**AO2 PROGRAMACIÓN 1**

**Parte A (Teórico)**

1. Enunciar las diferencias entre un tipo simple y un tipo estructurado de datos en Java.

**Tipo de dato simple:**

-Ocupan una sola celda de memoria

-Se debe declarar una variable diferente para cada dato que se quiera guardar

-Se debe referenciar cada vez que se requiera el uso de esa variable en el programa

-Algunos de ellos son los datos de tipo entero, reales, caracteres, etc.

**Tipo de dato Estructurado:**

-Ocupan un grupo de celdas de memoria

-Se declara una sola variable para guardar un conjunto de datos del mismo tipo

-Optimiza la referenciación de cada conjunto de elementos cada vez que se necesiten utilizarlos a lo largo del programa

-Los más conocido son arreglos, cadena de caracteres, registros, etc.

1. ¿Qué es un array (vector) y cuáles son las características de los mismos?

Un Array es una de las estructuras de datos mas utilizada ya que permite guardar varios datos con solo declarar una variable y cuenta con las siguientes características:

-**Estructura lineal->** guarda datos en una sola dimensión por lo que también se lo denomina vector

-**Homogeneidad->** los datos que guarda como elementos en su interior deben ser del mismo, es decir, deben tener la misma naturaleza para poder considerarlo como un array de caracteres o un array de números enteros

-**Datos indexados->** permite su manipulación ya que a medida que se va ingresando cada dato como parte del array, se le va asignando una índice o posición (nro. entero) de 0 hacia la derecha contando de uno en uno, cuya posición servirá como referencia a lo largo del programa para trabajar con el array correspondiente al que pertenece

-**Acceso Aleatorio->** permite obtener un elemento del array de forma rápida con solo hacer referencia a su posición o índice, en el que se encuentra dicho elemento

-Tamaño Fijo-> no se puede cambiar la cantidad de elementos que componen un array una vez creado

3. Enunciar y explicar los distintos tipos paso de parámetros a métodos en POO en Java.

**Paso de parámetros por valor (por copia):** es cuando java compila el método y el código que llama al método y el método recibe una copia de los valores de los parámetros. Si se quisieran cambiar el valor de uno de ellos, el cambio afecta al método. Cuando se pasa un objeto, lo que se pasa en si es una referencia de este y esa referencia no puede modificarse, pero si los datos del miembro del objeto.

**Lista de parámetros múltiples:** es una secuencia de parámetros separada por comas, con el modificador final si se desea que ningún parámetro modifique su valor o sin él en caso contrario. Si la lista de parámetros está vacía se dice que es de tamaño 0.

**Modificadores de Acceso:** como mencionamos anteriormente, la función del modificador es que si no queremos que el parámetro cambie de valor al momento de declararlos antes del nombre que le coloquemos se deberá escribir la palabra reservada ***final***, esto hace que después de que el parámetro haya sido inicializado no se modifique, si quisiera modificarse dentro del cuerpo del método no se deberá colocar la palabra reservada.

**Sobrecarga de métodos:** se llama así a la definición de dos o más métodos dentro de la misma clase que comparten el mismo nombre pero que las declaraciones de sus parámetros son diferentes, permitiendo así implementar el polimorfismo. Es importante que el llamado a método sobrecargado sea especifico y no ambiguo para que el compilador decida sin equivocarse cuál es el método llamado.

4. ¿Qué tipo de estructura repetitiva se puede utilizar para acceder a los elementos de un array? Justificar su elección.

Para acceder a un array la estructura repetitiva que se suele utilizar es el bucle **for**, ya que generalmente sabemos cuántos elementos contiene, permitiendo así controlar la iteración con un contador que se inicializa, se incrementa e indica una condición de finalización, que coincide con el tamaño del array, además de que nos brinda mayor legibilidad y entendimiento.

5. Nombrar, describir y graficar los procesos de Algoritmos de ordenación y búsqueda básicos.

**Ordenación por Intercambio:** Ordena los elementos de una lista en orden ascendente o descendente comparando el elemento inferior de la lista con los restantes, efectuando el intercambio de posiciones cuando el orden resultante de la comparación no sea el correcto.

[5, 2, 4, 6, 1, 3] (Inicio) Se compara a[0] con a[1] para ver quien es menor, en este caso 2 es menor que 5 se realiza el intercambio.

[2, 5, 4, 6, 1, 3] Se compara a[0] con a[2], no se realiza intercambio.

[2, 5, 4, 6, 1, 3] Se compara a[0] con a[3], no se realiza intercambio.

[2, 5, 4, 6, 1, 3] Se compara a[0] con a[4], se realiza intercambio.

[1, 5, 4, 6, 2, 3] Se compara a[0] con a[5], no se realiza intercambio.

Luego de que el elemento mas pequeño se compara con todos los demás, se localiza en la posición 0 y continua con la siguiente pasada con la sublista restante por ordenar. Cuando el procedimiento termina de comparar con cada elemento y ubica a los números en orden, la lista resultante se verá así:

[1, 5, 4, 6, 2, 3] Lista resultante inicial.

[1, 2, 5, 6, 4, 3] Lista resultante 1er pasada.

[1, 2, 3, 6, 5, 4] Lista resultante 2da pasada

[1, 2, 3, 4, 6, 5] Lista resultante 3era pasada

[1, 2, 3, 4, 5, 6] Lista resultante 4ta pasada, queda ordenado el vector.

**Ordenación por selección:** Ordena arrays de enteros en orden ascendente o descendente buscando el elemento menor o mayor de la lista y lo intercambia con el primer elemento. Luego busca el segundo elemento menor o mayor de la lista y lo cambia por el segundo elemento, y así sucesivamente.

[51, 20, 39, 80, 36] (Inicio) Se selecciona el 20 y se lo coloca en la posición a[0].

[20, 51, 39, 80, 36] (Pasada 1) Se selecciona el 36 y se lo coloca en la posición a[1].

[20, 36, 39, 80, 51] (Pasada 2) Se selecciona el 39 y se lo coloca en la posición a[2].

[20, 36, 39, 80, 51] (Pasada 3) Se selecciona el 51 y se lo coloca en la posición a[3].

[20, 36, 39, 51, 80] Lista ordenada.

**Ordenación por Inserción:** Ordena elementos a medida que se van insertando de menor a mayor o viceversa. Ejemplo: 64,25,12,22,11

[64] Inserta 64 en a[0], la lista esta ordenada.

[25, 64] Inserta 25 en a[0] y 64 se mueve a posición a[1].

[12, 25, 64] Inserta 12 en a[0], se desplaza a la derecha la sublista.

[12, 22, 25, 64] Inserta 22 en a[1], se desplaza a la derecha la sublista.

[11, 12, 22, 25,64] Inserta 11 en a[0], se desplaza hacia la derecha la sublista.

Generalizando, el primer elemento se considera ordenado, porque la lista inicial cuenta con solo un elemento. Por cada iteración se explora la sublista buscando la posición correcta de inserción y al mismo tiempo se mueven una posición hacia la derecha todos los elementos mayores o menores que el elemento a insertar, para dejar libre la posición, e insertarlo.

**Ordenación por Burbuja:** Ordena de pares de elementos, es decir, realiza varias pasadas por el array, en cada una de ellas se comparan pares sucesivos de elementos, si están en orden creciente no cambia de posición, pero si un par está en orden decreciente se intercambian de posición en el mismo.



[25, 60, 45, 35, 13, 92] Lista inicial



[25, 45, 35, 13, 60, 92] Pasada 1, el a[0] se compara con a[1] pero al ser menor, continua en su misma posición, luego se compara a[1] con a[2], al ser mayor que a[2] se realiza intercambio. Se compara a[2] con a[3], se realiza intercambio, a[3] con a[4], se realiza intercambio, se compara a[4] con a[5], no se realiza intercambio.



[25, 35, 13, 45, 60, 92]Pasada 2, se realiza las mismas comparaciones y se realiza intercambio o no en las posiciones desordenadas.

[25, 13, 35, 45, 60, 92]Pasada 3, se realiza las mismas comparaciones y se realiza intercambio o no en las posiciones desordenadas.



[13, 25, 35, 45, 60, 92]Pasada 4, se realizan intercambios, lista ordenada.

[13, 25, 35, 45, 60, 92]Pasada 5, no se realizan intercambios lista ordenada.

Para evitar que se realicen bucles innecesarios, ya que la lista en la pasada 4 ya estaba ordenada, se coloca una variable bandera de tipo interruptor (v o f) que cambia de estado cuando se produce intercambio. Viendo el ejemplo anterior, en la pasada inicial 92 es el elemento mayor ordenado el interruptor = true. En la pasada 1 60 y 92 están ordenados. En la pasada 2 45, 60 y 92 están ordenados, en la pasada 3 35, 45, 60, 92 están ordenados y en la pasada 4 se realiza la única comparación entre 13 y 25, al no producirse intercambios, la lista completa esta ordenada, el interruptor pasa en estado false y la pasada 5 no se ejecuta.

**Búsqueda secuencial:** busca un elemento en una lista utilizando un valor destino llamado clave, el cual será comparado de manera secuencial con cada elemento hasta encontrar el mismo valor que la clave.

[4, 3, 7, 1, 2, 5, 3, 8, 3, 4] Clave = 8

La clave dará como resultado a[7] que es el índice en el que se encuentra el valor buscado. En el caso de tener clave 9, hará la comparación con todos los elementos hasta terminar la lista y dará como resultado -1 porque no se encuentra en el listado.

**Búsqueda binaria:** Si la lista esta ordenada, realizara una búsqueda mejora, consiste en sumar el índice mas alto con el índice mas bajo y dividirlo por dos para buscar en el centro de la lista.

Clave = 51

[20, 36, 39, 51, 80] lista ordenada,

Nuestro central es 39 que se encuentra en la posición a[2], pero como es menor a nuestra búsqueda tenemos que buscar en la sublista derecha

Nuestro central es 51 que se encuentra en la posición a[3] y al mismo tiempo es el número que estamos buscando. En conclusión, la búsqueda binaria en vez de realizar 5 comparaciones como lo haría la búsqueda secuencial, solo realizo 2 comparaciones para encontrar la clave.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Parte B (Práctico)**  **Ejercico\_1** | | | | | | | | |
| PROBLEMA | | | | | | | |
| Realizar un programa que genere números aleatorios en cada ronda y el jugador de esa ronda adivine el número en la menor cantidad de intentos posible, registrando en vectores nombres, intentos, errores y tiempo de cada partida | | | | | | | |
| INPUTS |  | | | | | | |
| Variables | | | | | | | |
| nJugadores | Cantidad de jugadores | | | int | | s/d | |
| numero | nro ingresado por jugador | | | int | | 0 | |
| Método | | | Descripción | | Tipo | | Valor Inicial |
| nombreJugador(String[] nombres) | | | Entrada con JOptionPane | | void | | s/d |
| VECTORES | | |  | | | | |
| [] nombres | | | Guarda nombre jugador | | String | | [nJugadores] |
| [] intentos | | | Guarda intentos en adivinar | | int | | [nJugadores] |
| [] errores | | | Guarda intentos fallidos | | int | | [nJugadores] |
| [] tiempos | | | Guarda los tiempos de partida c/jugador | | long | | [nJugadores] |
| OUTPUTS | | |  | | | | |
| Método | | | Descripción | | Tipo | | Valor Inicial |
| masLento(String[] nombres, long[] tiempos) | | | Devuelve nombre y tiempo en segundos del jugador que más tardó en adivinar | | void | | s/d |
| menorPerformance(String[] nombres, int[] intentos, int[] errores) | | | Devuelve nombre, intentos y errores del jugador que más intentos y errores tuvo | | void | | s/d |
| informe() | | | Muestra Estadísticas puntuales en una sola ventana | | void | | s/d |
| Métodos con retorno | | | | | | | |
| menosIntentos(String[] nombres, int[] intentos) | | | Devuelve nombre y cantidad intentos del jugador que menos intentos hizo | | String | | s/d |
| masRapido(String[] nombres, long[] tiempos) | | | Devuelve nombre y tiempo en segundos del jugador que menos tardó en adivinar | | String | | s/d |
| promErrores(String[] nombres, int[] errores) | | | Devuelve el promedio de errores totales entre todas las partidas | | String | | s/d |
| PROCESO |  | | | | | | |
| Instancia | | | Descripción | | | | |
| nJugadores | | | Ingresa cantidad jugadores=tamaño de cada vector | | | | |
| nombreJugador(String[] nombres) | | | Asigna nombres a cada posición del array | | | | |
| numeroAleatorio | | | Genera número a adivinar con la clase random | | | | |
| numero | | | El jugador ingresa el número a adivinar, mientras no lo adivine se contabilizan intentos y tiempo en segundos, una vez adivinado se asignan los valores como elementos a los arrays correspondientes | | | | |
| String[] opcionNro | | | Guarda las opciones del menú de estadísticas y se muestra en lista desplegable con JOptionpane | | | | |
|  | Case 0 | Muestra ganador (- intentos) | | | | | |
|  | Case 1 | Muestra jugador más rápido (- tiempo) | | | | | |
|  | Case 2 | Muestra jugador más lento (+ tiempo) | | | | | |
|  | Case 3 | Muestra jugador menor performance (+intentos + errores) | | | | | |
|  | Case 4 | Muestra promedio errores (entre todas las partidas) | | | | | |
|  | Case 5 | Muestra Estadísticas ganador, más rápido y promedio errores | | | | | |
|  | Default | Finaliza el menú Estadísticas | | | | | |
| FÓRMULAS | | | | | | | |
| contaIntentos++ | | | | | | | |
| contaErrores=contaIntentos-1 | | | | | | | |
| participantes++ | | | | | | | |
| sumaError+=errores[i]; | | | | | | | |
| promErrores=(sumaError/nombres.length); | | | | | | | |

**Ejercicio\_2 (UML)**

