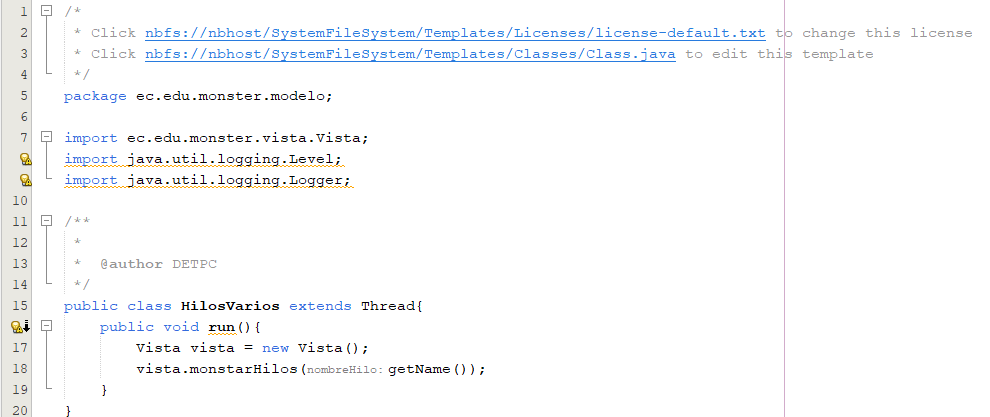
Por fines didácticos se adjuntará una captura del código necesario para esta clase, como también un cuadro de texto con el código para que pueda ser copiado con mayor facilidad:  


Figura 24. Implementación de la Clase Hilos Varios

Código de la Clase Hilos Varios:

/\*

\* 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** **ec.edu.monster.vista.Vista**;

**import** **java.util.logging.Level**;

**import** **java.util.logging.Logger**;

/\*\*

\*

\* @author DETPC

\*/

**public** **class** **HilosVarios** **extends** Thread{

**public** **void** **run**(){

Vista vista = **new** Vista();

vista.monstarHilos(getName());

}

}

Tabla 4. Codificación de la Clase Hilos Varios

Dentro del mismo paquete Model creas otra clase llamada HilosVarios2

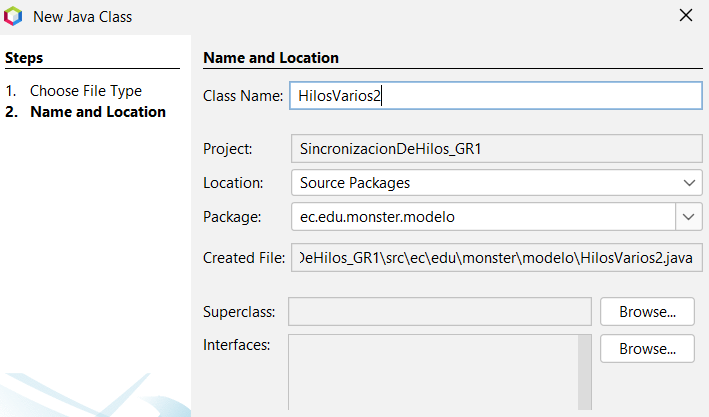


Figura 25. Nombrar la Clase como Hilos Varios2

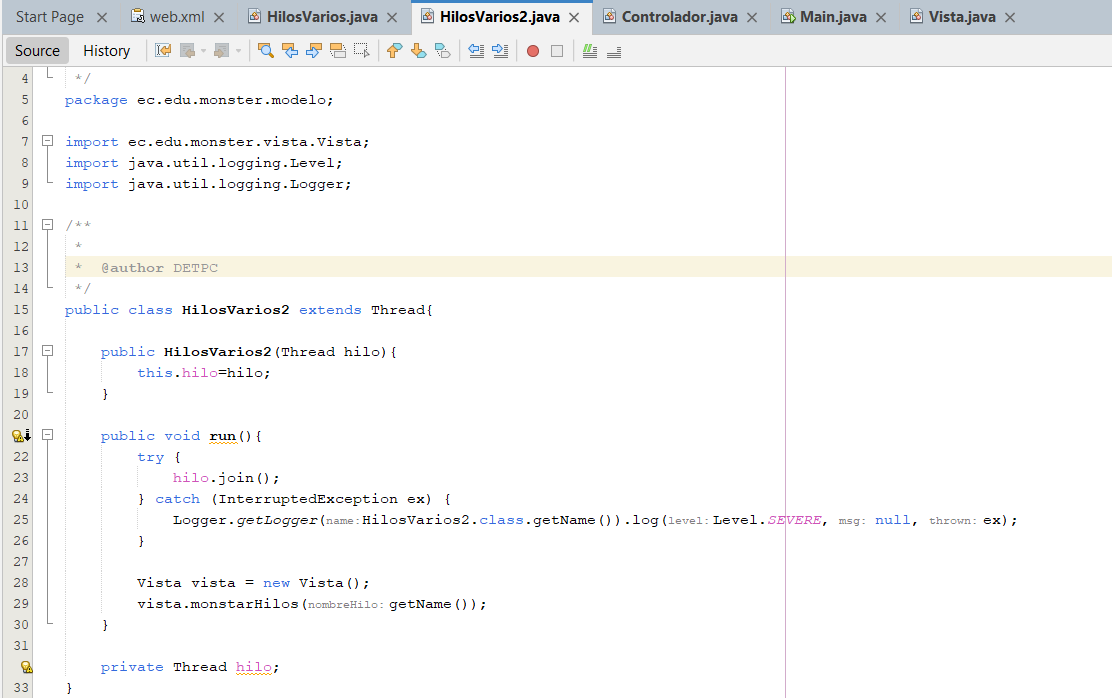
Por fines didácticos se adjuntará una captura del código necesario para esta clase, como también un cuadro de texto con el código para que pueda ser copiado con mayor facilidad:  


Figura 26. Implementación de la Clase Hilos Varios

Código de la Clase Hilos Varios2:

/\*

\* 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** **ec.edu.monster.vista.Vista**;

**import** **java.util.logging.Level**;

**import** **java.util.logging.Logger**;

/\*\*

\*

\* @author DETPC

\*/

**public** **class** **HilosVarios2** **extends** Thread{

**public** **HilosVarios2**(Thread hilo){

**this**.hilo=hilo;

}

**public** **void** **run**(){

**try** {

hilo.join();

} **catch** (InterruptedException ex) {

Logger.getLogger(HilosVarios2.class.getName()).log(Level.SEVERE, **null**, ex);

}

Vista vista = **new** Vista();

vista.monstarHilos(getName());

}

**private** Thread hilo;

}

Tabla 5.Codificación de la Clase Hilos Varios2

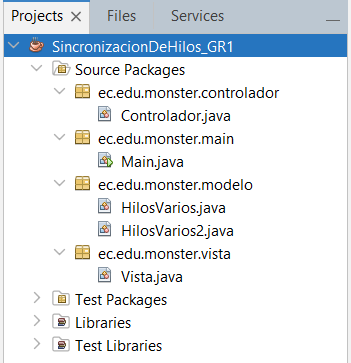
La estructura del Proyecto Debería ser la Siguiente   


Figura 27.Implementación de la Clase Hilos Varios

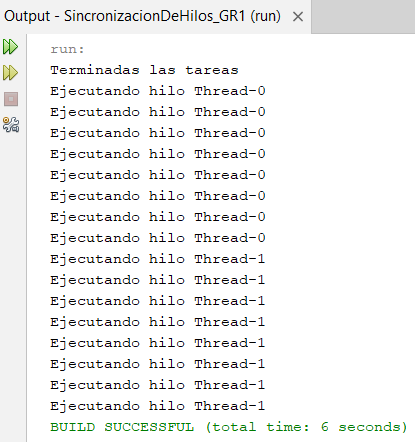
EJECUCIÓN DEL PROYECTO:  
Para ejecutar el proyecto debes ejecutar la clase Main, y la visualización esperada para este proyecto sería la siguiente:  


Figura 28. Implementación de la Clase Hilos Varios

# CONCLUSIONES

* La utilización de hilos en Java permite la ejecución simultánea de tareas y mejora la capacidad de respuesta de las aplicaciones. Al aplicar el patrón MVC, se logra una estructura clara y organizada que separa las responsabilidades y facilita el desarrollo y mantenimiento del código.
* La sincronización de hilos es fundamental para evitar problemas de concurrencia y garantizar la consistencia de los datos compartidos. Es importante aplicar técnicas de sincronización adecuadas, como el uso de bloques sincronizados o métodos sincronizados, para evitar condiciones de carrera y asegurar la integridad de los datos.
* El uso del entorno de desarrollo NetBeans facilita la implementación del patrón MVC, brindando herramientas y funcionalidades que agilizan el desarrollo y la depuración del código. NetBeans proporciona una interfaz intuitiva y opciones de generación de código que simplifican la estructura y la interacción entre el modelo, la vista y el controlador.

# RECOMENDACIONES

* Al utilizar hilos en Java, es importante realizar un análisis cuidadoso de los recursos compartidos y la sincronización requerida. Identificar correctamente los puntos críticos donde se deben aplicar bloqueos y sincronización garantizará un comportamiento correcto y eficiente del programa.
* Al implementar el patrón MVC, se recomienda seguir buenas prácticas de diseño y separación de responsabilidades. Mantener un modelo independiente de la vista y el controlador permitirá una mayor flexibilidad y reutilización del código, facilitando futuras modificaciones o mejoras en la aplicación.
* Es fundamental realizar pruebas exhaustivas y rigurosas en el programa que utiliza hilos y el patrón MVC. Las pruebas deben abarcar diferentes escenarios de concurrencia y validar la correcta interacción entre los componentes del MVC. Además, se deben verificar los resultados y la consistencia de los datos compartidos para garantizar la robustez y la calidad del software desarrollado

# BIBLIOGRAFÍA

*The Java® Virtual Machine Specification*. (s. f.). <https://docs.oracle.com/javase/specs/jvms/se16/html/index.html>

*Serializable (Java Platform SE 8 )*. (2023, 5 abril). <https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html>

*Java Platform, Standard Edition Documentation - Releases*. (2023, 21 marzo). Oracle Help Center. <https://docs.oracle.com/en/java/javase/index.html>

*Lesson: Concurrency (The JavaTM Tutorials > Essential Java Classes)*. (s. f.). <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

Dhamdhere, D. M. (2011). Operating Systems: A Concept-based Approach. Tata McGraw-Hill Education

*Lesson: Concurrency (The JavaTM Tutorials > Essential Java Classes)*. (s. f.-b). <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

*Synchronization (The JavaTM Tutorials >                    Essential Java Classes > Concurrency)*. (s. f.). <https://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html>