UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA



FACULTAD DE INGENIERÍA Y CIENCIAS MATEMÁTICAS

**PROGRAMACIÓN I (CCC104), SECC. (852)**

**PROYECTO FINAL:**

**EXPLICACIÓN E INVESTIGACIÓN PARA PROYECTO FINAL**

PARCIAL II

MAYRA ALEJANDRA SALAZAR RODRÍGUEZ (11511306)

TRIMESTE II

SEMESTRE I

20/06/ 2016

CAMPUS TEGUCIGALPA

**Tabla de Contenidos**

[**Proyecto Final - Explicación e Investigación del Proyecto Final** 3](#_Toc454175452)

[**I - Adivina Letras** 3](#_Toc454175453)

[**II – Cálculo de Índice de Masa Corporal** 3](#_Toc454175454)

[**III – Threads** 4](#_Toc454175455)

[***Creación y ejecución de Threads*** 4](#_Toc454175456)

[***Uso de Subclase*** 5](#_Toc454175457)

[***Ejecución del método run()*** 5](#_Toc454175458)

[***Implementación de la Interfaz Runnable*** 5](#_Toc454175459)

[**IV – Snake** 6](#_Toc454175460)

[***Barra de Menu*** 7](#_Toc454175461)

[***Instanciando la clase Snake*** 7](#_Toc454175462)

[***Instanciando una snake*** 8](#_Toc454175463)

[***Metodo para hacer crecer el snake*** 8](#_Toc454175464)

[**V- X0 Player vs Player** 9](#_Toc454175465)

**Proyecto Final - Explicación e Investigación del Proyecto Final**

El proyecto que se presenta contiene los siguientes programas de código:

* Adivina letras.
* Cálculo de Índice de Masa Corporal de las personas.
* Un juego de Snake.
* Un X0 Player vs Player.
* Una clase Main donde se ejecutan todas las clases del proyecto.
* Varias clases que contienen los códigos del proyecto.

**I - Adivina Letras**

En este código se utilizó un método **public char[] charRandom()** que contiene un arreglo de caracteres de letras el cual mete de manera random 4 caracteres a otro arreglo. Este arreglo se regresa al main, y se manda como argumento a otro método **public void encontrarChar(char carac)** el cual busca un carácter ingresado por el usuario en el arreglo creado en charRandom(). El arreglo que se creó en charRandom() es leído por el método recursivo **public String leerArray(char[] array, int inicioArray)** que devuelve una String la cual es leída por el JOptionPane.

**II – Cálculo de Índice de Masa Corporal**

Durante los últimos años, el porcentaje de la población con sobrepeso ha aumentado considerablemente. La obesidad puede causar diversos trastornos de salud como enfermedades cardiovasculares, diabetes, apnea del sueño, padecimientos de la vesícula, esofagitis, osteartrosis, osteoporosis, esteatosis hepática, problemas ortopédicos y disfunciones renales y muchas de estas complicaciones pueden llevar a la muerte.

Es por esto que se han ideado cálculos para verificar si una persona está en sobrepeso o no. Una medida de la obesidad se determina mediante el índice de masa corporal (IMC), que se calcula dividiendo los kilogramos de peso por el cuadrado de la estatura en metros (IMC = peso [kg]/ estatura [m2]). Según el Instituto Nacional del Corazón, los Pulmones y la Sangre de los Estados Unidos (NHLBI), el sobrepeso se define como un IMC de más de 25 (Texas Heart Institute, 2015).

En la clase **Main** del proyecto, se hizo un método **public static float indiceMasaCorporal( float peso, float metros)** la cual retorna el cálculo del bmi (siglas en inglés de Body Mass Index) en la clase main y el método **public static double promedioBMI( double acumulador, int numPersonas)** retorna un double con el promedio de los índices de masa corporal.En el **case 2** del **switch**, pedirá varias veces los cálculos hasta el usuario desee salir del ciclo (por medio de respuesta de usuario). Al final, se dan el número de personas desnutridas y el número de personas con sobrepeso.

**III – Threads**

***Creación y ejecución de Threads***

\*Tanto el Snake como el X0 utilizan lo que son Threads.

Una thread es un único flujo de control dentro de un programa. Algunas veces es llamado contexto de ejecución porque cada thread debe tener sus propios recursos, como el program counter y el stack de ejecución, como el contexto de ejecución. Sin embargo, toda thread en un programa aún comparte muchos recursos, tales como espacio de memoria y archivos abiertos. Threads también son llamadas procesos livianos (lightweight process) (Departamento de Electronica, 2010).

Es decir, un Thread es un trozo de código de nuestro programa que puede ser ejecutado al mismo tiempo que otro (Invarato, 2013).

Hay dos formas de hacer una tarea correr concurrentemente con otra: crear una nueva clase como subclase de la clase Thread o declarar una clase e implementar la interfaz Runnable.

***Uso de Subclase***

Cuando se crea una subclase de Thread, la subclase debería definir su propio método run() para sobre montar el método run() de la  clase Thread. La **tarea concurrente es desarrollada en este método run**().

***Ejecución del método run()***

Una instancia de la subclase es creada con new, luego llamamos al método start() de la thread para hacer que la máquina virtual Java ejecute el método run(). Ojo para iniciar la concurrencia invocamos a start(), así invocamos a run() en forma indirecta. Si invocamos a run() directamente, se comportará como el llamado a cualquier método llamado dentro de un mismo hilo (sin crear uno independiente).

***Implementación de la Interfaz Runnable***

La interfaz Runnable requiere que sólo un método sea implementado, el método run(). Primero creamos una instancia de esta clase con new, luego creamos una instancia de Thread con otra sentencia new y usamos el objeto recién creado en el constructor. Finalmente, llamamos el método start() de la instancia de Thread para iniciar la tarea definida en el método run().

Una instancia de una clase que defina el método **run()** - ya sea como subclase de Thread o implementando la interfaz Runnable - debe ser pasada como argumento en la creación de una instancia de Thread. Cuando el método **start()** de esta instancia es llamado, Java run  time sabe qué método **run()** ejecutar.

**IV – Snake**

La clase Snake deriva de **JFrame** de manera en la que sea una JFrame pero con las características que deseemos, ya que si creamos nuestro juego utilizando una JFrame de swing no nos permite modificar sus propiedades porque java los declara como instancia privada.

Seguidamente utilizamos **implements** para aplicar interfaces de java en la clase. Las interfaces de nuestro interés son **KeyListener** que es la interfaz encargada de hacer que nuestra clase sea sensible a eventos del teclado, de esa forma utilizamos inputs del teclado y **Runnable**.

La interfaz **Runnable** debe ser implementada por cualquier clase cuyas instancias están destinadas a ser ejecutadas por un hilo o Thread (Oracle.com, 2013).

Inicializamos las variables que necesitaremos como atributos de la clase Snake:

Declaramos **p1** y **p2** de tipo JPanel refiriéndose a Panel 1 y a Panel 2 que ocuparan un espacio en la ventana. También se necesita un arreglo de tipo **Jbutton** que será el arreglo de botones que conforma a nuestro snake. **Bonusfood** es el botón que se instanciará para la comida del snake. La variable de tipo **JTextArea** es la que utilizamos para incluir texto en cualquier Panel. Las variables de tipo **int** serán necesarias para definir valores cuánticos en la clase snake, tales como **gu** refiriéndose al tamaño del snake, **speed** refiriéndose a la velocidad con la que descansa el thread, **score** refiriéndose al puntaje y la dirección del snake ya sea de izquierda a derecha en **x** o arriba hacia abajo en **y**. Los arreglos **lbx** y **lby** de tipo int que representan las coordenadas x y de nuestra ventana. También necesitamos declarar un Thread o hilo, necesario para el ciclo del juego. Las **variables booleanas** son para validar la dirección en la que el usuario desea dirigir al snake. Finalmente el **JmenuBar** y sus elementos.

La clase Snake necesita una función pública que solo se encarga de iniciar los valores de cada variable declarada, también es útil para cuando se requiera reiniciar el juego.

***Barra de Menu***

EL método **reset**() utilizado para el evento de New Game simplemente llama el método de initializeValues() para restablecer los valores de las variables, detener el hilo, instanciar una nueva snake, setear las puntuación del Texto y crear un nuevo hilo que cumplirá su función como nuevo ciclo del juego.

***Instanciando la clase Snake***

La clase Snake también se encarga de iniciar el KeyListener para que empiece a esperar inputs del teclado y finalmente instancia e inicia el Thread encargado del ciclo del juego.

***Instanciando una snake***

Nuestra función de crear una snake es un **for** donde la condicion del for debe ser igual a la variable **gu** la cual se refiere al tamaño inicial que tendrá el gusanito. En el for se van instanciando botones que constituyen cada vertebra de la snake, donde gracias a las variables **lbx** y **lby** se le dice en que coordenadas de la ventana se colocaran estos botones y se define su color. Por ultimo de define que por default el snake avanzará en dirección a la derecha al escribir **lbx[i + 1] = lbx[i] – 10** y que mantendrá su posición en y igual.

***Metodo para hacer crecer el snake***

Como ya se sabe lb es el arreglo de botones que conforman el cuerpo del snake y para hacerlo crecer le ponemos gu que es el tamaño actual del gusanito como posición del arreglo y le creamos un nuevo botón que luego se deshabilita para evitar que se le dé clic. Por ultimo se añade al panel 1.

La variable **r** es nuestra variable random, **a** y **b** serán las coordenadas donde aparecerá la comida nueva después de ser comida porque **void growup()** implica que el snake ha consumido un botón de comida. Al añadirlo en el array del lbx y lby ya tenemos ese valor numérico para esa coordenada. Por último se le suma 1 a la variable que determina la longitud del snake.

En el Main, el juego de Snake se manda a llamar mediante su constructor con un **new SnakeGame().**

**V- X0 Player vs Player**

El X0 Player vs Player se hizo creando un **JFrame container** desde el paquete donde está el proyecto. Este contenedor sirve para familiarizarse visualmente con la interfaz gráfica (GUI’S Builder Interface) el cual contiene varios botones y opciones que facilita al programador definir la interfaz gráfica de su programa (NetBeans, s.d.).

Se inicializan las variables tipo campo que controlaran el sistema de juego(una matriz donde se montará el juego, booleans, etc). Los botones **JButton** utilizan un **Action Performed**, esto implica que los botones al ser clickeados desarrollarán una función. En este caso, cada vez que se clickean mandan a llamar los métodos **void mover(int x, int y)** y **void llamarBotones()**.

Los botones llaman con **mover(int x, int y)** lo que la cual según la posición que se le dé, (x = filas, y= columnas) de la matriz inicializada, verifica si el espacio está en blanco. Si es así, va cambiando el booleano **playerChoice,** el cual permite que cambien las decisiones entre X y 0. En este mismo método manda a llamar **void elección()** (que cambia el booleano playerChoice), **void ganar()** que verifica quien gana y quien ataca o empatan y **void leerMatriz()** que lee en consola la matriz.

Los botones también llaman a **void llamarBotones()** el cual les da el texto a los JButtons a medida vayan cambiando los jugadores y decisiones.

Para llamarlo al Main, se tuvo que crear un arreglo de **String[]** vacío (ya que el main pide como tal como parámetro) y se manda a llamar con la variable de referencia **X0.main(arg)**.

**Bibliografía**

BrandonioProductions (Diretor). (2013). *Making Snake in Java: Part 1 - Planning/Getting Started* [Filme Cinematográfico]. Acesso em Junio de 2016, disponível em https://www.youtube.com/watch?v=FABTl1Q1byw

BrandonioProductions (Diretor). (2013). *Making Snake in Java: Part 2 - Creating structure* [Filme Cinematográfico]. Acesso em 2016 de Junio, disponível em https://www.youtube.com/watch?v=kTSYU3z3vT8

BrandonioProductions (Diretor). (2013). *Making Snake in Java: Part 3 - Finishing Logic* [Filme Cinematográfico]. Acesso em Junio de 2016, disponível em https://www.youtube.com/watch?v=JA3tR3A6Tuo

BrandonioProductions (Diretor). (2013). *Making Snake in Java: Part 4 - Making the Applet and Getting Input* [Filme Cinematográfico]. Acesso em Junio de 2016, disponível em https://www.youtube.com/watch?v=07UdiMlvDcI

Bustard, J. (Diretor). (2014). *Programming Snake in Java!* [Filme Cinematográfico]. Acesso em Junio de 2016, disponível em https://www.youtube.com/watch?v=S\_n3lryyGZM

Departamento de Electronica. (2010). *Hilos (Threads) en Java.* Acesso em 19 de Junio de 2016, disponível em Departamento de Electrónica - Universidad Técnica Federico Santa Maria: http://profesores.elo.utfsm.cl/~agv/elo330/2s10/lectures/Java/threads/JavaThreads.html

Invarato, R. (23 de Abril de 2013). *Multitaréa e Hilos, fácil y muchas ventajas.* Acesso em 19 de Junio de 2016, disponível em jarroba.com: http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

Khan, A. (Diretor). (2016). *Programming Simple Snake Game in Java Eclipse* [Filme Cinematográfico]. Acesso em Junio de 2016, disponível em https://www.youtube.com/watch?v=9ob\_j8rim2g

NetBeans. (s.d.). *Designing a Swing GUI in NetBeans IDE*. Acesso em Junio de 2016, disponível em NetBeans Docs: https://netbeans.org/kb/docs/java/quickstart-gui.html

News Gadget. (30 de Noviembre de 2010). *Create Game Snake With Netbeans*. Acesso em Junio de 2016, disponível em News Gadget: http://gadgetz4all.blogspot.com/2010/11/membuat-game-snake-dengan-netbeans.html

Oracle.com. (2013). *Interface KeyListener.* Acesso em 19 de Junio de 2016, disponível em Java Platform Standard Ed 7.: https://docs.oracle.com/javase/7/docs/api/java/awt/event/KeyListener.html

Texas Heart Institute. (Octubre de 2015). *Calculadora del índice de masa corporal (IMC).* Acesso em 19 de Junio de 2016, disponível em Texas Heart Institute: http://www.texasheart.org/HIC/Topics\_Esp/HSmart/bmi\_calculator\_span.cfm

**AutoEvaluación**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Contiene | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| [s/n] | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** | **s** |

|  |  |
| --- | --- |
| **Concepto** | **Puntos** |
| **Creativo** | 5 |
| **Innovador** | 5 |
| **Amigable** | 5 |
| **Documentado** | 4 |
| **Robusto** | 5 |
| **Uso de métodos** | 5 |