Introducción: Conceptos básicos

Andrés Ramos Seguí y Alejandro Muñoz Plaza

Resumen—Uso de Java y el entorno de programación InteliJ IDEA para llevar acabo las trabajos que se van a desarrollar a lo largo del semestre. Se utilizarán librerías com Swing, para realizar las taréas de interfaces gráficas de usuario. Además se explican los conceptos básicos como la Programación Orientada a Objetos, interfaces, modelo Vista Controlador, concurrencia, costes de computación y estructuras de datos.

Index Terms—Entornos de programación, Librerías, Conceptos básicos LaTeX.

1. Introducción

PARA hacer referencia a nuestro entorno y lenguaje de programación, además de varios de los conceptos más importantes en el mundo de la informática se ha escrito el documento actual.

2. Entorno de programación: InteliJ

Las diferencias entre entornos (IDE) de programación de Java actualmente son más que nada visuales, ya que la gran mayoría de funcionalidades son compartidas entre ellos, más o menos accesibles pero las tienen. En nuestro caso al disponer, gracias a la universidad, de una licencia gratuita del ultimate vamos a utilizar este entorno.

3. librerías para interfaz gráfica: swing

Para la creación de ventanas en Java disponemos de la librería "swing" en ella disponemos de multitud de componentes como JPanel y JFrame (ventanas), JTextBox (campos de texto), JLabel (etiquetas de contenido), entre otros que nos permiten crear un innumerable número de ventanas diferentes. Para su colocación el IDE tiene su propio generador de código en el cual a través de una interfaz gráfica puedes colocar los elementos sin preocuparte por el layout, colocación del elemento por código, y programar los eventos de una forma más simple.

4. Conceptos Básicos

4.1. Importancia del lenguaje

Actualmente hay una gran cantidad de lenguajes de programación de los cuales cada uno se centra en una o varias características en las cuales destacar, por ejemplo Java con la POO y su portabilidad, entre otros o Ada que destaca por la rigurosidad del compilador. Por ello es mejor no centrarse en un lenguaje en concreto y aprender los conceptos básicos de la programación, así cuando vayas a tener que implementarlo te puedes amoldar al lenguaje que te hayan impuesto, por trabajar en determinada empresa, o elegir según tus gustos.

4.2. Programación Orientada a Objetos

La programación orientada a objetos (comúnmente llamada POO) es un paradigma de programación cuyas interacciones se basan en el uso de objetos. Éstos objetos son entidades de una clase con un estado (atributos) y comportamiento (métodos) determinados. Cada objeto tiene un identificador único que lo diferencia del resto (de forma análoga al resto de variables). Un objeto contiene toda la información que permite definirlo e identificarlo inequívocamente frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen de mecanismos de interacción llamados métodos, que favorecen la comunicación entre ellos. Algunos de los lenguajes orientados a objetos más conocidos en la actualidad son Java, C# y Python. La programación orientada a objetos introduce una serie de conceptos particulares que se relacionan entre ellos y nos permiten realizar una representación de acciones u objetos de la vida real. Dichos conceptos son los siguientes:

- **Clase**: una clase es un conjunto de propiedades y de acciones de un objeto.
- **Instanciación**: creación de un objeto a través de la clase correspondiente.
- **Objeto**: instancia de una clase.
- **Mensaje**: la acción en la cual dos objetos (de la misma clase o de clases diferentes) se comunican es denominada mensaje.
- **Método**: un método es un algoritmo (programa) que está asociado a una clase, pueden realizar cambios en los datos y las propiedades de los objetos, generar eventos en el sistema o provocar la creación de un mensaje destinado a un objeto de la aplicación. Constructor: método especial destinado a la creación de un objeto.
- **Evento**: suceso que se produce en el sistema que se maneja enviando un mensaje al objeto correspondiente.
- **Atributo**: características de una clase que la definen. Componentes de un objeto: conjunto de atributos, métodos, relaciones e identidad de un objeto.
- **Identificación de un objeto**: representación en forma de tabla del objeto. Dicha tabla está compuesta por

los atributos y los métodos.
- **Estado interno**: permanece oculta al programador y únicamente puede ser consultada o modificada a través de métodos del objeto.

La programación orientada a objetos ha de cumplir con unos objetivos ayudándose de unas características que todos los lenguajes de POO deben cumplir:

- **Abstracción**: la abstracción es el hecho en el cual un elemento es aislado de su contexto; se pone más énfasis en lo que hace el elemento que en como lo hace.
- **Encapsulamiento**: reunir todos los elementos de una entidad con el mismo nivel de abstracción.
- **Polimorfismo**: hecho en el cual objetos de distintas clases pueden invocar procedimientos aparentemente distintos pero que producirán el comportamiento y las acciones correctas para cada tipo de objeto.
- **Herencia**: una clase obtiene (hereda) los atributos y los métodos de otra clase, dando la sensación de que han sido definidos en la misma; como si de un árbol genealógico se tratase, donde la subclase hija hereda los conceptos de la clase padre.
- **Modularidad**: permite dividir una aplicación en conjuntos más pequeños y manejables llamados módulos. Principio de ocultación: indica que los atributos de una clase que son privados no pueden ser consultados ni modificados desde el exterior, a no ser que se hagan con métodos que lo permitan.

4.3. Modelo Vista Controlador

A la hora de construir nuestras aplicaciones es importante acordar un sistema claro que nos permita construir la arquitectura de manera sencilla y clara delimitando siguientes partes de la aplicación:

- **Modelo**: Es el encargado de gestionar los datos y de gestionar las estructuras de estos.
- **Vista**: Es la parte responsable de la visualización de los datos y de la aplicación con las características gráficas que serán vistas directamente por el usuario
- **Controlador**: Se encarga de gestionar el sistema, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.

Complementaremos esta arquitectura con el patron por eventos, este, consiste en que las tres partes se comunican entre sí mediante el enviamiento y recivimiento de eventos.

4.4. Interficies de Usuario

Las interfaces proporcionan un entorno visual sencillo para permitir la comunicación entre el usuario y la aplicación. La información y las acciones disponibles de la interfaz se ofrecen mediante el uso de imágenes y objetos gráficos.

4.5. Concurrencia

La concurrencia consistente en que dos o más procesos trabajen de manera simultánea de forma que la obtención de sus recursos con los que trabajan es compartida entre

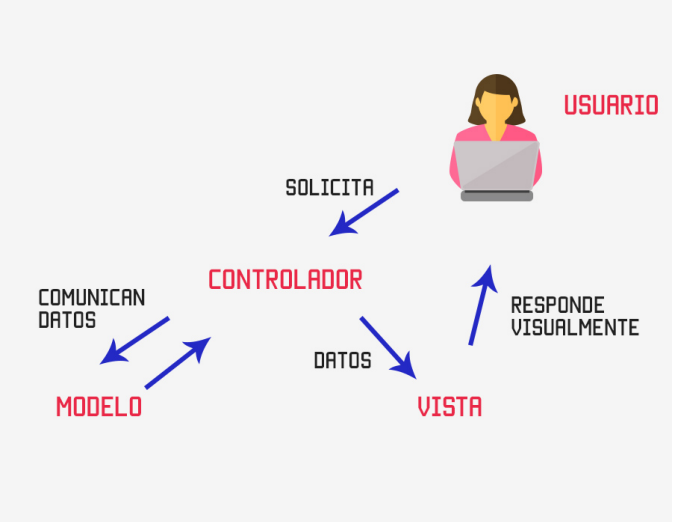

Figura 1. Modelo-Vista-Controlador.

ellos y/o el resultado obtenido es producido por los procesos.Esta interacción provoca que el uso de estos recursos sea usado de forma individual para evitar resultados no esperados. Este problema se conoce como "El problema de la sección crítica", este nombre se debe a que en las secciones en las que se use alguno de estos recursos compartidos es crítico para el resultado.

Por ejemplo, pongamos el caso concreto de la cantidad de dinero que tiene un cliente en el banco y las operaciones sobre su cantidad, ingresos y retiros de dinero de la cuenta. Cuando se realiza un retiro de dinero un programa que realice dicha operación puede ser el siguiente, forzado para ver la idea: [1]

1. double cantidad = consultarDinero(usuario);
2. double retiro = cantidadRetirar();
3. si cantidad - retiro ¿0 entonces
4. retiroDinero(usuario, retiro);

Si tenemos los procesos 1 y 2 que realizan el programa anterior y se da la siguiente ejecución:

1. Proceso 1 → (1) cantidad1 = 500€
2. Proceso 1 → (2) retiro1 = 100€
3. Proceso 2 → (1) cantidad2 = 500€
4. Proceso 2 → (2) retiro2 = 400€
5. Proceso 2 → (3) cantidad2 - retiro2 ¿0 = 500 - 400 ¿0 = True
6. Proceso 2 → (4) usuario.Dinero = cantidad2 - retiro2 = 500 - 400 = 100€FIN; Proceso 2
7. Proceso 1 → (3) cantidad1 - retiro1 ¿0 = 500 - 100 ¿0 = True
8. Proceso 1 → (4) usuario.Dinero = cantidad1 - retiro1 = 500 - 100 = 400€FIN; Proceso 1

Como resultado de los dos procesos el usuario ha sacado 400€y 100€de una cuenta de 500€, se espera a que este usuario tenga la cuenta con 0€pero al realizarse esa ejecución el usuario verá que tiene 400€.

Los requisitos que se deben cumplir las soluciones a este problema son los siguientes:

- **Interbloqueo**: Se produce cuando dos procesos requieren de dos recursos y se encuentran en la situación en la que cada uno de ellos tiene un recurso y necesita el que tiene el otro, por lo tanto como ninguno cede su recurso ninguno de los dos acaba.
- **Espera infinita**: Un proceso espera un tiempo infinito a que un recurso se libere.
- **Exclusión mútua**: Dos procesos no pueden entrar en la sección crítica.

Para poder resolver este problema se han propuestas varias soluciones que cumplen los requisitos anteriores, algunos son:

- **Algoritmo de Dekker.**
- **Algoritmo de Paterson.**
- **Soluciones por Hardware.**
- **Semáforos.**
- **Monitores.**
- **Canales.**

4.6. Eficiencia de los algoritmos

[2] *"Análisis a priori del tiempo de computación que ignora los factores de máquina y lenguaje de programación, centrándonos en el orden de ejecución de las sentencias"*. La anotación más usada es la de **O(f(n))** donde expresa el límite superior del tiempo de ejecución de un algoritmo. Consideraciones del coste asintótico:

- Resulta útil para hacer una primera aproximación de los tiempos de ejecución.
- Hay que tener en cuenta que algunos algoritmos esconden constantes multiplicativas muy grandes.
- Hay que tener presente la eficiencia entre la memoria y el tiempo de ejecución, ya que al optimizar su coste en CPU aumenta su uso de memoria.
- Balanza entre coste de producción y rendimiento del algoritmo. Se trata de buscar el algoritmo que cumpla mejor los requerimientos en el tiempo previsto.

4.7. Estructura de los datos

La estructura en la cual se guardarán los datos es una de las decisiones más importantes a tomar, ya que definirá los tiempos de sus operaciones de gestión. A la hora de elegir qué estructura de datos elegir hay que tener en cuenta los siguientes aspectos:

- **El tamaño del set**: Para un número reducido de datos es preferible el uso de técnicas simples como arrays que ofrecen un tiempo de ejecución menor y son más simples de implementar, mientras que para los grupos grandes de datos las técnicas simples dan tiempos desorbitados.
- **Las claves como tipo discreto**: El uso de índices de tipo numérico para acceder a un array es la mejor opción al superar con creces la cantidad de datos a guardar lo que hay que tomar un control extra para evitar errores.
- **Las claves van relacionadas con el orden**: Si no se da el caso no se puede usar en estructuras como árboles de búsqueda binaria o B+.
- **Las claves tienen una única representación binaria**: Son necesarias para que las funciones de hashing hagan una distribución adecuada.
- **Las claves son arrays de símbolos**: Son las que se adecuan mejor para los "Tries".
- **Los datos se tienen que almacenar en memoria secundaria**: Las técnicas más adecuadas son los árboles B+ y Hashing extensible. Las demás se tienen que adaptar para poder llevar a cabo su almacenamiento en memoria secundaria.
- **Se requiere de recuperar datos de forma ordenada**: Esta operación descarta técnicas como la de Hashing que aun siendo las más eficientes requieren de excesivas modificaciones que ralentizan su obtención.
- **Se requiere eliminar datos**: Se descarta el Hasing cerrado, ya que cuando se produce una concordancia del resultado de la función de hash se crea un puntero al siguiente elemento en la posición de la tabla por ello si eliminamos uno un elemento perdemos la conexión con los que se han almacenado a continuación.
- **Frecuencia de las operaciones a realizar**: Normalmente esta es la razón que determina que estructura al elegir las funciones más comunes que sean las más eficientes.

Tabla de los costes asintóticos de cada operación de las estructuras de datos: [3]

	Empty	Put Remove	Get Update	Get-Ordenado	Unión Intersección Diferencia
Cursores	n	1	1	n	n^2
Array Booleans	n	1	1	n	n
Array Elementos (unsort)	1	n	n	---**	n^2
Array Elementos (sort)	1	n	$\log n$	n	n
Lista punteros (unsrot)	1	n	n	---**	n^2
Lista Punteros (sort)	1	n	n	n	n
Árbol Búsqueda binario	1	$\log n$*	$\log n$*	n	n
Árbol binario balanceado	1	$\log n$	$\log n$	n	n
Árboles B^+	1	$\log n$	$\log n$	n	n
Tries	1	1	1	n	n
Hashing	n	1	1	—***	n

*: Es un valor medio, existen casos de $O(n)$.

**: Se necesita un algoritmo de ordenación, se puede aplicar en la misma estructura.

***: Se necesita un algoritmo de ordenación, pero no se puede aplicar directamente.

Figura 2. Costes asintóticos de operaciones segun la Estructura de datos.

Sin embargo, como se ha comentado en clase, nosotros nos delimitaremos a utilizar estructuras como listas dinámicas y árboles.

Referencias

[1] Miquel Mascaró Oliver. *"2. El problema de la regió crítica"*.
[2] Gabriel Fiol Roig. *Tema 3: Análisis del coste de los algoritmos.*
[3] Albert Llemosí Cases. *A Primer on Program Construction, IV. Data Structures*