Programación Orientada a Objetos

Ingeniería Informática

Arreglos

Integrantes:

Gutiérrez Montaño Juan Luis 22010601

Hernández Heredia Kevin 22010603

Grupo:

2a3B

Fecha de entrega:

12/Mayol/2023

Introducción

En la programación orientada a objetos, uno de los elementos fundamentales es el concepto de clase y objeto. Una clase es una plantilla que define un conjunto de atributos y métodos comunes a un grupo de objetos. Mientras que un objeto es una instancia específica de una clase. Los arreglos unidimensionales son un tipo de estructura de datos que puede ser implementada en la programación orientada a objetos como una clase.

Un arreglo unidimensional es una colección ordenada de elementos del mismo tipo de datos, como enteros, cadenas, objetos, entre otros. Cada elemento en el arreglo se almacena en una posición numerada llamada índice. Los arreglos unidimensionales son especialmente útiles cuando se necesita almacenar y acceder a una gran cantidad de datos de manera eficiente.

Por otro lado, los arreglos bidimensionales son una extensión de los arreglos unidimensionales en los que los elementos están dispuestos en filas y columnas formando una tabla o matriz. Al igual que los arreglos unidimensionales, los arreglos bidimensionales también se pueden implementar como clases en la programación orientada a objetos.

En este proyecto, se utilizan arreglos unidimensionales y/o bidimensionales en conjunto con la programación orientada a objetos para crear estructuras de datos eficientes y escalables que permitan trabajar con grandes cantidades de información. Además, la implementación de instancias de clases y objetos permitirá manipular estos arreglos de manera más fácil y flexible.

Competencia especifica

Conoce y aplica programas que implementen el uso de arreglos para reconoceerlos como una herramienta para agrupar datos y facilitar la solucion de problemas .

Marco teórico

Los arreglos unidimensionales y bidimensionales son estructuras de datos importantes en la programación que permiten almacenar y manipular grandes cantidades de información de manera eficiente. En la programación orientada a objetos, los arreglos unidimensionales y bidimensionales pueden ser implementados como clases y objetos, lo que les permite ser utilizados de manera más flexible y escalable.

En la implementación de arreglos unidimensionales como una clase en la programación orientada a objetos, se define una clase que representa el arreglo, y se agregan métodos que permiten realizar operaciones comunes como agregar, eliminar o buscar elementos. También se pueden agregar métodos que permitan ordenar, filtrar o manipular los datos del arreglo de manera más compleja.

En la programación orientada a objetos, un objeto es una instancia de una clase, y cada objeto creado a partir de la clase representa un arreglo unidimensional específico. Cada objeto tiene su propia copia de los atributos de la clase y puede ser manipulado de manera independiente.

Por otro lado, los arreglos bidimensionales pueden ser implementados en la programación orientada a objetos de manera similar a los arreglos unidimensionales, pero en lugar de una sola dimensión, se define una estructura de dos dimensiones para representar la tabla o matriz.

En la implementación de arreglos bidimensionales como una clase en la programación orientada a objetos, se define una clase que representa la tabla o matriz, y se agregan métodos que permiten realizar operaciones comunes como

agregar, eliminar o buscar elementos en las filas y columnas. También se pueden agregar métodos que permitan ordenar, filtrar o manipular los datos del arreglo de manera más compleja.

Al igual que con los arreglos unidimensionales, en la programación orientada a objetos, un objeto es una instancia de una clase, y cada objeto creado a partir de la clase representa una tabla o matriz bidimensional específica. Cada objeto tiene su propia copia de los atributos de la clase y puede ser manipulado de manera independiente

Material y Equipo:

- ✓ Computadora
- ✓ Eclipse IDE for Java Developers – 2023 – 03
- ✓ Java SE Development kit 19.0.2

Desarrollo de la practica

Para el Desarrollo de la practica se tomaron como antecedente de los ejercicios a clases que se encargan del procesado de areglos tanto unidimencionales como bidimencionesles, por ello se emplean clases que tiene caracteristicas de areglos con un enfasis hacia la programacion orientada a objetos. Las principals clases son: Clase array objetos, AlmacenarDatosTabla y AlmacenarDatos, en donde esta ultima ocurren la almacenacion de los diversos metodos para los cuatro ejercicios que se plantean. Para algunos ejercicios -no los principals cuatro- fue necesario el reciclaje una clase anterior ya trabajada con el ligero cambio de que se añadieron nuevo atributos para el calculo y así almacenarlos en esa variable.

Ademas, cabe mencionar que se hizo uso de menus que mostraban la informacion utilizando un switch case para su empleo, por ello se realizaron un total de cuatro menus que peritieron mostrar la informacion.

Clase array objetos

Esta clase se encarga basicamente del procesado de un arreglo unidmencional en el que se crea un metodo para verificar que el arreglo este vacio, otro para saber si aun queda espacio, otro donde se crea el objeto, uno donde se añaden datos al arreglo y por ultimo uno en donde se imprime los datos almacenados en el arreglo.

ArrayObjetos
-Rectangulo datos[] -byte i
+ArrayObjetos() +ArrayObjetos(byte) +boolean ArrayVacio() +boolean hayEspacio() +Rectangulo crearObjetos() +void AñadirObjetos() +String imprimeDatosArray()

AlmacenarDatosTabla

El principal enfoque de esta clase fue la de hacer uso de los areglos de dos dimenciones --bidimencionales-- para el procesado de areglos bidimencionales, sus principals funciones son la inseccion de datos proporcionado para guardarlos en las celdas de las matrices, la imprecion de las mismas y la verificacion de el vacio en el arreglo y el espacio suficiente en el mismo.

Además, tambien se cuenta con algunos metod para hacer una suma de diagonalse, transformar un dato a formato octal, una opcion de matriz constante ya predefinida, matrices de triangulo superior e inferior y por ultimo un método de ordenacion basado en el algoritmo de la burbuja.

AlmacenarDatosTabla
-Object matriz[][] -byte i

-byte j
+AlmacenarDatosTabla(byte m, byte n) +boolean MatrizVacia() +boolean isEspacioMatriz() +void LeerFilas() +String verMatriz() +int SumaDiagonal() +String enOctal() +String MatrizConstante(int valor[][]) +String MatrizTrianguloSuperior() +String MatrizTrianguloInferior() +String ordenaBurbujaMatriz()

AlmacenarDatos

En esta clase se toma como antecedente a las clases anteriores ya que entre ellas se conectarán para formar un caso en los menus, los metodos de esta clase son principalmente los que están en los ejercicios uno a cuatro pero ademas cuenta con un pequeño apartado de procesamiento de datos.

AlmacenarDatos
-Object datos[] -byte i
+AlmacenarDatos(int max) +boolean arrayVacio() +boolean isEspacio() +void AñadirDatos() +String imprimeDatosArray() +String imprimePrimos() +String imprimeBinario() +String imprimeFrecuencia() +String imprimeSuma(int valor) +void ordenaBurbuja()

+String imprimePares()
+String imprimeImpares()
+void datosAleatorios()
+void PersonasAltura()
+void CadenasTexto()
+void Taxista()

ArrayDos

Esta clase representa un arreglo unidimensional de objetos de tipo "IMS". En la clase se encuentran las siguientes funcionalidades:

Constructor: Hay dos constructores definidos para la clase. Uno sin parámetros y otro que toma un parámetro de tipo byte llamado "tam". El segundo constructor inicializa el arreglo "datos" con el tamaño especificado por "tam" y establece el valor de "i" en -1.

Métodos de verificación:

arrayVacio(): Devuelve un valor booleano que indica si el arreglo está vacío o no.

espacioArray(): Devuelve un valor booleano que indica si hay espacio disponible en el arreglo para agregar más elementos.

Método CreaObjeto(): Crea un objeto de tipo "IMS" utilizando la clase "Tools" para leer los valores de diferentes atributos del objeto, como el nombre, la edad, el sexo, el peso y la altura.

Método insertaObjetoLectura(): Inserta un objeto de tipo "IMS" en el arreglo "datos" después de leer los valores correspondientes utilizando el método CreaObjeto(). Si no hay espacio disponible en el arreglo, se muestra un mensaje de error.

Método ordenaBurbuja(): Ordena los elementos del arreglo "datos" utilizando el algoritmo de ordenación de burbuja. El orden se basa en el nombre de cada objeto "IMS", utilizando el método compareTo() de la clase String.

Método buscarDato(): Permite buscar un objeto en el arreglo "datos" utilizando su posición. Se verifica si el arreglo está vacío y se solicita al usuario el número de posición que desea buscar. Si la posición está fuera del rango de datos capturados, se muestra un mensaje de error. De lo contrario, se imprime el objeto encontrado.

Método imprimeDatosArray(): Devuelve una cadena de texto que representa todos los objetos almacenados en el arreglo "datos" y su posición correspondiente.

Método Estadisticas(): Calcula y devuelve una cadena de texto con diferentes estadísticas basadas en los objetos almacenados en el arreglo "datos". Las estadísticas incluyen el número total de personas de género masculino y femenino, así como el número total de personas en diferentes categorías de peso.

En resumen, esta clase proporciona métodos para agregar objetos a un arreglo, ordenar el arreglo, buscar objetos por posición, imprimir los datos del arreglo y calcular estadísticas sobre los objetos almacenados en el arreglo.

ArrayDos
-IMS[] datos - byte i
+ArrayDos() +ArrayDos(byte) +boolean arrayVacio() +boolean espacioArray() +IMS CrearObjeto() +void insertaObjetoLectura() +void buscarDato() +String imprimeDatosArray() +String Estadisticas()

IMS

Esta clase en Java, llamada "IMS", representa un objeto de índice de masa corporal (IMC) con varios atributos y métodos relacionados. Aquí está un resumen de lo que hace esta clase:

Atributos:

numFol: Un atributo estático de tipo short que representa el número de folio asignado a cada objeto IMS. Se inicializa en 1000 y se incrementa cada vez que se crea un nuevo objeto IMS.

nom: Un atributo de tipo String que almacena el nombre de la persona.

edad: Un atributo de tipo byte que almacena la edad de la persona.

sex: Un atributo de tipo char que almacena el sexo de la persona.

peso: Un atributo de tipo float que almacena el peso de la persona en kilogramos.

altura: Un atributo de tipo float que almacena la altura de la persona en centímetros.

imc: Un atributo de tipo double que almacena el índice de masa corporal calculado.

estPeso: Un atributo de tipo String que almacena el estado del peso de la persona basado en el índice de masa corporal.

Constructores:

IMS(): Un constructor vacío sin parámetros.

IMS(String nom, byte edad, char sex, float peso, float altura): Un constructor que toma los valores para los atributos y los asigna a los respectivos campos. También calcula el IMC y determina el estado del peso llamando a los métodos correspondientes.

Métodos de acceso:

Varios métodos de tipo get y set para acceder y modificar los valores de los atributos privados.

Métodos de cálculo y determinación:

calculaIMC(): Calcula el índice de masa corporal dividiendo el peso por la altura al cuadrado (altura/100).

determinaEstadoIMC(): Determina el estado del peso de la persona según el valor del IMC calculado. Retorna una cadena de texto que indica si el peso está en la categoría de bajo peso, peso normal, sobrepeso, obesidad, obesidad severa o obesidad mórbida.

Método toString():

Sobrescribe el método toString() para devolver una cadena de texto que representa todos los atributos del objeto IMS.

En resumen, esta clase "IMS" representa un objeto de índice de masa corporal con atributos para almacenar información relacionada con el nombre, la edad, el sexo, el peso, la altura y el estado del peso de una persona. Además, proporciona métodos para calcular el IMC, determinar el estado del peso y acceder a los valores de los atributos.

IMS
-short numFol -String nom -byte edad -char sex -float peso -float altura -double imc -String estPeso
+IMS() +IMS(String, byte, char, float, float) +short getNumFol() +String getNom() +byte getEdad() +char getSex() +float getPeso() +float getAltura()

+String getEstPeso()
+double getImc()
+void setNom(String)
+void setEdad(byte)
+void setSex(char)
+void setPeso(float)
+void setAltura(float)
+double calculaIMC()
+String determinaEstadoIMC()
+String toString()

Resultados

1) Ejercicio

a) Método booleano que reciba un parámetro y valide si es un numero par

b) Método booleano que reciba un parámetro y valide si es un numero impar

c) Método booleano que reciba un parámetro y valide si es un valor cero.

Posteriormente deberá de crear un métodos de clases donde le pregunte al usuario el tamaño del array unidimensional a usar para almacenar valores aleatorios de tipo byte (Random aleatorio = new Random()), deberá de imprimir en una pantalla emergente los datos almacenados en el array , la media de los números pares, la media de los impares y el total de ceros, deberá de ocupar la invocación de los incisos a,b y c respectivamente.

Primeramente se crea el metodo en donde se hará uso de los metodos de la unidad pasada, los cuales serán llamados como argumento de una cadena de if y else ya que retornan un valor Boolean y en caso de que alguno de los casos llegara a ser verdad entonces el Contador aumenta, el metodo al ser vacio no retorna nada y solo imprime los resultados de los contadores.

```java
public void datosAleatorios(int num) {
    int dat;byte par=0,impar=0,ceros=0;
    for (byte f=0;f<num;f++) {
        dat=(byte)(1+Math.random()*100);
        AñadirDatos(dat);
    }
    for(byte j=0;j<num;j++) {
        if(Metodos.Cero((int)datos[j]))
            ceros+=1;
        else if(Metodos.NumeroPar((int)datos[j]))
            par+=1;
        else if (Metodos.NumeroImpar((int)datos[j]))
            impar+=1;
    }
    Tools.imprimeSalida(imprimeDatosArray()+"\n Numero de pares:"+par+
            "\n Numero de impares:"+impar+"\n Numero de ceros:"+ceros);
}
```

2) Ejercicio

Crear un método de clase donde le pregunte al usuario el tamaño del array unidimensional a usar para almacenar Altura en cm, de N personas leídos por teclado, deberá de calcular la altura media. Calcular cuántas personas tienen una altura superior a la media y cuántas tienen una altura inferior a la media. El valor de N se pide por teclado y debe ser entero un positivo.

El metodo "PersonasAltura" recopila datos de altura en centímetros de un número especificado de personas, calcula la altura media y determina cuántas personas son más altas o más bajas que la altura media, y luego imprime estos datos junto con la lista de alturas ingresadas por el usuario.

```java
public void PersonasAltura(int n) {
    float x,sumAltura=0,media;byte alSup=0,alMen=0;
    if (Metodos.ValidaPosEnt(n)) Tools.salidaError("Solo se permiten enteros positivos");
    else {
        for (byte j=0;j<n;j++) {
            x=Tools.leerFloat("Introduce la altura en centimetros de la persona "+j);
            AñadirDatos(x);
        }
        for (byte k=0;k<n;k++) {
            sumAltura+=(float)datos[k];
        }
        media=sumAltura/n;
        for (byte l=0;l<n;l++) {
            if ((float)datos[l]>media) alSup++;
            else alMen++;
        }
        Tools.imprimeSalida("Los datos capturados son:\n"+imprimeDatosArray()+"\nAltura Media: "+media+
                "\nPersonas mayores a la altura media:"+alSup+"\nPersonas menores a la altura media:"+alMen);
    }
}
```

3) Ejercicio

Crear un método de clase donde le pregunte al usuario el tamaño del array unidimensional a usar para almacenar líneas de texto por teclado, de tamaño, posteriormente recorra el arreglo e imprima cada cadena al revés y cuantos caracteres tiene cada de ellas.

El método "CadenasTexto" solicita al usuario que ingrese un número entero n, y luego solicita al usuario que ingrese n cadenas de texto. Luego, el método invierte cada una de las cadenas de texto ingresadas utilizando el método "CadenaAlRevez" de la clase "Metodos". Además, el método también almacena el tamaño de cada cadena de texto invertida. Finalmente, el método imprime en pantalla una lista de las cadenas de texto originales, seguida de una lista de las cadenas invertidas junto con sus tamaños. Por lo tanto, el método "CadenasTexto" es útil para invertir cadenas de texto y obtener información sobre el tamaño de las mismas.

```
public void CadenasTexto(int n) {
    String x,cad="";
    for (byte j=0;j<n;j++) {
        x=Tools.leerString("Introduce la cadena "+j);
        AñadirDatos(x);
    }
    for (byte k=0;k<n;k++) {
        cad+=Metodos.CadenaAlRevez((String)datos[k])+"\n";
    }
    Tools.imprimeSalida("Datos capturados:\n"+imprimeDatosArray()+"\n\nCadenas invertidas y tamaño de las mismas:\n"+cad);
}
```

4) Ejercicio

Crear un método donde un taxista almacene en un arreglo unidimensional de 30 posiciones el número de carreras diarias que ha realizado en cada uno de los días del mes. El dueño del taxi ha decidido pagarle por cada día trabajado, de la siguiente manera:

a. Si en el día hizo menos de 10 carreras, le pagará a $1500 por carrera.

b. Si en el día hizo entre 11 y 30 carreras, le pagará a $3500 cada carrera.

c. Si en el día hizo más de 30 carreras, cada carrera se la pagará a 5000.

d. Hallar el día en que más carrera hizo

e. Hallar el total de dinero que hizo el taxista en el mes

f. Determinar el promedio de carreras hechas por el taxista.

El método "Taxista" recopila información sobre el número de carreras que realiza un taxista en cada día del mes, encuentra el día con el mayor número de carreras, calcula el dinero ganado por el taxista en el mes y el promedio de carreras realizadas por día, y luego imprime esta información en pantalla

```
public void Taxista() {
    int x,sum=0,carr = 0,dia,diaEsp = 0;
    for (byte j=0;j<30;j++) {
        x=Tools.leerInt("Introduce el numero de carreras del dia "+j);
        AñadirDatos(x);
        sum+=Metodos.PrecioDeCarreras((int)datos[j]);
        carr+=(int)datos[j];
    }
    dia=(int)datos[0];
    for (byte k=1;k<30;k++) {
        if(dia<(int)datos[k]) {
            dia=(int)datos[k];
            diaEsp=k;
        }
    }
    Tools.imprimeSalida("Datos capturados:\n"+imprimeDatosArray()+
            "\nEl dia que mas carreras hizo: "+
            diaEsp+"\nDinero ganado en todo el mes: "+sum
            +"\nPromedio de carreras hechas por dia: "+(carr/30));
}
```

Metodos Bidimencionales

a**)ordenaBurbujaMatriz:**

Este metodo se basa en el algoritmo de ordenamiento de la burbuja pero ahora tiene un cambio para adaptarse al odrdenamiento de espacios en matrices.

```
public String ordenaBurbujaMatrizB(int[][] datos) {
    String cad = "";
    Object ux;
    for (int k = 0; k < datos.length; k++) {
        for (int f = 0; f < datos[k].length - 1; f++)
            if (datos[k][f] > datos[k][f + 1]) {
                ux = datos[k][f];
                datos[k][f] = datos[k][f + 1];
                datos[k][f + 1] = (int) ux;
            }
        }
        for (int i = 0; i < datos[k].length; i++) {
            cad += " " + datos[k][i];
        }
        cad += "\n";
    }
    return cad;
}
```

b) MatrizTrianguloInferior y MatrizTrianguloSuperior

Este método tiene como prinipale objetivo mostrar ciertos datos de la matriz, sin embargo a su vez se divide en dos metodos pues uno se encarga de mostrar la primera mitad -superior- y el segundo la mitad correspondiente -inferior- por lo que la matriz a su vez se divide en dos partes.

```
public String MatrizTrianguloSuperior() {
    String cad="\n";byte t=0,p=0;
    for(byte k=0;k<matriz.length;k++) {
        for (byte l=0;l<matriz[0].length;l++) {
            if (t>p)
                cad+="|  "+matriz[k][l]+"  |";
            else
                cad+="        ";
            t++;
        }
        p++;
        t=0;
        cad+="\n";
    }
    return cad;
}
public String MatrizTrianguloInferior() {
    String cad="";byte t=0,p=0;
    for(byte k=0;k<matriz.length;k++) {
        for (byte l=0;l<matriz[0].length;l++) {
            if (t<p)
                cad+="|  "+matriz[k][l]+"  |";
            t++;
        }
        p++;
        t=0;
        cad+="\n";
    }
    return cad;
}
```

c) MatrizConstante

En este método simplemente se presentan los datos de la matriz que le pasan como argumento.

```
public String MatrizConstante(int valor[][]) {
    String cad="";
    for(byte k=0;k<valor.length;k++) {
        for (byte l=0;l<valor[0].length;l++) {
            cad+=" "+valor[k][l]+" ";
        }
        cad+="\n";
    }
    return cad;
}
```

d) enOctal

Este método recicla otro método ya hecho en el que se transforman los argumentos del segundo método en binario, posteriormente los almacena en una variable de tipo String.

```
  public String enOctal() {
      String cad="Datos almacenados \n";
      for(byte k=0;k< matriz.length;k++) {

          for(byte l=0;l<matriz[0].length;l++) {
              cad+=" ("+Metodos.decimalOctal((int) matriz[k][l])+")       ";
          }
      cad+="\n";

  }return cad;
}
```

e)SumaDiagonal

El metod suma los valores de la matriz cuando su índice es el mismo -lo que sucede cuando hay una diagonal indexada-.

```
public int SummaDiagonal() {
    int suma = 0;
    for(byte k=0;k<matriz.length;k++) {
        for(byte l=0;l<matriz[0].length;l++){
            if(k==l){suma += (int)matriz[k][l];}
        }
    }
    return suma;
}
```

Conclusion:

Los arreglos unidimensionales y bidimensionales son estructuras de datos comunes en la programación, y son muy útiles para almacenar y acceder a datos de manera eficiente. En la programación orientada a objetos, estos arreglos se pueden implementar como parte de una clase para representar un conjunto de datos relacionados y organizar el código de una manera más estructurada.

Por ejemplo, en un programa de contabilidad, se podrían utilizar arreglos unidimensionales para almacenar los ingresos y gastos de cada mes, mientras

que los arreglos bidimensionales se podrían utilizar para almacenar los datos de varios usuarios.

En la programación orientada a objetos, se pueden utilizar diversas técnicas para trabajar con arreglos, como encapsulamiento, herencia y polimorfismo. Por ejemplo, una clase que encapsule un arreglo podría proporcionar métodos para agregar, eliminar y buscar elementos en el arreglo de forma segura y eficiente.

En conclusión, los arreglos unidimensionales y bidimensionales son estructuras de datos importantes en la programación, y su uso en la programación orientada a objetos permite organizar y trabajar con los datos de manera más eficiente y estructurada.

Bibliografía:

KevinDCCsHeOs. (2023, Mayo 12). *KevinDCCsHeOs/Tema5Arrays*. GitHub. https://github.com/KevinDCCsHeOs/Tema5Arrays

Guia de reporte de practicas. (2023). Padlet. https://padlet.com/mtzcastillo2023/tema-3-metodos-7ykd9bx1lgv1xqwr/wish/2538860880

TEMA 5 METODOS. (2023). Padlet. https://padlet.com/mtzcastillo2023/mi-padlet-lujoso-6odgm8fw0bexohp5

en, P. (2023). Programación ATS [YouTube Video]. In *YouTube*. https://www.youtube.com/@ProgramacionATS

de, C. (2023). pildorasinformaticas [YouTube Video]. In *YouTube*. https://www.youtube.com/@pildorasinformaticas