

**Tema**

DOMINIOS DE SEGURIDAD EN APLICACIONES JAVA ENTERPRISE EDITION

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

12/02/2022

DOMINIOS DE SEGURIDAD EN APLICACIONES JEE

[1. MARCO TEÓRICO 6](#_Toc120511155)

[1.1. OBJETIVOS 6](#_Toc120511156)

[1.2.1. OBJETIVO GENERAL 6](#_Toc120511157)

[1.2.2. OBJETIVOS ESPECÍFICOS 6](#_Toc120511158)

[1.2. JAVA. 6](#_Toc120511159)

[1.3. APACHE NETBEANS 6](#_Toc120511160)

[1.4. CONCURRENCIA 7](#_Toc120511161)

[1.5. HILOS 7](#_Toc120511162)

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

[1.7. HILOS PARALELOS Y CONCURRENTES 7](#_Toc120511163)

[1.8. TERMINOLOGÍA Y METODOS DE HILOS EN JAVA. 8](#_Toc120511164)

[2. PARTE PRÁCTICA 9](#_Toc120511165)

[2.1. Video 173 9](#_Toc120511166)

[2.1.1 CREACIÓN DEL PROYECTO. 9](#_Toc120511167)

[2.1.2 Creación de paquetes MVC. 10](#_Toc120511168)

[2.1.3. CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 11](#_Toc120511169)

[2.1.4. CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 14](#_Toc120511170)

[2.1.5. CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 17](#_Toc120511171)

[2.2. Video 174 20](#_Toc120511172)

[2.2.1 CREACIÓN DEL PROYECTO. 20](#_Toc120511173)

[2.2.2 Creación de paquetes MVC. 21](#_Toc120511174)

[2.2.3 CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 22](#_Toc120511175)

[2.2.4 CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 25](#_Toc120511176)

[2.2.5 CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 28](#_Toc120511177)

[2.2.6 CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN 34](#_Toc120511178)

[2.2.7 EJECUCIÓN DEL PROYECTO. 36](#_Toc120511179)

[2.3. Video 175 37](#_Toc120511180)

[2.3.1 CREACIÓN DEL PROYECTO. 37](#_Toc120511181)

[2.3.2 Creación de paquetes MVC. 38](#_Toc120511183)

[2.3.3 CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 39](#_Toc120511184)

[2.3.4 CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 42](#_Toc120511185)

[2.3.5 CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 45](#_Toc120511186)

[2.3.6 CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN 51](#_Toc120511187)

[2.3.7 EJECUCIÓN DEL PROYECTO. 53](#_Toc120511188)

[2.4. Video 176 54](#_Toc120511189)

[2.4.1 CREACIÓN DEL PROYECTO. 54](#_Toc120511190)

[2.4.2 Creación de paquetes MVC. 55](#_Toc120511192)

[2.4.3 CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 56](#_Toc120511193)

[2.4.4 CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 59](#_Toc120511194)

[2.4.5 CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 63](#_Toc120511195)

[2.4.6 CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN 71](#_Toc120511196)

[2.4.7 EJECUCIÓN DEL PROYECTO. 73](#_Toc120511197)

[2.5. Video 177 74](#_Toc120511198)

[2.5.1 CREACIÓN DEL PROYECTO. 74](#_Toc120511199)

[2.5.2 Creación de paquetes MVC. 75](#_Toc120511201)

[2.5.3 CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 76](#_Toc120511202)

[2.5.4 CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 79](#_Toc120511203)

[2.5.5 CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 83](#_Toc120511204)

[2.5.6 CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN 91](#_Toc120511205)

[2.5.7 EJECUCIÓN DEL PROYECTO. 93](#_Toc120511206)

[2.6. Video 178 94](#_Toc120511207)

[2.6.1 CREACIÓN DEL PROYECTO. 94](#_Toc120511208)

[2.6.2 Creación de paquetes MVC. 95](#_Toc120511210)

[2.6.3 CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 96](#_Toc120511211)

[2.6.4 CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 99](#_Toc120511212)

[2.6.5 CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 103](#_Toc120511213)

[2.6.6 CODIFICACIÓN DE LOS ARCHIVOS DEL PAQUETE MAIN 111](#_Toc120511214)

[2.6.7 EJECUCIÓN DEL PROYECTO. 113](#_Toc120511215)

[3. CONCLUSIONES 114](#_Toc120511216)

[4. RECOMENDACIONES 114](#_Toc120511217)

[5. BIBLIOGRAFIA 114](#_Toc120511218)

**INDICE DE IMÁGENES**

[Figura 1. Logotipo identificativo de Java 7](#_Toc120511219)

[Figura 2. Logotipo identificativo de Netbeans 10](#_Toc120511220)

[Figura 3. Selección del tipo de aplicación a desarrollar 10](#_Toc120511221)

[Figura 4. Nombre del proyecto 11](#_Toc120511222)

[Figura 5. Estructura del proyecto generado. 11](#_Toc120511223)

[Figura 6. Creación de paquete. 11](#_Toc120511224)

[Figura 7. Selección del nombre de los paquetes del video 173 12](#_Toc120511225)

[Figura 8. Creación de clase Java. 13](#_Toc120511226)

[Figura 9. Creación archivo Banco. 13](#_Toc120511227)

[Figura 10. Codificación archivo Banco 13](#_Toc120511228)

[Figura 11. Creación de clase Java. 15](#_Toc120511229)

[Figura 12. Creación archivo BancoVista. 15](#_Toc120511230)

[Figura 13. Codificación archivo BancoVista 16](#_Toc120511231)

[Figura 14. Creación de clase Java. 18](#_Toc120511232)

[Figura 15. Codificación archivo BancoControlador. 18](#_Toc120511233)

[Figura 16. Logotipo identificativo de Netbeans 21](#_Toc120511234)

[Figura 17. Selección del tipo de aplicación a desarrollar 21](#_Toc120511235)

[Figura 18. Nombre del proyecto 22](#_Toc120511236)

[Figura 19. Estructura del proyecto generado. 22](#_Toc120511237)

[Figura 20. Creación de paquete. 23](#_Toc120511238)

[Figura 21. Selección del nombre de los paquetes del video 174 23](#_Toc120511239)

[Figura 22. Creación de clase Java. 23](#_Toc120511240)

[Figura 23. Creación archivo Banco. 24](#_Toc120511241)

[Figura 24. Codificación archivo Banco 24](#_Toc120511242)

[Figura 25. Creación de clase Java. 26](#_Toc120511243)

[Figura 26. Creación archivo BancoVista. 26](#_Toc120511244)

[Figura 27. Codificación archivo BancoVista 27](#_Toc120511245)

[Figura 28. Creación de clase Java. 29](#_Toc120511246)

[Figura 29. Codificación archivo BancoControlador. 29](#_Toc120511247)

[Figura 30. Creación de clase Java. 32](#_Toc120511248)

[Figura 31. Codificación archivo TransferenciasControlador. 32](#_Toc120511249)

[Figura 32. Codificación archivo principal del video 174. 35](#_Toc120511250)

[Figura 33. Resultado del video 174. 37](#_Toc120511251)

[Figura 34. Logotipo identificativo de Netbeans 38](#_Toc120511252)

[Figura 35. Selección del tipo de aplicación a desarrollar 38](#_Toc120511253)

[Figura 36. Nombre del proyecto 39](#_Toc120511254)

[Figura 37. Estructura del proyecto generado. 39](#_Toc120511255)

[Figura 38. Creación de paquete. 40](#_Toc120511256)

[Figura 39. Selección del nombre de los paquetes del video 175 40](#_Toc120511257)

[Figura 40. Creación de clase Java. 40](#_Toc120511258)

[Figura 41. Creación archivo Banco. 41](#_Toc120511259)

[Figura 42. Codificación archivo Banco 41](#_Toc120511260)

[Figura 43. Creación de clase Java. 43](#_Toc120511261)

[Figura 44. Creación archivo BancoVista. 43](#_Toc120511262)

[Figura 45. Codificación archivo BancoVista 44](#_Toc120511263)

[Figura 46. Creación de clase Java. 46](#_Toc120511264)

[Figura 47. Codificación archivo BancoControlador. 46](#_Toc120511265)

[Figura 48. Creación de clase Java. 49](#_Toc120511266)

[Figura 49. Codificación archivo TransferenciasControlador. 49](#_Toc120511267)

[Figura 50. Codificación archivo principal del video 175. 52](#_Toc120511268)

[Figura 51. Resultado del video 175. 54](#_Toc120511269)

[Figura 52. Logotipo identificativo de Netbeans 55](#_Toc120511270)

[Figura 53. Selección del tipo de aplicación a desarrollar 55](#_Toc120511271)

[Figura 54. Nombre del proyecto 56](#_Toc120511272)

[Figura 55. Estructura del proyecto generado. 56](#_Toc120511273)

[Figura 56. Creación de paquete. 57](#_Toc120511274)

[Figura 57. Selección del nombre de los paquetes del video 176 57](#_Toc120511275)

[Figura 57. Creación de clase Java. 57](#_Toc120511276)

[Figura 58. Creación archivo Banco. 58](#_Toc120511277)

[Figura 59. Codificación archivo Banco 58](#_Toc120511278)

[Figura 60. Creación de clase Java. 60](#_Toc120511279)

[Figura 61. Creación archivo BancoVista. 60](#_Toc120511280)

[Figura 62. Codificación archivo BancoVista 61](#_Toc120511281)

[Figura 63. Continuación codificación archivo BancoVista 61](#_Toc120511282)

[Figura 64. Creación de clase Java. 64](#_Toc120511283)

[Figura 65. Creación archivo BancoControlador. 64](#_Toc120511284)

[Figura 66. Codificación archivo BancoControlador. 65](#_Toc120511285)

[Figura 67. Creación de clase Java. 68](#_Toc120511286)

[Figura 68. Creación de archivo TransferenciasControlador. 68](#_Toc120511287)

[Figura 69. Codificación archivo TransferenciasControlador. 69](#_Toc120511288)

[Figura 70. Codificación archivo principal del video 176. 72](#_Toc120511289)

[Figura 71. Resultado del video 176. 74](#_Toc120511290)

[Figura 72. Logotipo identificativo de Netbeans 75](#_Toc120511291)

[Figura 73. Selección del tipo de aplicación a desarrollar 75](#_Toc120511292)

[Figura 74. Nombre del proyecto 76](#_Toc120511293)

[Figura 75. Estructura del proyecto generado. 76](#_Toc120511294)

[Figura 76. Creación de paquete. 77](#_Toc120511295)

[Figura 77. Selección del nombre de los paquetes del video 176 77](#_Toc120511296)

[Figura 78. Creación de clase Java. 77](#_Toc120511297)

[Figura 79. Creación archivo Banco. 78](#_Toc120511298)

[Figura 80. Codificación archivo Banco 78](#_Toc120511299)

[Figura 81. Creación de clase Java. 80](#_Toc120511300)

[Figura 82. Creación archivo BancoVista. 80](#_Toc120511301)

[Figura 83. Codificación archivo BancoVista 81](#_Toc120511302)

[Figura 84. Creación de clase Java. 84](#_Toc120511303)

[Figura 85. Creación archivo BancoControlador. 84](#_Toc120511304)

[Figura 86. Codificación archivo BancoControlador. 85](#_Toc120511305)

[Figura 87. Creación de clase Java. 88](#_Toc120511306)

[Figura 88. Creación de archivo TransferenciasControlador. 88](#_Toc120511307)

[Figura 89. Codificación archivo TransferenciasControlador. 89](#_Toc120511308)

[Figura 90. Codificación archivo principal del video 177. 92](#_Toc120511309)

[Figura 91. Resultado del video 177. 94](#_Toc120511310)

[Figura 92. Logotipo identificativo de Netbeans 95](#_Toc120511311)

[Figura 93. Selección del tipo de aplicación a desarrollar 95](#_Toc120511312)

[Figura 94. Nombre del proyecto 96](#_Toc120511313)

[Figura 95. Estructura del proyecto generado. 96](#_Toc120511314)

[Figura 96. Creación de paquete. 97](#_Toc120511315)

[Figura 97. Selección del nombre de los paquetes del video 176 97](#_Toc120511316)

[Figura 98. Creación de clase Java. 97](#_Toc120511317)

[Figura 99. Creación archivo Banco. 98](#_Toc120511318)

[Figura 100. Codificación archivo Banco 98](#_Toc120511319)

[Figura 101. Creación de clase Java. 100](#_Toc120511320)

[Figura 102. Creación archivo BancoVista. 100](#_Toc120511321)

[Figura 103. Codificación archivo BancoVista 101](#_Toc120511322)

[Figura 104. Creación de clase Java. 104](#_Toc120511323)

[Figura 105. Creación archivo BancoControlador. 104](#_Toc120511324)

[Figura 106. Codificación archivo BancoControlador. 105](#_Toc120511325)

[Figura 107. Creación de clase Java. 108](#_Toc120511326)

[Figura 108. Creación de archivo TransferenciasControlador. 108](#_Toc120511327)

[Figura 109. Codificación archivo TransferenciasControlador. 109](#_Toc120511328)

[Figura 110. Codificación archivo principal del video 178. 112](#_Toc120511329)

[Figura 111. Resultado del video 178. 114](#_Toc120511330)

**ÍNDICE DE TABLAS**

[Tabla 1. Terminología utilizada en Hilos Java 8](#_Toc120511331)

[Tabla 2. Codificación del archivo Banco 14](#_Toc120511332)

[Tabla 3. Codificación del archivo BancoVista 17](#_Toc120511333)

[Tabla 4. Codificación del archivo BancoControlador 19](#_Toc120511334)

[Tabla 5. Codificación del archivo Banco 25](#_Toc120511335)

[Tabla 6. Codificación del archivo BancoVista 28](#_Toc120511336)

[Tabla 7. Codificación del archivo BancoControlador 30](#_Toc120511337)

[Tabla 8. Codificación del archivo TransferenciasControlador 33](#_Toc120511338)

[Tabla 9. Codificación del archivo principal del proyecto 36](#_Toc120511339)

[Tabla 10. Codificación del archivo Banco 42](#_Toc120511340)

[Tabla 11. Codificación del archivo BancoVista 45](#_Toc120511341)

[Tabla 12. Codificación del archivo BancoControlador 47](#_Toc120511342)

[Tabla 13. Codificación del archivo TransferenciasControlador 50](#_Toc120511343)

[Tabla 14. Codificación del archivo principal del proyecto 53](#_Toc120511344)

[Tabla 15. Codificación del archivo Banco 59](#_Toc120511345)

[Tabla 16. Codificación del archivo BancoVista 62](#_Toc120511346)

[Tabla 17. Codificación del archivo BancoControlador 66](#_Toc120511347)

[Tabla 18. Codificación del archivo TransferenciasControlador 70](#_Toc120511348)

[Tabla 19. Codificación del archivo principal del proyecto 73](#_Toc120511349)

[Tabla 20. Codificación del archivo Banco 79](#_Toc120511350)

[Tabla 21. Codificación del archivo BancoVista 82](#_Toc120511351)

[Tabla 22. Codificación del archivo BancoControlador 86](#_Toc120511352)

[Tabla 23. Codificación del archivo TransferenciasControlador 90](#_Toc120511353)

[Tabla 24. Codificación del archivo principal del proyecto 93](#_Toc120511354)

[Tabla 25. Codificación del archivo Banco 99](#_Toc120511355)

[Tabla 26. Codificación del archivo BancoVista 102](#_Toc120511356)

[Tabla 27. Codificación del archivo BancoControlador 106](#_Toc120511357)

[Tabla 28. Codificación del archivo TransferenciasControlador 110](#_Toc120511358)

[Tabla 29. Codificación del archivo principal del proyecto 113](#_Toc120511359)

# MARCO TEÓRICO

## OBJETIVOS

### OBJETIVO GENERAL

Comprender la funcionalidad de los hilos en Java para ejecutar múltiples 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.

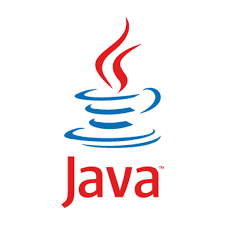


Figura 1. Logotipo identificativo de Java

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.

## SINCRONIZACIÓN 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 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.

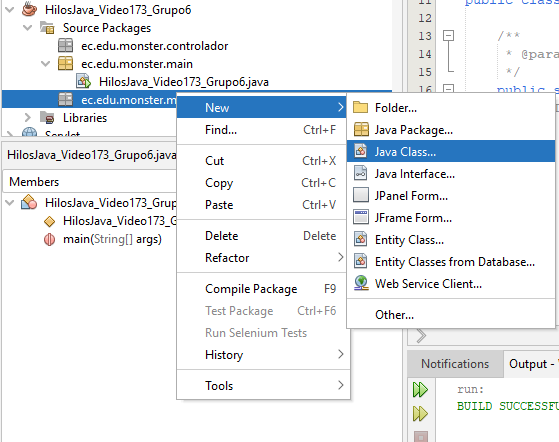


Figura 8. Creación de clase Java.

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 2. 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 3. 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 4. 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

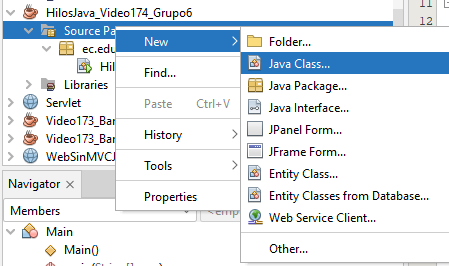


Figura 20. Creación de paquete.

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 5. 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 6. 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 7. 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 8. 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 9. 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 10. 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.

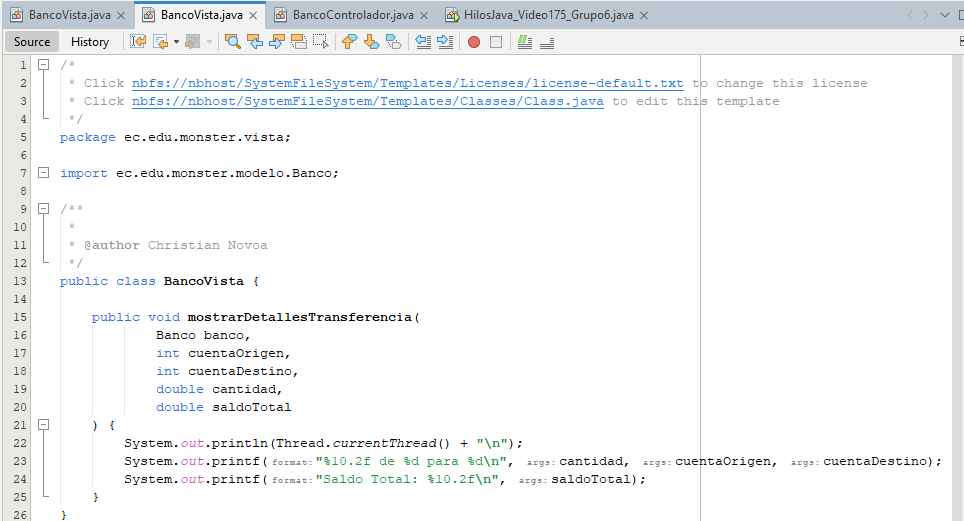
****

Figura 45. Codificación archivo BancoVista

Tabla 11. 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.

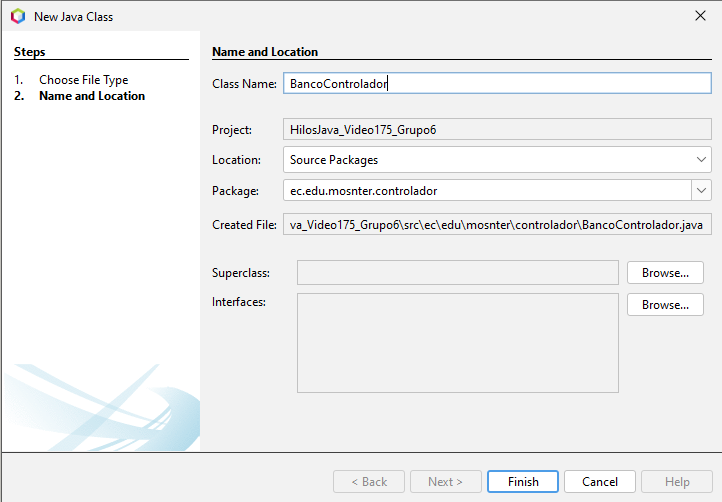


Figura 47. Codificación archivo BancoControlador.

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 12. 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 13. 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.

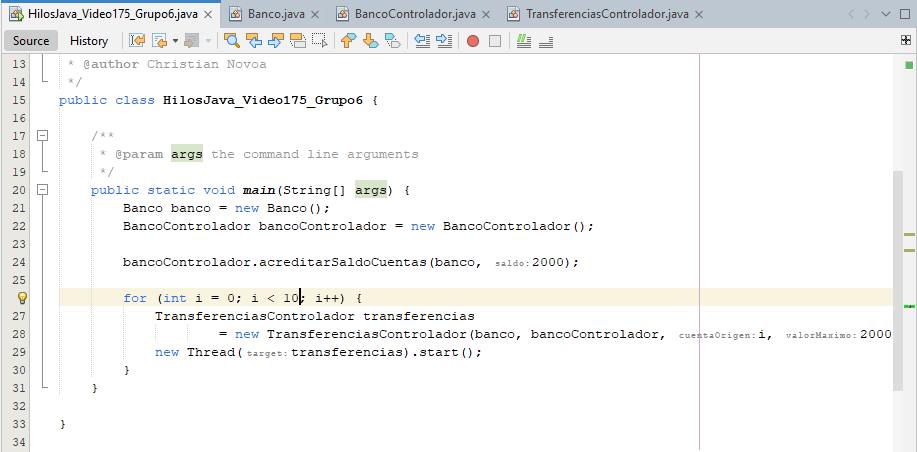


Figura 50. Codificación archivo principal del video 175.

Tabla 14. 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.

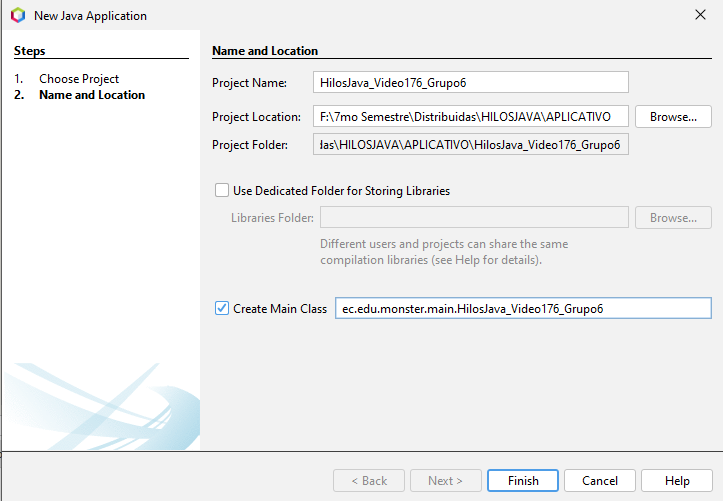
****

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 15. 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 16. 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.

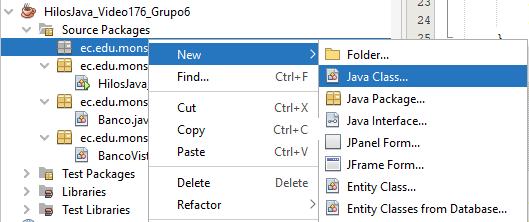


Figura 64. Creación de clase Java.

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 17. 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.

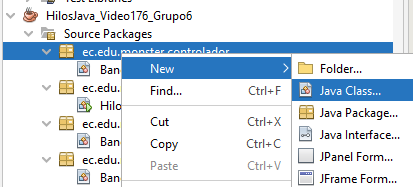


Figura 67. Creación de clase Java.

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 18. 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 19. 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.

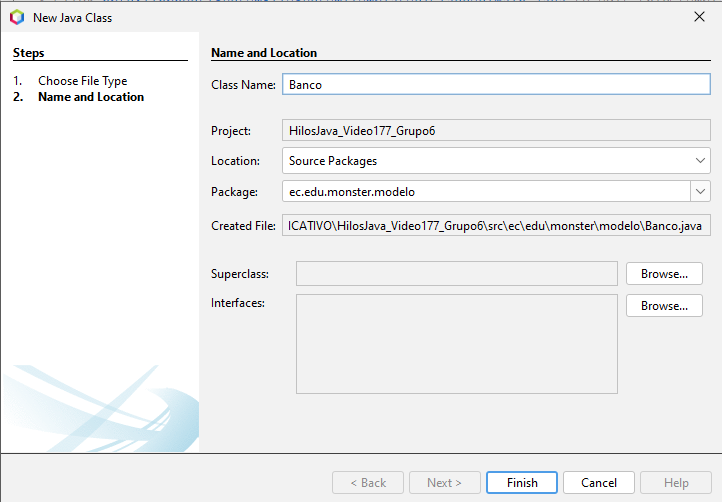


Figura 79. Creación archivo Banco.

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 20. 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 21. 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 22. 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 23. 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 24. 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.

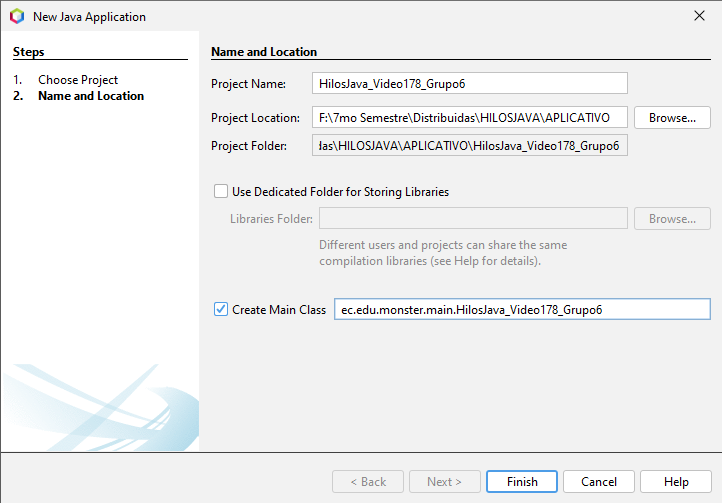
****

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 25. 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.

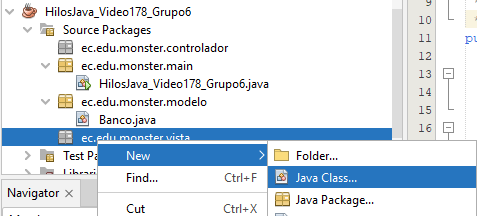


Figura 101. Creación de clase Java.

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 26. 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 27. 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 28. 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 29. 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.
* Java nos permite generar proyectos con la capacidad de utilizar hilos de manera fácil y rápida ya que incluye métodos y bibliotecas propias.
* 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.
* Cuando se utilizan hilos las capacidades de nuestros proyectos aumentan ya que pueden hacer más tareas a la vez.
* Se pueden realizar una especie de semáforos a los hilos cuando se utiliza programación concurrente para que de esta manera se pueda controlar el bloqueo de los hilos cuando uno necesita completar su tarea antes de que otro entre a la misma.

# 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]. |