Tema

MANUAL

TALLER HILOS

Autores

JOHAO ALEJANDRO MORALES PISCO

MAYCOL ESTALIN TITUAÑA TUPIZA

ALEX DARIO VELASTEGUI SOLIS

Tutor

Ing. Eduardo Mauricio Campana Ortega

MIS.MDU.CCNA.CCIA.

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

Fecha

08/07/2023

TABLA DE CONTENIDOS

INTRODUCCIÓN 6

OBJETIVO 6

MARCO TEÓRICO 6

JAVA 6

PROGRAMAS MONOTAREA 7

PROGRAMAS MULTITAREA 7

PROGRAMACIÓN CONCURRENTE 7

PATRÓN MODELO VISTA CONTROLADOR 8

MODELO 9

VISTA 9

AWT 9

SWING 10

CONTROLADOR 11

HILOS 11

CICLO DE VIDA DE UN HILO 12

OTROS MÉTODOS UTILIZADOS EN THREADS 13

INTERFAZ RUNNABLE 13

USO DE EXCEPCIONES 14

INTERFAZ ACTIONLISTENER 15

GRAPHICS 15

GRAPHICS2D 15

COLOR 15

ELLIPSE2D 16

RECTANGLE2D 16

RANDOM 16
JBUTTON, JFRAME, JPANEL 16
DESCRIPCIÓN DEL TALLER DE HILOS 16
INSTALACIÓN DE LAS HERRAMIENTAS 17
INSTALACIÓN DE NETBEANS 17
DESARROLLO DE LA APLICACIÓN 26
CREACIÓN DEL PROYECTO 26
CÓDIGO 36
EJECUCIÓN DEL PROYECTO 42
Pelotas 42
CONCLUSIONES 44
RECOMENDACIONES 44
BIBLIOGRAFÍA 45

ÍNDICE DE FIGURAS

Figura 1. Logo de Java *7*

Figura 2. Esquema del patrón MVC *8*

Figura 3. Esquema de componentes de la librería AWT *10*

Figura 4. Esquema de componentes de la librería Swing *11*

Figura 5. Esquema del ciclo de vida de un hilo *13*

Figura 6. Ejemplo de uso de la interfaz Runnable *14*

Figura 7. Ejemplo de manejo de excepciones en Java *15*

Figura 8. Página descarga NetBeans *17*

Figura 9. Página descarga imagen Java *18*

Figura 10. Guardar archivo en computador *18*

Figura 11. Ventana instalador java *19*

Figura 12. Selección características JDK, JRE *19*

Figura 13. Selección carpeta destino Java *20*

Figura 14. Finalizar la instalación Java *20*

Figura 15. Instalación NetBeans *21*

Figura 16. Guardar el archivo *22*

Figura 17. Configuración del instalador NetBeans *22*

Figura 18. Características a instalar NetBeans *23*

Figura 19. Selección de paquetes y servidores a instalar *24*

Figura 20. Aceptación de términos *24*

Figura 21. Selección de carpeta de instalación *25*

Figura 22. Instalación de paquetes seleccionados *25*

Figura 23. Finalización de la instalación *26*

Figura 24. Crear un nuevo proyecto *27*

Figura 25. Selección del tipo de proyecto a crear *28*

Figura 26. Asignación de nombre al proyecto *29*

Figura 27. Archivo vacío "UsoThreads.java" *29*

Figura 28. Importaciones *29*

Figura 29. Clase principal main *30*

Figura 30. Clase Pelota 1 ..... *31*

Figura 31. Clase Pelota 2 ..... *31*

Figura 32. Clase LaminaPelota *32*

Figura 33. clase MarcoRebote *33*

Figura 34. Clase PelotaHilos *34*

Figura 35. ponerBoton *35*

Figura 36. Metodo comienza_el_juego() *35*

Figura 37. Forma Inicial *42*

Figura 38. Ejecución con una pelota *42*

Figura 39. Ejecución con varias pelotas *43*

INTRODUCCIÓN

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

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

OBJETIVO

Desarrollar e implementar una aplicación utilizando hilos en Java con el IDE de desarrollo Apache NetBeans, que permita crear un programa animado de pelotas en movimiento en la pantalla. La aplicación utilizará hilos para controlar el movimiento de las pelotas de forma simultánea, brindando una experiencia visualmente atractiva y aprovechando los beneficios de la programación concurrente.

MARCO TEÓRICO

JAVA

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

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

- Simple y potente: Java ofrece un lenguaje potente, derivada de C y C++, pero sin las características más complicadas (y menos usadas) de estos.
- Orientado a objetos: el enfoque de Java es Orientado a Objetos (OO), es decir, posee los cuatro principios de la programación OO, siendo estos: herencia, encapsulamiento, polimorfismo y abstracción.
- Distribuido: Java proporciona de manera estándar bibliotecas y herramientas para que las aplicaciones desarrolladas sean distribuidas.

- Portable: los programas desarrollados en Java son independientes de la plataforma.
- Multihilo: Java puede llevar a cabo varias tareas de forma simultánea dentro del mismo programa, permitiendo mayor rendimiento y velocidad en la ejecución.

Figura 1. Logo de Java

PROGRAMAS MONOTAREA

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

PROGRAMAS MULTITAREA

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

PROGRAMACIÓN CONCURRENTE

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

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

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

PATRÓN MODELO VISTA CONTROLADOR

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

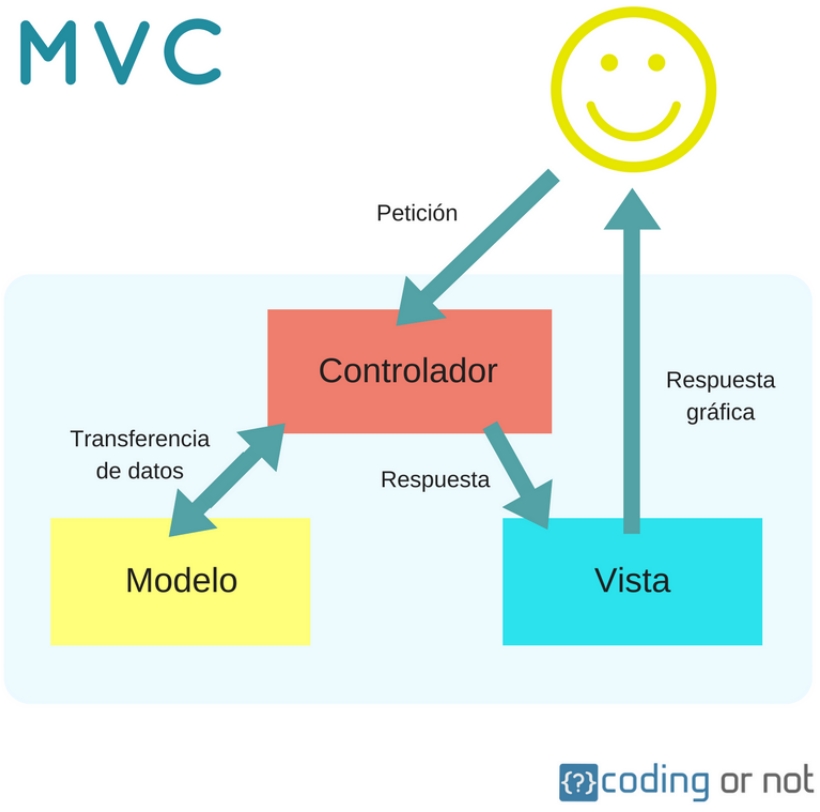

Figura 2. Esquema del patrón MVC

MODELO

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

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

VISTA

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

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

AWT

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

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

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

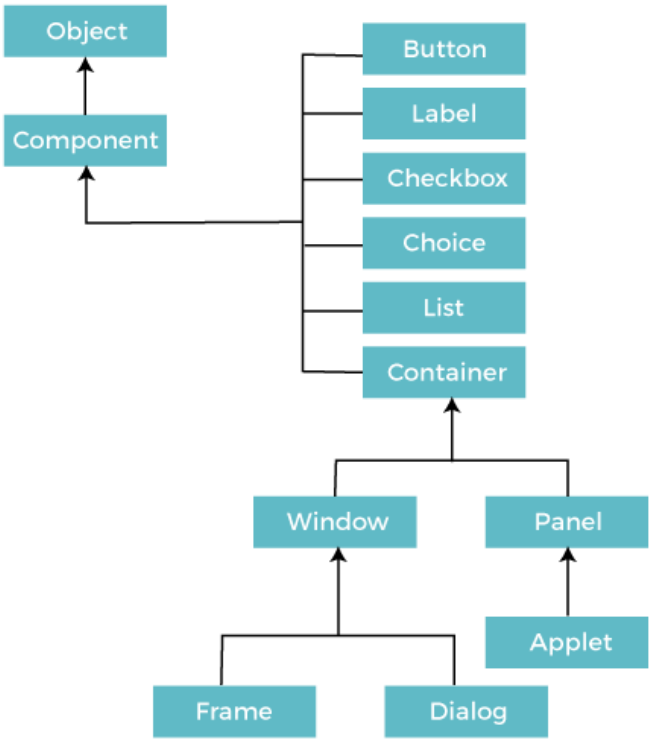

Figura 3. Esquema de componentes de la librería AWT

SWING

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

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

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

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

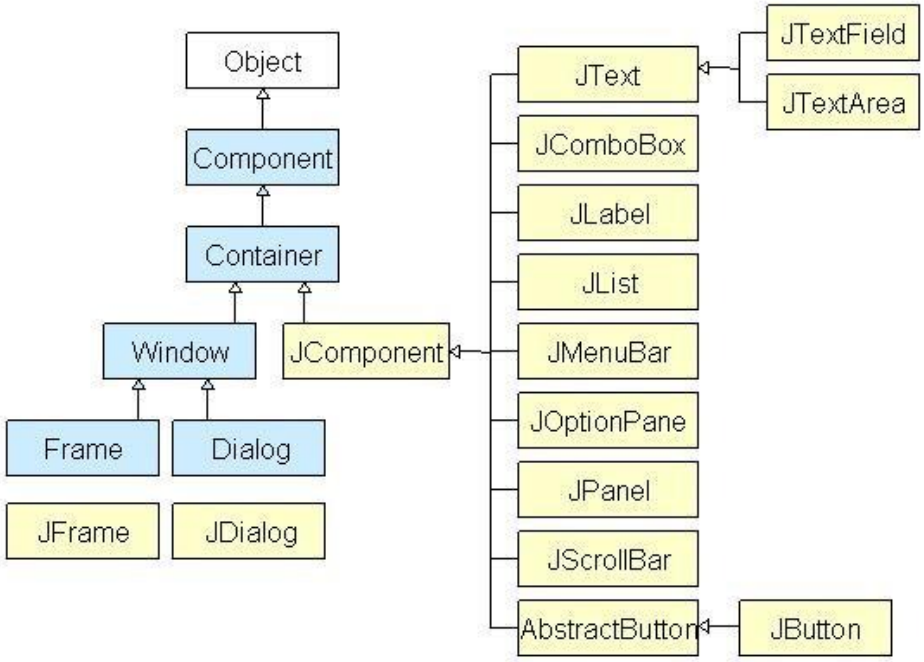

Figura 4. Esquema de componentes de la librería Swing

CONTROLADOR

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

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

HILOS

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

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

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

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

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

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

CICLO DE VIDA DE UN HILO

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

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

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

- Cuando el hilo se encuentre dormido por haber llamado al método sleep, pasará al estado "no ejecutable" durante la cantidad de milisegundos indicada en el método [6].

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

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

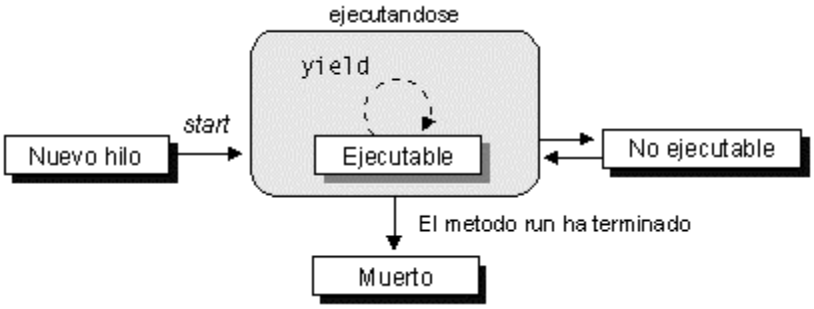

Figura 5. Esquema del ciclo de vida de un hilo

OTROS MÉTODOS UTILIZADOS EN THREADS

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

INTERFAZ RUNNABLE

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

```
public class ThreadEjemplo implements Runnable {
    public void run() {
        for (int i = 0; i < 5 ; i++)
            System.out.println(i + " " +
                               Thread.currentThread().getName());
        System.out.println("Termina thread " +
                               Thread.currentThread().getName());
    }
    public static void main (String [] args) {
        new Thread (new ThreadEjemplo(), "Pepe").start();
        new Thread (new ThreadEjemplo(), "Juan").start();
        System.out.println("Termina thread main");
    }
}
```

Figura 6. Ejemplo de uso de la interfaz Runnable

USO DE EXCEPCIONES

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

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

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

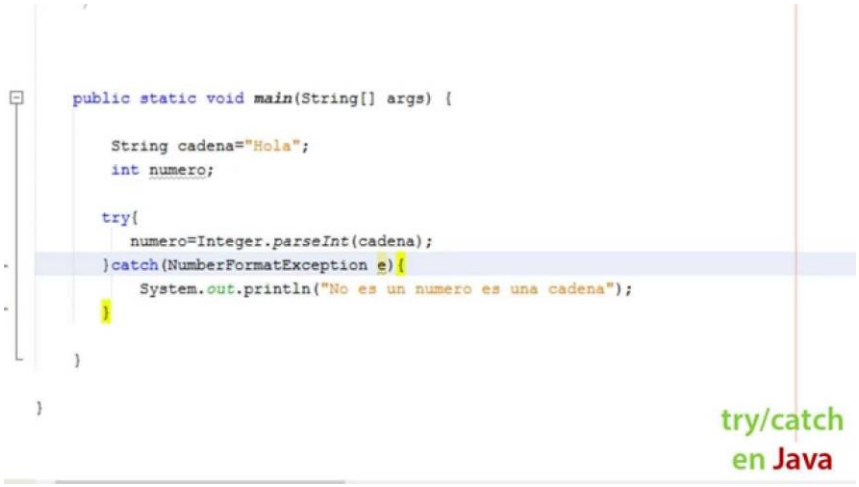

Figura 7. Ejemplo de manejo de excepciones en Java

INTERFAZ ACTIONLISTENER

ActionListener es una interfaz de Java que se encuentra dentro del paquete java.awt.event. ActionLister pertenece al grupo de Listeners (escuchadores); posee un único método llamado: actionPerformed(ActionEvent e), en el que se especifican las acciones a realizar en caso de que un evento ocurra [9].

GRAPHICS

Esta librería proporciona clases y métodos para realizar operaciones de dibujo y manipulación gráfica básica en Java. La clase Graphics es una clase abstracta que actúa como un lienzo en el que se pueden dibujar formas, texto y otros elementos gráficos [10].

GRAPHICS2D

Esta librería extiende la funcionalidad de la clase Graphics y proporciona métodos adicionales para realizar operaciones de dibujo más avanzadas y precisas. La clase Graphics2D es una subclase de Graphics y permite trabajar con gráficos en dos dimensiones (2D) [11].

COLOR

Esta librería proporciona una amplia gama de constantes predefinidas para representar colores en Java. La clase Color permite crear objetos que representan colores mediante combinaciones de valores RGB (rojo, verde, azul) o mediante nombres predefinidos de colores [12].

ELLIPSE2D

Esta librería proporciona clases y métodos para trabajar con formas geométricas en Java. La clase Ellipse2D representa una elipse o círculo en un espacio bidimensional y permite realizar operaciones como la obtención de sus coordenadas, tamaño y área [13].

RECTANGLE2D

Esta librería proporciona clases y métodos para trabajar con formas geométricas en Java. La clase Rectangle2D representa un rectángulo en un espacio bidimensional y permite realizar operaciones como la obtención de sus coordenadas, tamaño y área [14].

RANDOM

Esta librería proporciona una clase Random que permite generar números aleatorios en Java. Puede ser utilizada para generar valores aleatorios en un rango específico o para simular eventos o comportamientos aleatorios en una aplicación [15].

JBUTTON, JFRAME, JPANEL

Estas librerías forman parte del conjunto de librerías de Java Swing, que proporcionan componentes y clases para construir interfaces gráficas de usuario (GUI) en Java. La clase JFrame representa una ventana de la aplicación, la clase JPanel proporciona un contenedor para organizar y mostrar componentes gráficos, y la clase JButton es un botón interactivo en la interfaz [16].

DESCRIPCIÓN DEL TALLER DE HILOS

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

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

INSTALACIÓN DE LAS HERRAMIENTAS

INSTALACIÓN DE NETBEANS

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

   - https://www.oracle.com/technetwork/es/java/javase/downloads/index.html

Figura 8. Página descarga NetBeans

2. En esta página se da clic en Download con la imagen de java.

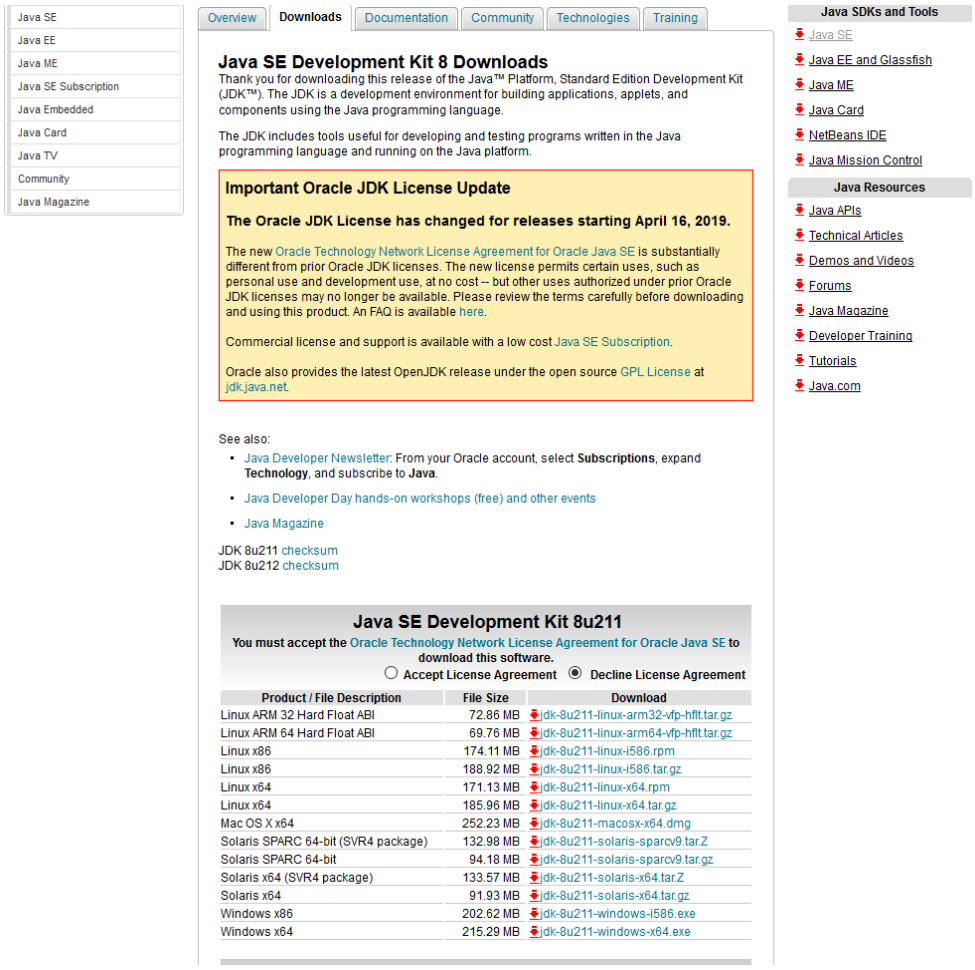

Figura 9. Página descarga imagen Java

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

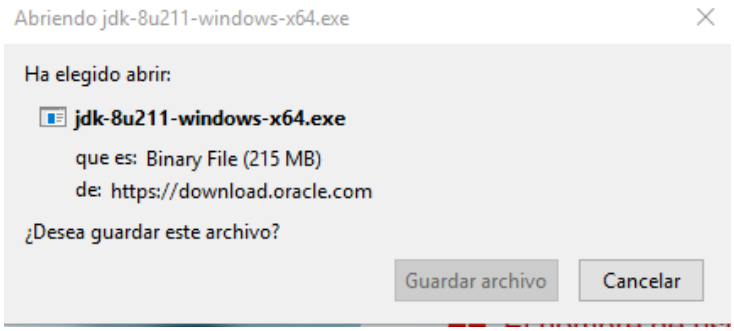

Figura 10. Guardar archivo en computador

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

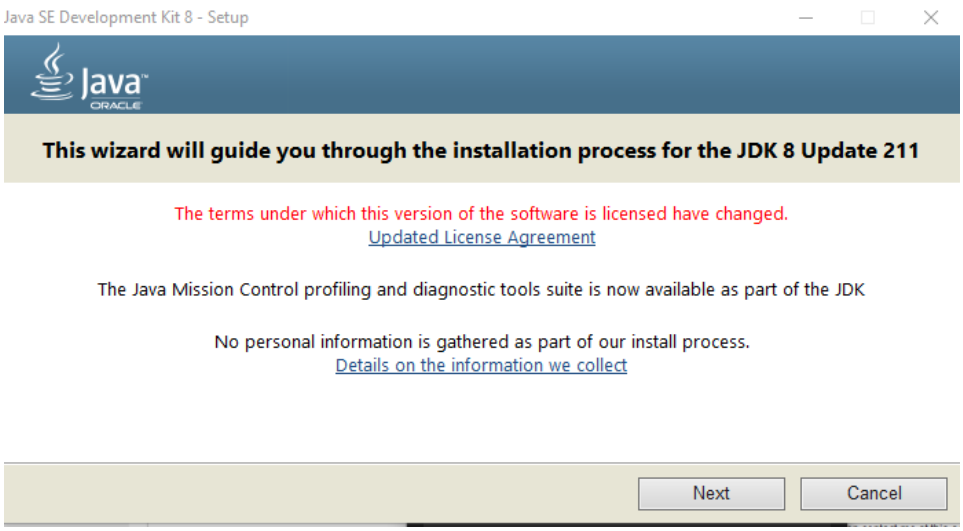

Figura 11. Ventana instalador java

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

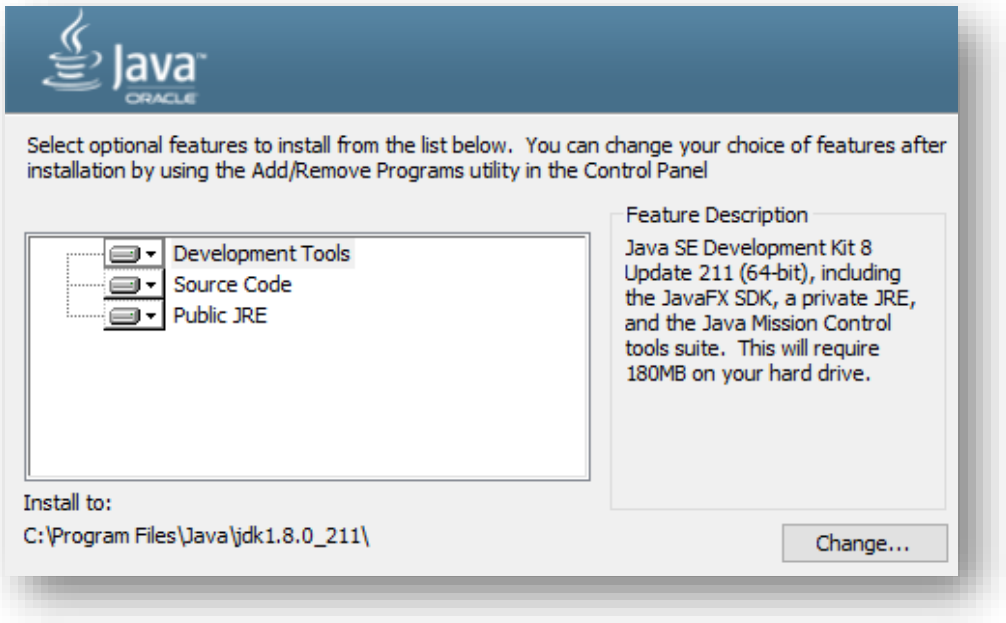

Figura 12. Selección características JDK, JRE

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

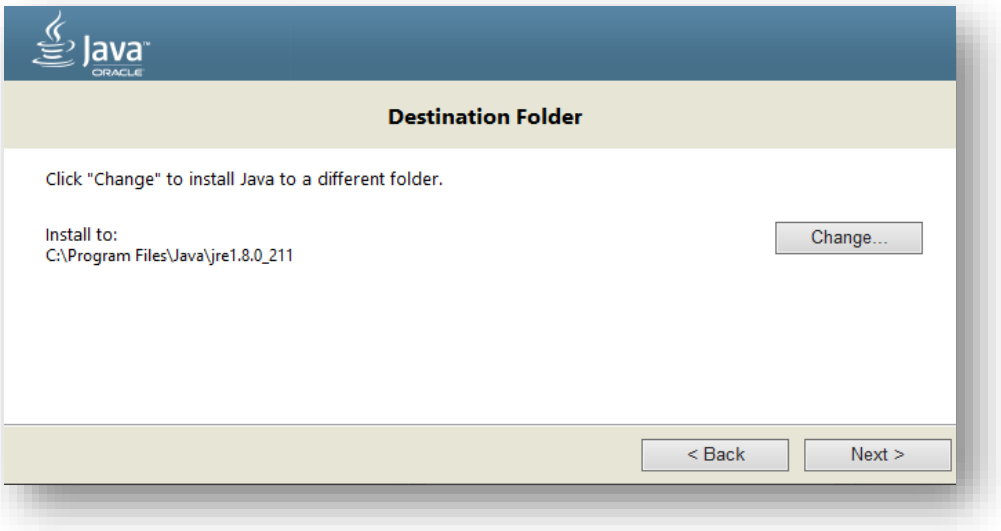

Figura 13. Selección carpeta destino Java

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

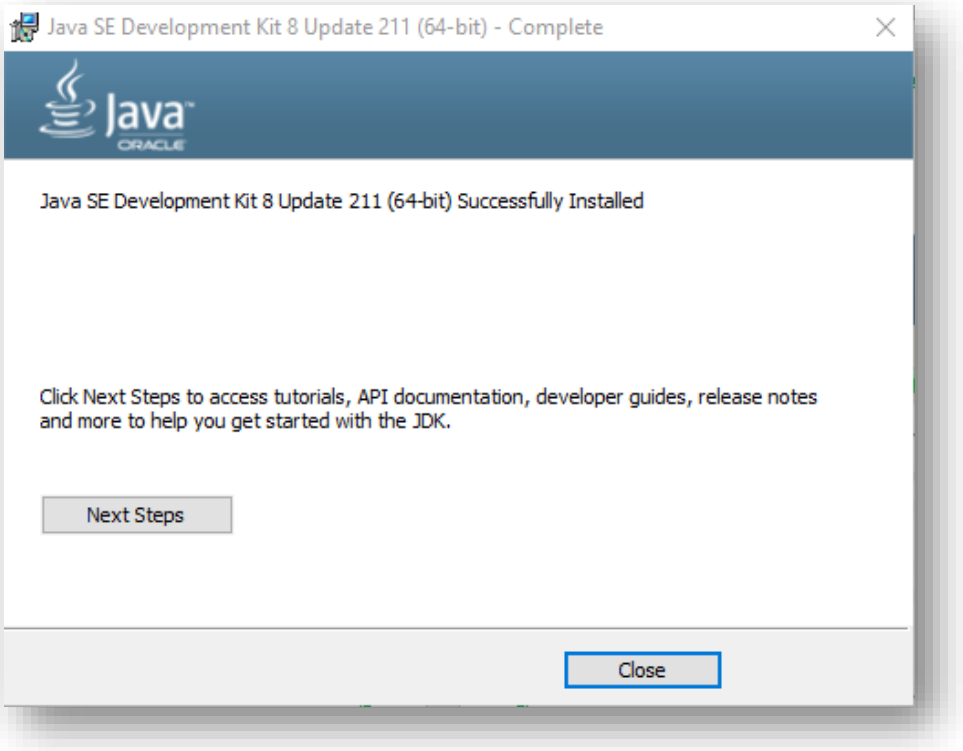

Figura 14. Finalizar la instalación Java

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

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

- https://netbeans.org/downloads/8.2/

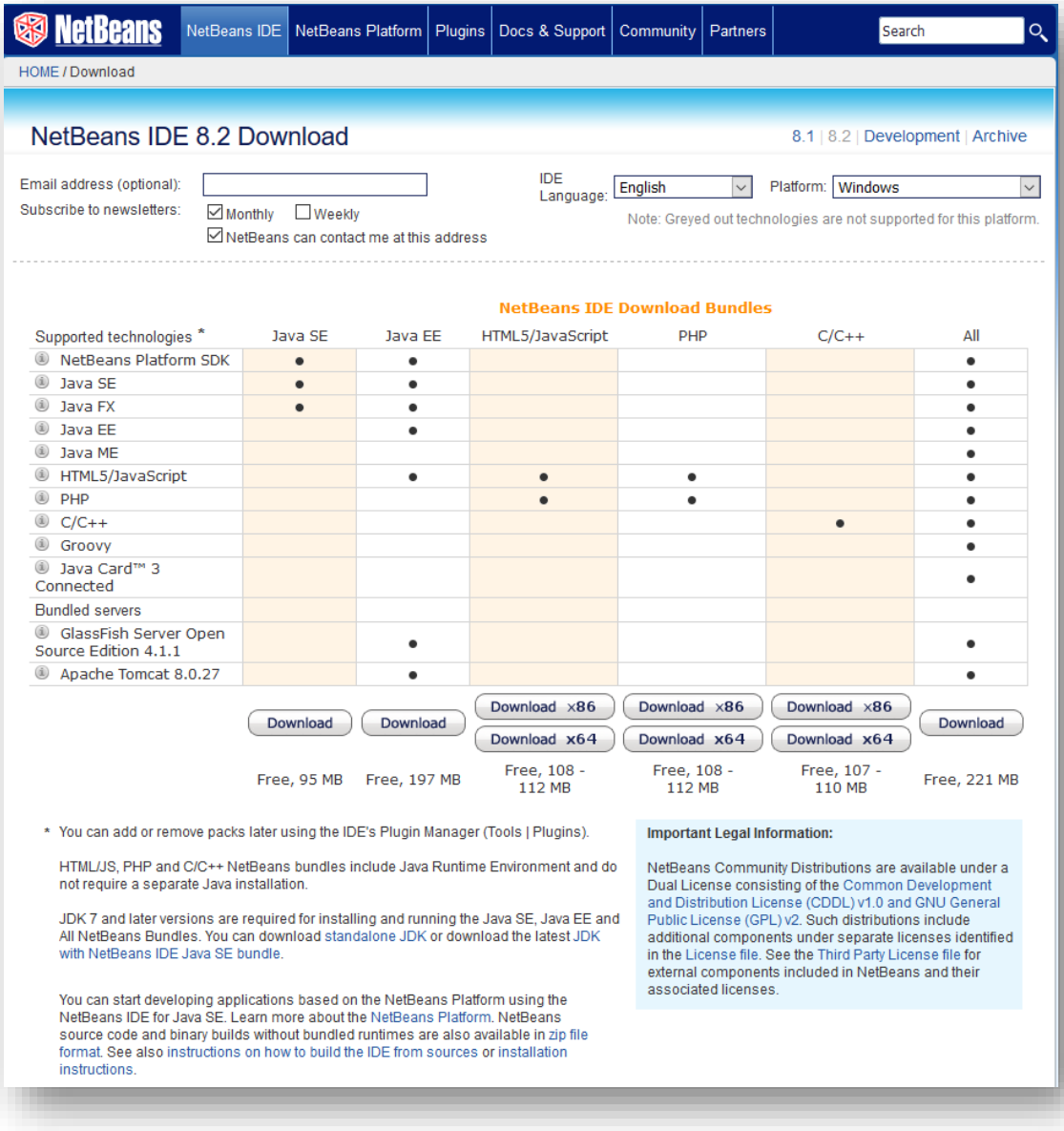

Figura 15. Instalación NetBeans

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

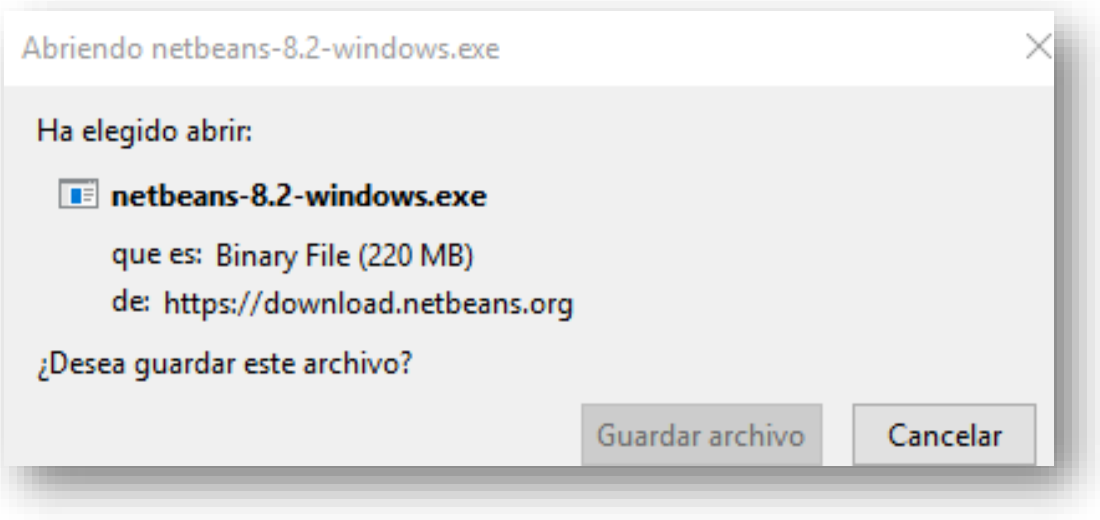

Figura 16. Guardar el archivo

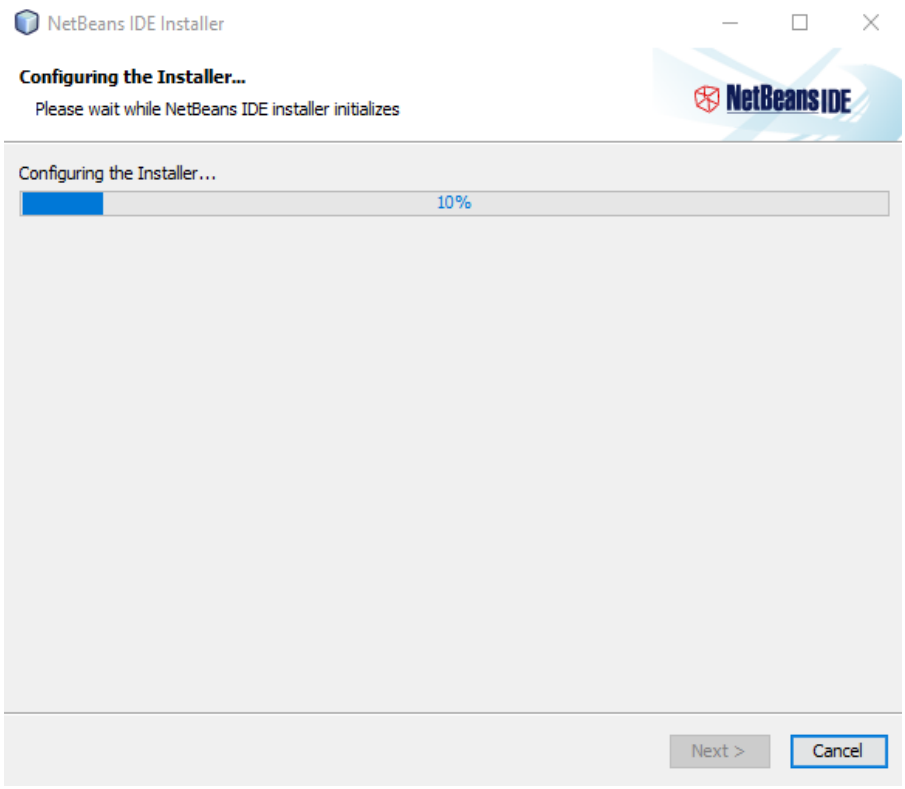

Figura 17. Configuración del instalador NetBeans

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

Figura 18. Características a instalar NetBeans

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

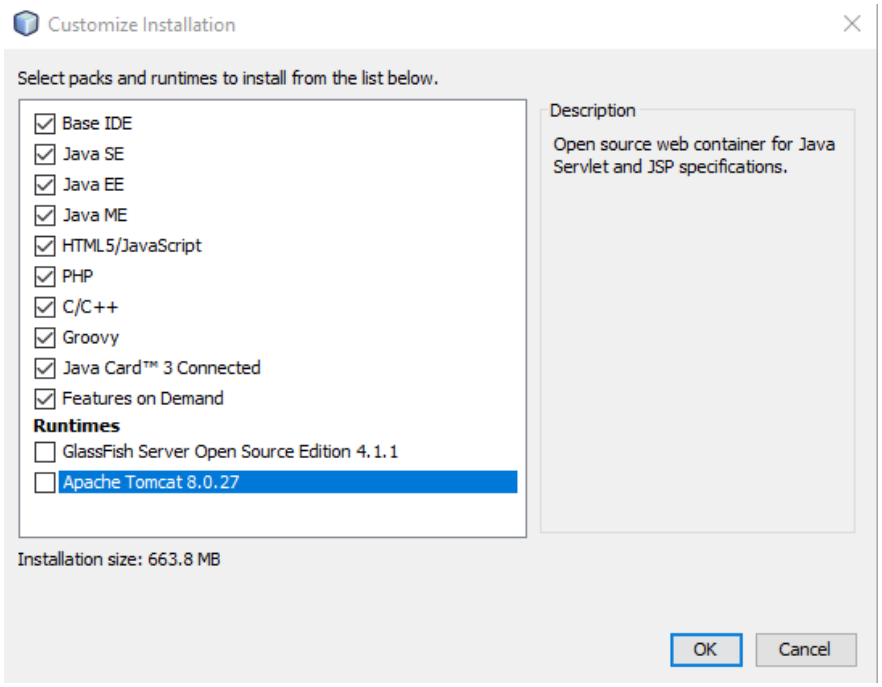

Figura 19. Selección de paquetes y servidores a instalar

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

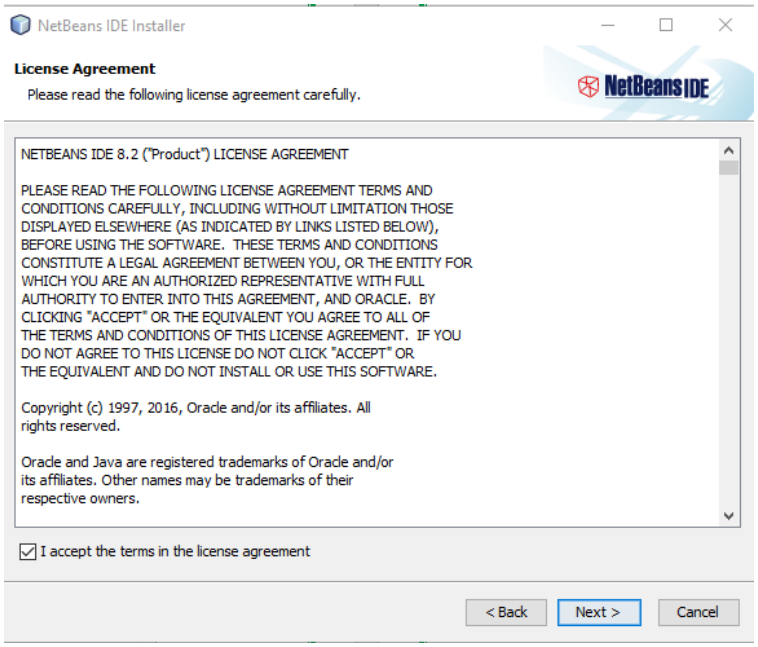

Figura 20. Aceptación de términos

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

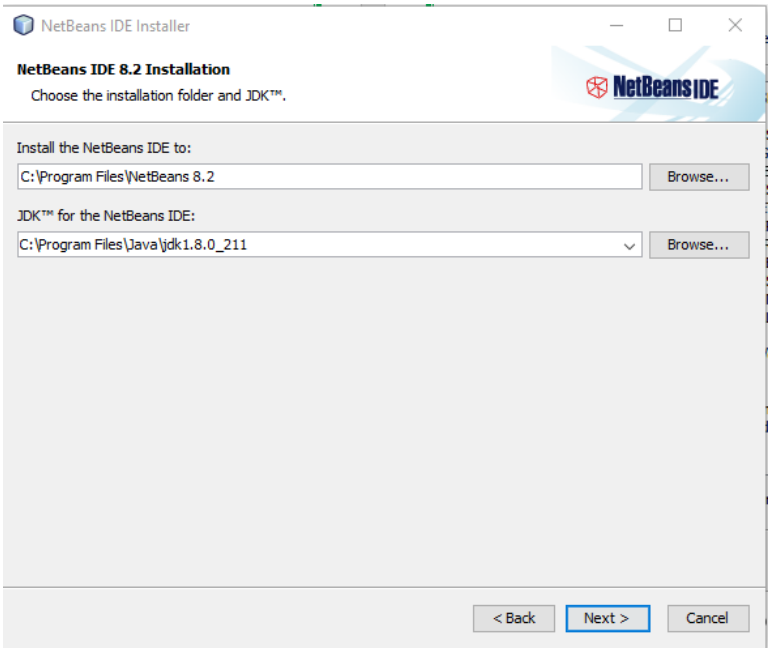

Figura 21. Selección de carpeta de instalación

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

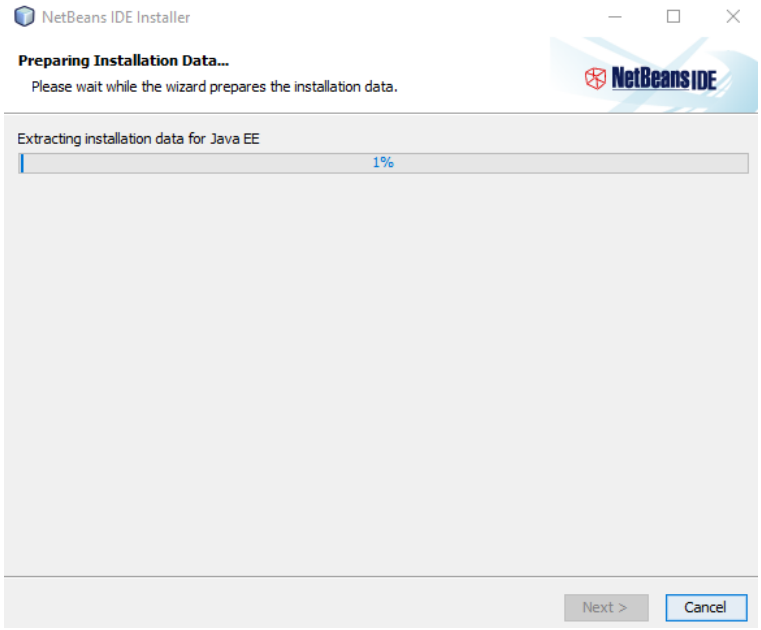

Figura 22. Instalación de paquetes seleccionados

15. Se instalarán todos los paquetes seleccionados previamente.

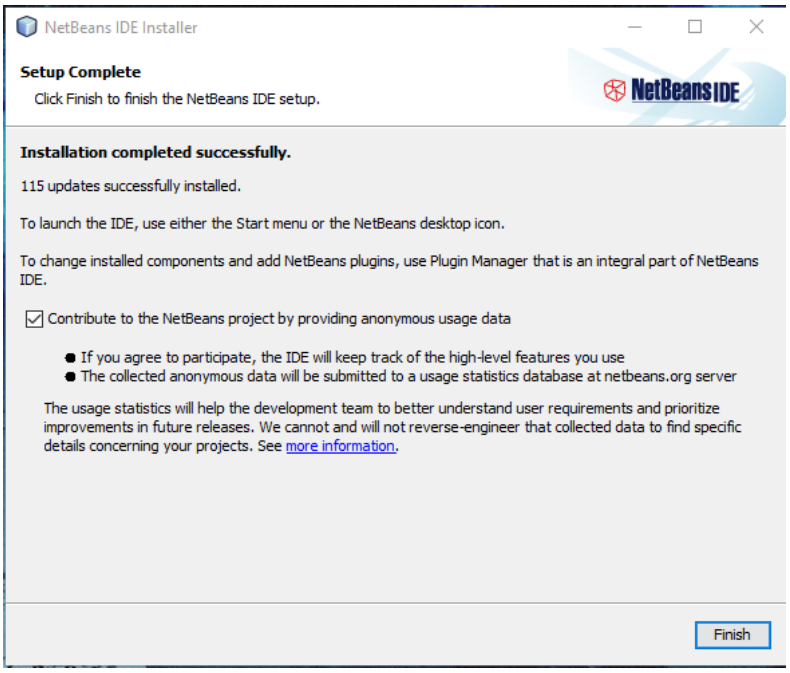

Figura 23. Finalización de la instalación

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

DESARROLLO DE LA APLICACIÓN

CREACIÓN DEL PROYECTO

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

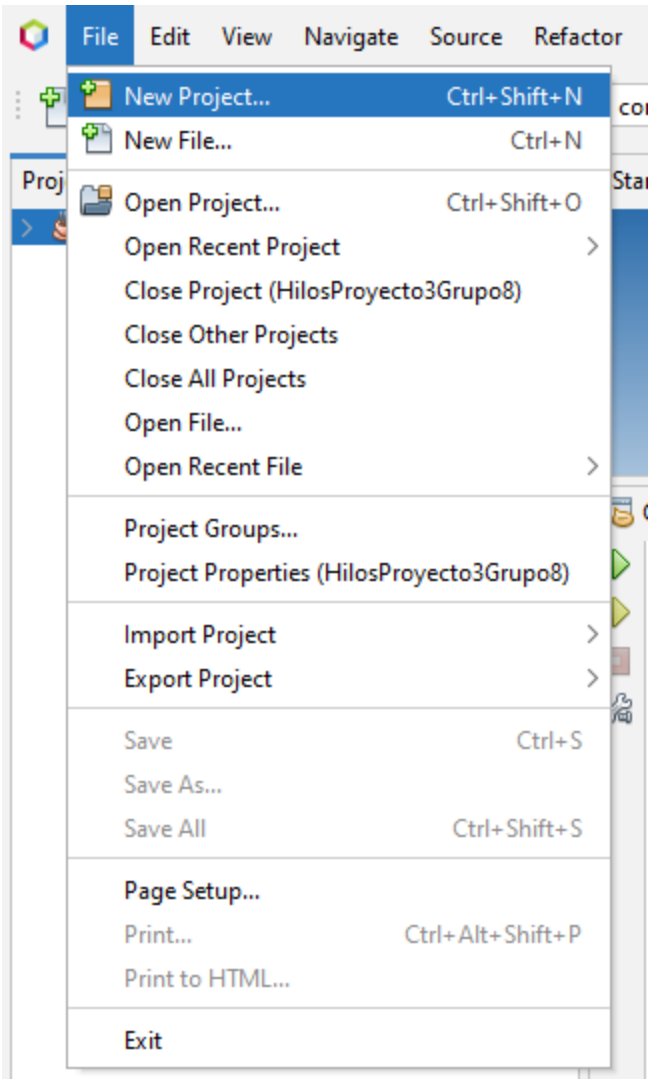

Figura 24. Crear un nuevo proyecto

2. Se selecciona la opción "Java with Ant" y, a continuación, se elige "Java application". Luego, se procede a hacer clic en el botón "Siguiente".

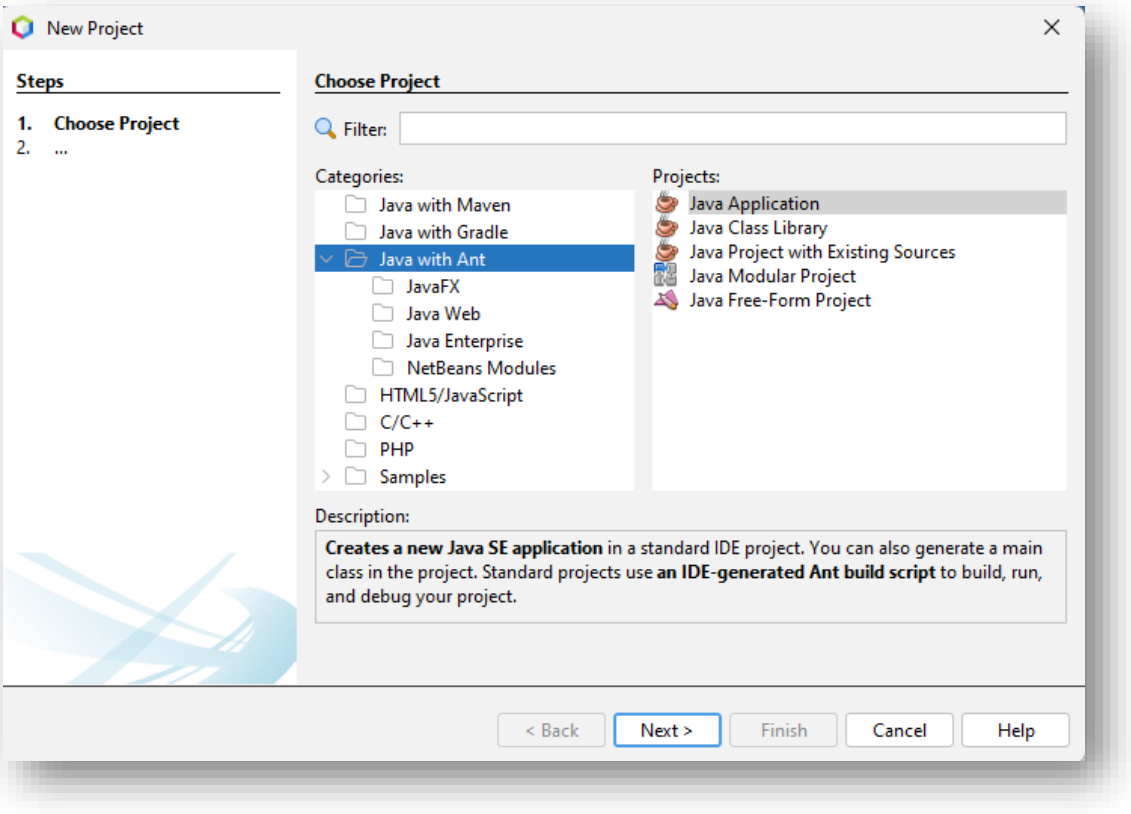

Figura 25. Selección del tipo de proyecto a crear

3. Finalmente se crea el proyecto y se le da el nombre de “UsoThreads”

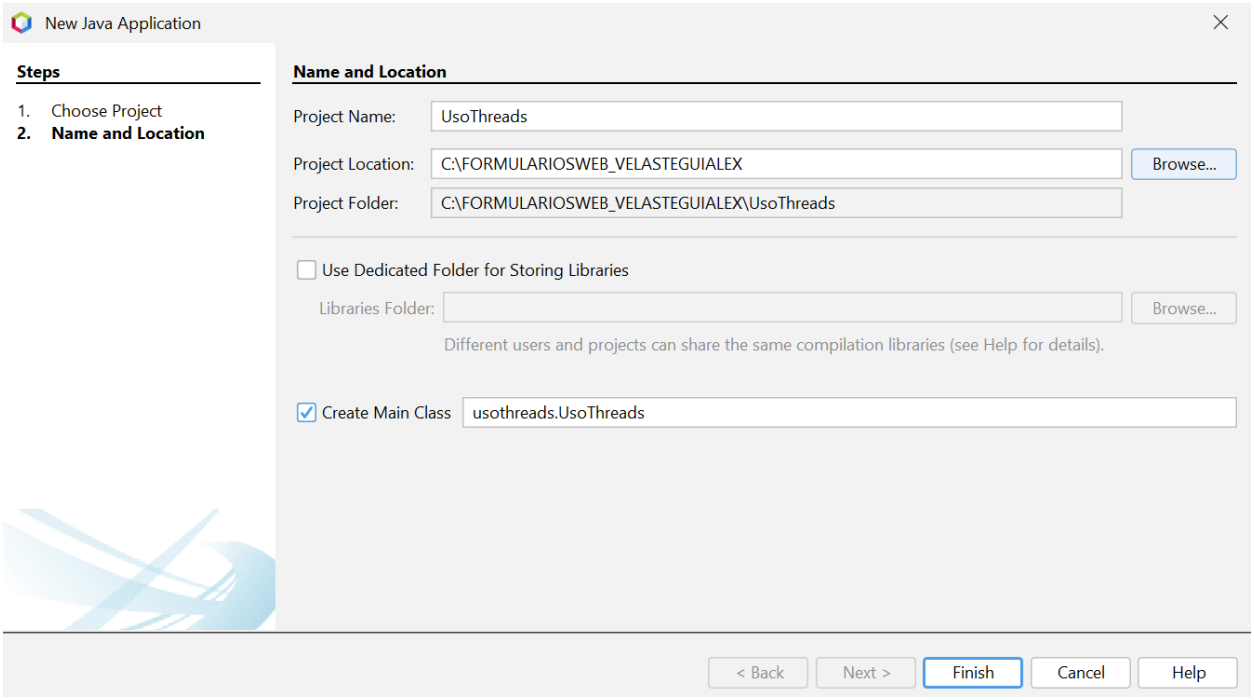

Figura 26. Asignación de nombre al proyecto

Una vez realizado estos pasos lo primero que se obtendrá será el archivo vacío "UsoThreads.java" como se muestra a continuación.

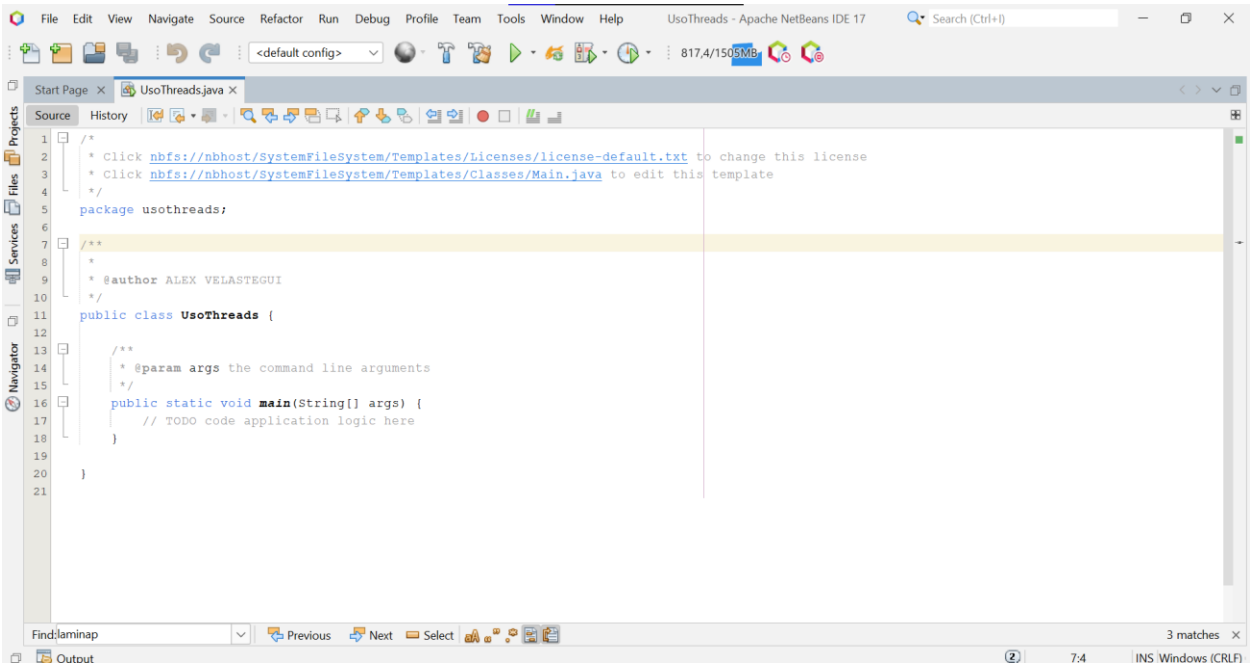

Figura 27. Archivo vacío "UsoThreads.java"

Para comenzar en la realización del proyecto primero se agregan las importaciones necesarias.

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Container;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
```

Figura 28. Importaciones

Después de esto en la clase principal “main” se tiene un método, en el cual se instancia el MarcoRebote(), y a su vez se agrega que se cierra al momento de aplastar la “X” de la esquina superior derecha, la ventana se cierre, utilizando la función marco.setDefaultCloseOperation. Y por último que se vuelva visible con setVisible.

```
public class UsoThreads {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		JFrame marco=new MarcoRebote();

		marco.setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);

		marco.setVisible(b: true);

	}

}
```

Figura 29. Clase principal main

Después se tiene un Clase “Pelota”, la cuál se encarga del movimiento de la pelota, de que cuando se encuentre con un límite en la lámina este rebote, y esto se realiza con un método mueve_pelota en el cuál se recibo por parámetro un objeto de tipo Rectangle2D, el cuál recibe las dimensiones de la lámina, y estas dimensiones(ancho y alto) serán guardadas.

Una vez hecho esto lo primero que se realiza es incrementar las coordenadas x e y, y luego con una serie de métodos pertenecientes a la clase Rectangle2D por ejemplo: getMinX ,getMinY, getMaxX, getMaxY. Se utilizan para detectar cuál es el punto máximo y mínimo tanto en el eje de las x e y, para que cuando se encuentre con los límites se inviertan las coordenadas.

```
class Pelota {
    private static final int TAMX = 15;
    private static final int TAMY = 15;
    private double x = 0;
    private double y = 0;
    private double dx = 1;
    private double dy = 1;
    private Color color;

    public Pelota(Color color) {
        this.color = color;
    }

    // Mueve la pelota invirtiendo posicionn si choca con los limites

    public void mueve_pelota(Rectangle2D limites) {
        x += dx;
        y += dy;

        if (x < limites.getMinX()) {
            x = limites.getMinX();
            dx = -dx;
        }

        if (x + TAMX >= limites.getMaxX()) {
            x = limites.getMaxX() - TAMX;
            dx = -dx;
        }

        if (y < limites.getMinY()) {
```

Figura 30. Clase Pelota 1

Por último se le agrego una clase getColor(), la cuál retornara el color que va a ser alternado por cada pelota lanzada.

```
66          }
67
68          if (y < limites.getMinY()) {
69              y = limites.getMinY();
70              dy = -dy;
71          }
72
73          if (y + TAMY >= limites.getMaxY()) {
74              y = limites.getMaxY() - TAMY;
75              dy = -dy;
76          }
77      }
78
79      //Forma de la pelota en su posicion inicial
80      public Ellipse2D getShape() {
81          return new Ellipse2D.Double(x, y, w: TAMX, h: TAMY);
82      }
83
84      public Color getColor() {
85          return color;
86      }
87  }
```

Figura 31. Clase Pelota 2

Debajo de la clase pelota, se crea la clase LaminaPelota, que construye la lámina por la cuál va a moverse la pelota, utilizando los componentes paintComponent(), la clase Graphics y

Graphics2D, y el método fill se consigue pintar la pelota. Y con el método setColor se lográ darle colores diferentes a la pelota.

```
}
// Lamina que dibuja las pelotas-----------------------------------------
class LaminaPelota extends JPanel {
    private List<Pelota> pelotas = new ArrayList<Pelota>();

    //Anadimos pelota a la lamina
    public void add(Pelota b) {
        pelotas.add(e: b);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        for (Pelota b : pelotas) {
            g2.setColor(c: b.getColor());
            g2.fill(s: b.getShape());
        }
    }

}
```

Figura 32. Clase LaminaPelota

En la clase MarcoRebote, se construye el marco y se agregan las laminas tanto como la de la pelota en la parte superior y la de los botones en la parte inferior. En la lámina inferior se tiene 2 botones: el primero es el botón “Dale!” del cuál saldrá la pelota a realizar su recorrido y el segundo botón es “Salir” que así como su nombre lo dice saldrá de la aplicación.

```
public MarcoRebote() {
    setBounds(x: 600, y: 300, width: 400, height: 350);
    setTitle(title: "Rebotes");
    lamina = new LaminaPelota();
    add(comp: lamina, constraints:BorderLayout.CENTER);
    JPanel laminaBotones = new JPanel();
    boton = ponerBoton(c: laminaBotones, titulo: "Dale!", new ActionListener() {
        public void actionPerformed(ActionEvent evento) {
            comienza_el_juego();
        }
    });

    ponerBoton(c: laminaBotones, titulo: "Salir", new ActionListener() {
        public void actionPerformed(ActionEvent evento) {
            System.exit(status: 0);
        }
    });
    add(comp: laminaBotones, constraints:BorderLayout.SOUTH);
}
```

Figura 33. clase MarcoRebote

Se define una clase PelotaHilos que implementa la interfaz Runnable. Esto significa que se puede ejecutar en un hilo. La clase tiene dos atributos: pelota y componente. pelota es una instancia de la clase Pelota, que probablemente contiene la lógica para mover la pelota. componente es una instancia de la clase Component, que representa el componente gráfico en el que se dibujará la pelota.

El método run() es parte de la interfaz Runnable y contiene el código que se ejecutará en el hilo. En un bucle for, se realiza el movimiento de la pelota llamando al método mueve_pelota() de la instancia pelota. El método mueve_pelota() probablemente toma los límites del componente gráfico como parámetro para asegurarse de que la pelota se mueva dentro de esos límites. Después de mover la pelota, se llama al método repaint() del componente para repintar el componente y mostrar la nueva posición de la pelota en la pantalla. Luego, el hilo se duerme durante 4 milisegundos usando Thread.sleep(4). Esto hace que el hilo se detenga brevemente antes de continuar con el siguiente ciclo del bucle.

Si se produce una interrupción mientras el hilo está durmiendo, se captura la excepción InterruptedException y se imprime la traza de la pila de la excepción.

```
class PelotaHilos implements Runnable {
    private Pelota pelota;
    private Component componente;

    public PelotaHilos(Pelota unaPelota, Component unComponente) {
        this.pelota = unaPelota;
        this.componente = unComponente;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 3000; i++) {
            pelota.mueve_pelota(limites:componente.getBounds());
            componente.repaint();
            try {
                Thread.sleep(millis: 4);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Figura 34. Clase PelotaHilos

El método ponerBoton() recibe tres parámetros: c, que es un contenedor en el que se agregará el botón, titulo, que es el texto que se mostrará en el botón, y oyente, que es un objeto ActionListener que escuchará los eventos del botón. Se crea una instancia de JButton llamada boton pasando el titulo como argumento. El botón boton se agrega al contenedor c utilizando el método add() del contenedor. El contenedor c se configura con un color de fondo Color.DARK_GRAY utilizando el método setBackground(). El botón boton se configura con un color de fondo Color.GREEN.darker() utilizando el método setBackground(). El método darker() se utiliza para obtener una versión más oscura del color verde. El botón boton se configura con un color de texto Color.WHITE utilizando el método setForeground(). Se agrega el oyente al botón utilizando el método addActionListener() para que el botón pueda generar eventos cuando se hace clic en él. Y Finalmente, se devuelve el objeto boton.

```java
//Ponemos botones
public JButton ponerBoton(Container c, String titulo, ActionListener oyente) {
    JButton boton = new JButton(text: titulo);
    c.add(comp: boton);
    c.setBackground(c: Color.DARK_GRAY);
    boton.setBackground(bg: Color.GREEN.darker());
    boton.setForeground(fg: Color.WHITE);
    boton.addActionListener(l: oyente);
    return boton;
}
```

Figura 35. ponerBoton

Se crea una instancia de Random llamada random para generar valores aleatorios. Se crea una instancia de Color llamada color utilizando random.nextInt(256) para generar valores aleatorios entre 0 y 255 para los componentes rojo, verde y azul del color RGB. La instancia de Pelota llamada pelota pasando el color como argumento. Se agrega la pelota a una instancia de lamina, que probablemente sea un panel o lienzo en la interfaz gráfica. Se configura el color de fondo del botón boton con el color generado utilizando setBackground(). Se establece setOpaque(true) para que el fondo del botón sea opaco y se oculte cualquier contenido detrás del botón. Se establece setBorderPainted(false) para eliminar el borde del botón. Se crea una instancia de PelotaHilos pasando pelota y lamina como argumentos, y se asigna a la variable r de tipo Runnable. Se crea una instancia de Thread llamada t pasando r como argumento. Se inicia el hilo t llamando al método start(). Esto ejecutará el método run() de la clase PelotaHilos en un hilo separado.

```java
//Anade pelota y la bota 1000 veces
public void comienza_el_juego() {
    Random random = new Random();
    Color color = new Color(r: random.nextInt(bound: 256), g: random.nextInt(bound: 256), b: random.nextInt(bound: 256));

    Pelota pelota = new Pelota(color);
    lamina.add(b: pelota);
    boton.setBackground(bg: color);
    boton.setOpaque(isOpaque: true);
    boton.setBorderPainted(b: false);

    Runnable r = new PelotaHilos(unaPelota: pelota, unComponente: lamina);
    Thread t = new Thread(target: r);
    t.start();
}
```

Figura 36. Metodo comienza_el_juego()

CÓDIGO

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

package com.mycompany.usothreads;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Container;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 *
 * @author JOHAO MORALES, MAYCOL TITUAÑA, ALEX VELASTEGUI
 */
public class UsoThreads {
```

```
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        JFrame marco = new MarcoRebote();
        marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        marco.setVisible(true);
    }
}

//Movimiento de la pelota----------------------------------------------------

class Pelota {
    private static final int TAMX = 15;
    private static final int TAMY = 15;
    private double x = 0;
    private double y = 0;
    private double dx = 1;
    private double dy = 1;
    private Color color;

    public Pelota(Color color) {
        this.color = color;
    }

    // Mueve la pelota invirtiendo posicionn si choca con los limites

    public void mueve_pelota(Rectangle2D limites) {
        x += dx;
        y += dy;

        if (x < limites.getMinX()) {
            x = limites.getMinX();
```

```
            dx = -dx;
        }

        if (x + TAMX >= limites.getMaxX()) {
            x = limites.getMaxX() - TAMX;
            dx = -dx;
        }

        if (y < limites.getMinY()) {
            y = limites.getMinY();
            dy = -dy;
        }

        if (y + TAMY >= limites.getMaxY()) {
            y = limites.getMaxY() - TAMY;
            dy = -dy;
        }
    }

    //Forma de la pelota en su posicion inicial
    public Ellipse2D getShape() {
        return new Ellipse2D.Double(x, y, TAMX, TAMY);
    }

    public Color getColor() {
        return color;
    }
}
// Lamina que dibuja las pelotas----------------------------------------------
class LaminaPelota extends JPanel {
    private List<Pelota> pelotas = new ArrayList<Pelota>();
```

```
    //Anadimos pelota a la lamina
    public void add(Pelota b) {
        pelotas.add(b);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        for (Pelota b : pelotas) {
            g2.setColor(b.getColor());
            g2.fill(b.getShape());
        }
    }

}

//Marco con lamina y botones----------------------------------------------------
class MarcoRebote extends JFrame {
    private LaminaPelota lamina;
    private JButton boton;

    public MarcoRebote() {
        setBounds(600, 300, 400, 350);
        setTitle("Rebotes");
        lamina = new LaminaPelota();
        add(lamina, BorderLayout.CENTER);
        JPanel laminaBotones = new JPanel();
        boton = ponerBoton(laminaBotones, "Dale!", new ActionListener() {
            public void actionPerformed(ActionEvent evento) {
                comienza_el_juego();
```

```
        }
    });

    ponerBoton(laminaBotones, "Salir", new ActionListener() {
        public void actionPerformed(ActionEvent evento) {
            System.exit(0);
        }
    });
    add(laminaBotones, BorderLayout.SOUTH);
}

class PelotaHilos implements Runnable {
    private Pelota pelota;
    private Component componente;

    public PelotaHilos(Pelota unaPelota, Component unComponente) {
        this.pelota = unaPelota;
        this.componente = unComponente;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 3000; i++) {
            pelota.mueve_pelota(componente.getBounds());
            componente.repaint();
            try {
                Thread.sleep(4);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

```
        }
    }

    public JButton ponerBoton(Container c, String titulo, ActionListener oyente) {
        JButton boton = new JButton(titulo);
        c.add(boton);
        c.setBackground(Color.DARK_GRAY);
        boton.setBackground(Color.GREEN.darker());
        boton.setForeground(Color.WHITE);
        boton.addActionListener(oyente);
        return boton;
    }

    //Anade pelota y la bota 1000 veces
    public void comienza_el_juego() {
        Random random = new Random();
        Color color = new Color(random.nextInt(256), random.nextInt(256),
random.nextInt(256));

        Pelota pelota = new Pelota(color);
        lamina.add(pelota);
        boton.setBackground(color);
        boton.setOpaque(true);
        boton.setBorderPainted(false);

        Runnable r = new PelotaHilos(pelota, lamina);
        Thread t = new Thread(r);
        t.start();
    }
}
```

EJECUCIÓN DEL PROYECTO

Pelotas

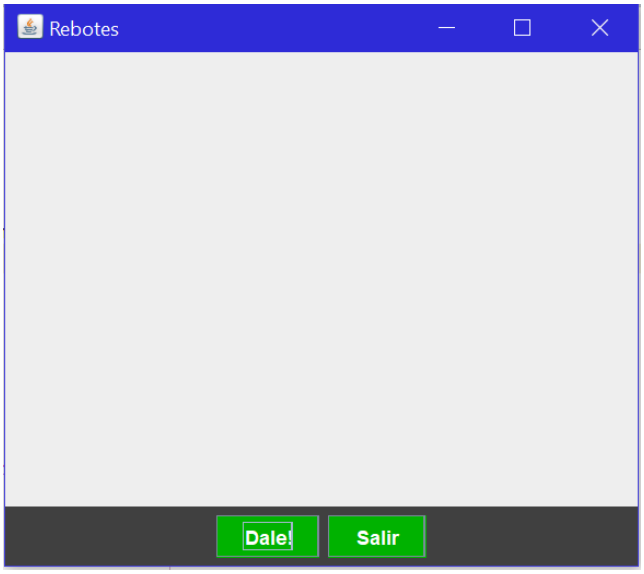

Figura 37. Forma Inicial

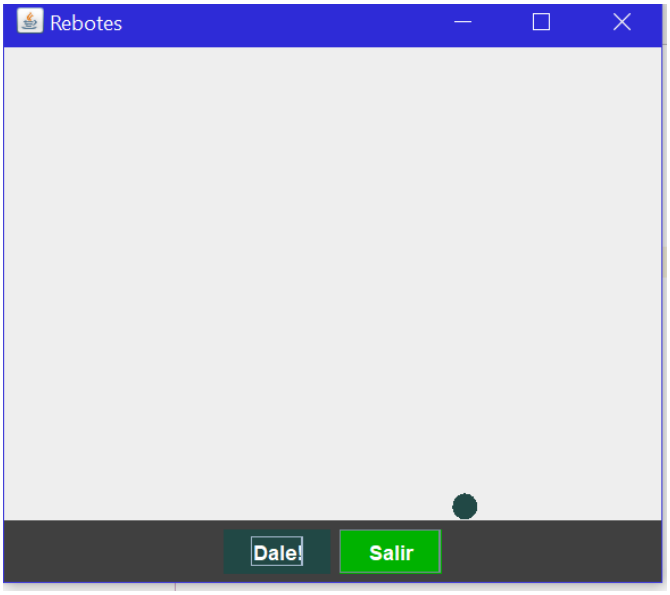

Figura 38. Ejecución con una pelota

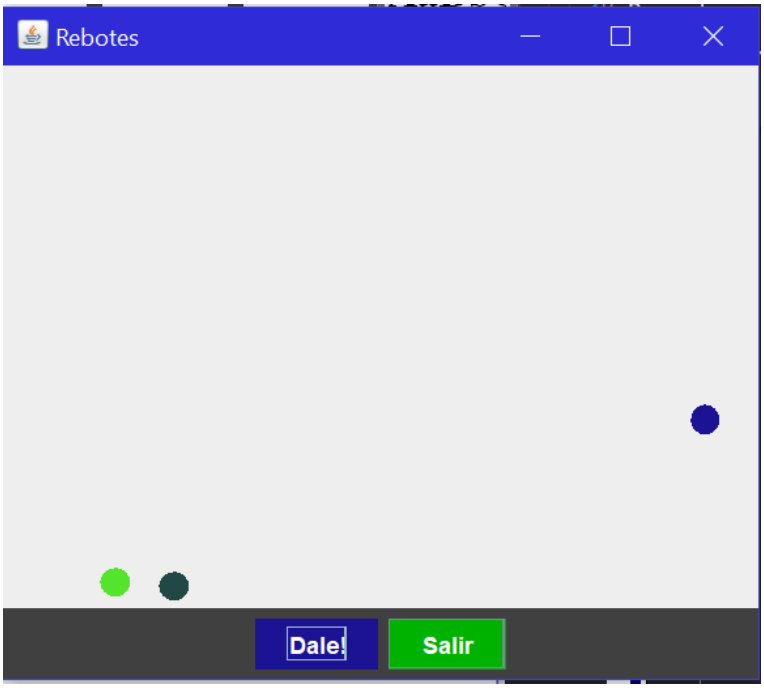

Figura 39. Ejecución con varias pelotas

CONCLUSIONES

- En este trabajo se ha desarrollado un modelo de Banco utilizando el lenguaje de programación Java. Este modelo incluye la creación de cuentas, transferencias y el cálculo del saldo total, brindando una base sólida para aplicaciones bancarias.
- La aplicación de conceptos de programación orientada a objetos, como la encapsulación, la herencia y el polimorfismo, ha facilitado la organización y el modularidad del código, mejorando su mantenibilidad y reutilización.
- El uso de herramientas como Apache NetBeans ha agilizado el desarrollo del proyecto, proporcionando un entorno de desarrollo integrado con características útiles para la codificación, depuración y gestión de proyectos Java.
- El programa implementa una aplicación gráfica de rebotes de pelotas utilizando hilos en Java. Permite al usuario iniciar y detener la animación de las pelotas en movimiento.
- Se utilizan las bibliotecas java.awt y javax.swing para crear los componentes gráficos, como el marco de la ventana, los botones y el lienzo donde se dibujan las pelotas.

RECOMENDACIONES

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

- Se recomienda que al momento de poner mas pelotas en la pantalla, estás tengan diferentes colores para que no exista ninguna confusión por parte del usuario al momento de agregar más pelotas.

BIBLIOGRAFÍA

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

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

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

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

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

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

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

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

[9] Oracle, “How to Write an Action Listener”, Recuperado el 08 de Julio 2023, Disponible en: https://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html

[10] Oracle, “Class Graphics”, Recuperado el 08 de Julio 2023, Disponible en: https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/java/awt/Graphics.html

[11] TutorialsPoint, “AWT Graphics2D Class”, Recuperado el 08 de Julio 2023, Disponible en: https://www.tutorialspoint.com/awt/awt_graphics2d_class.html.

[12] Oracle, “Class Color”, Recuperado el 08 de Julio 2023, Disponible en: https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/java/awt/Color.html

[13] Andrew1234, “Ellipse2D”, Recuperado el 08 de Julio 2023 de GeeksforGeeks, Disponible en: https://www.geeksforgeeks.org/java-awt-ellipse2d/

[14] Oracle, “Class Rectangle2D”, Recuperado el 08 de Julio 2023, Disponible en: https://docs.oracle.com/javase/8/docs/api/java/awt/geom/Rectangle2D.html

[15] Oracle, “Class Random”, Recuperado el 08 de Julio 2023, Disponible en: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

[16] JavaTpoint, “Java Swing Tutorial”, Recuperado el 08 de Julio 2023, Disponible en: https://www.javatpoint.com/java-swing