UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA



FACULTAD DE INGENIERÍA

**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**

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 **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. 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().

[RunnableThread.java](http://profesores.elo.utfsm.cl/~agv/elo330/Java/threads/RunnableThread.java)

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 p1 de tipo JPanel refiriéndose a Panel 1 y a Panel 2 que ocuparan un espacio en la ventana. Tambien 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 refiriendose al tamaño del snake, speed refiriendose a la velocidad con la que descanza el thread, score refiriendose al puntaje y la dirección del snake ya sea de izquiera 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. Tambien necesitamos declarer 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.

***Intanciando una snake***

Nuestra funcion 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 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.

***Lógica de movimiento***

El for del inicio indica las coordenadas en las que se encuentra el snake. La función principal de este método es valodar la secuencia de movimiento del snake que seria de 10 pixeles según la dirección en la cual sea llamado ya sea moviéndose de izquierda a derecha en x o de arriba hacia abajo en y.

Obviamente en la función de movimiento verifica si el gusanito osea gu ha hecho contacto con comida que esté en la coordenada lbx y lby. De ser así le suma 5 puntos al score. Para que crezca food debe ser falso y mas adelante se llama la funcion growup.

Esta parte del método valida si el jugador ha perdido llamando la funcion gameEnd()

gameEnd solamente verifica que la cabeza del snake no toque ciertas coordenadas de la ventana, si es asi se cambia el texto a Game Over y se rompe el ciclo con un Break.

***La lógica de la dirección***

Consiste en reconocer los inputs del teclado encargados de la dirección, las validaciones sirven para determinar en qué direcciones puede cambiar dependiendo de la dirección en la que avanza el snake.

**V- X0 Player vs Player**

**Bibliografía**

**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** | 5 |
| **Robusto** | 5 |
| **Uso de métodos** | 5 |