PROGRAMACION II MARKETING DIGITAL

LECTURAS OCTAVA SESION

OPERACIONES

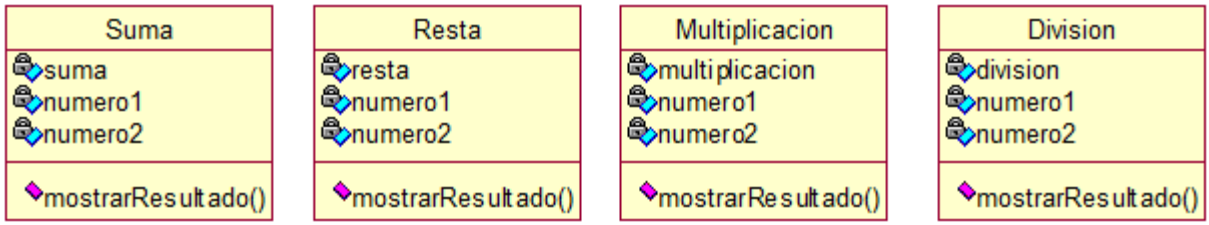
https://www.studocu.com/es/document/universidad-politecnica-de-madrid/programacion/apuntes/tema-3-tipos-y-operaciones-basicas/2440442/view

¿Qué es la herencia de clases?

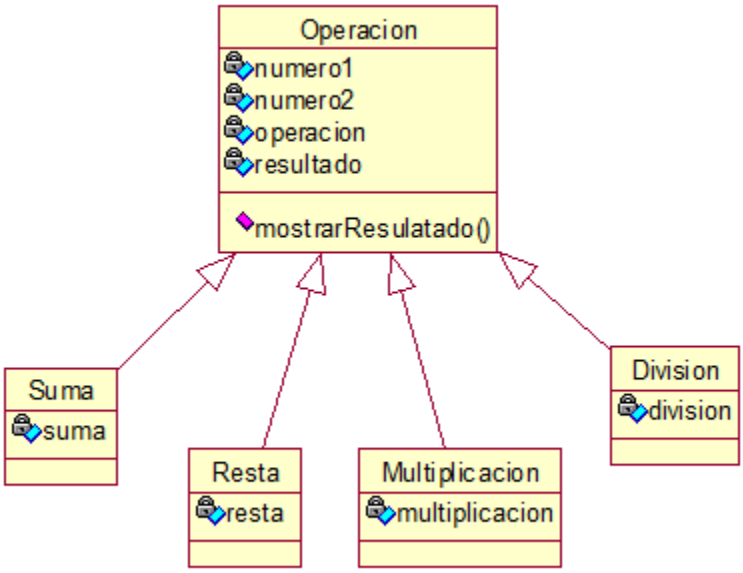
La herencia de clases es la posibilidad de una clase de poder usar los atributos y métodos que ya existen en otra clase distinta, en la Programación Orientada a Objetos (POO) se hace heredando de la clase que contengan los elementos necesarios para tal fin.

La herencia está fuertemente ligada a la reutilización del código en la POO. Esto es, el código de cualquiera de las clases puede ser utilizado sin más que crear una clase derivada de ella, o bien una subclase.

Para poder entender mejor estos conceptos veremos un ejemplo sencillo como lo es el desarrollo de una calculadora con las 4 operaciones básicas (suma, resta, multiplicación y división). Para empezar vamos a definir nuestras clases a usar.



Como podemos observar todas las clases tienen tanto atributos como métodos iguales, esto es perfecto para usar herencia, definimos una nueva clase que contenga todos los atributos y métodos de nuestras 4 clases, a esta nueva clase la llamaremos operación y tendríamos ahora el siguiente esquema:



Como podemos observar ahora tenemos la clase Operación la cual es una superClase porque contiene atributos y métodos que serán heredados por otras clases, y las clases Suma, Resta, Multiplicación y división que son las subClases porque heredan atributos y métodos de la superClase Operacion.

Ahora que ya entendimos lo que es herencia vamos a implementarla, para este tutorial vamos a trabajar con el lenguaje de programación Java y el entorno de desarrollo NetBeans.

Primero implementaremos la superClase Operacion que es la que contiene los atributos y métodos a heredar.

```
public class Operacion {

    double n1;
    double n2;
    double res;
    char operacion;



    public Operacion(double n1, double n2, char operacion) {

        this.n1 = n1;
        this.n2 = n2;
        this.operacion = operacion;
```

```
    }



    public void mostrarResultado(){

        System.out.println(this.n1+" "+this.operacion+" "+this.n2+" =
"+this.res);

    }
}
```

La herencia en Java (y en otros lenguajes de programación) se declara con la palabra reservada extends seguida de la clase de la cual deseamos heredar.

Para la clase Suma quedaría así:

```
public class Suma extends Operacion{

    double suma;

    public Suma(double n1, double n2) {

        super(n1, n2, '+');
        this.suma = n1 + n2;
        this.setRes(this.suma);
    }
}
```

Resta:

```
public class Resta extends Operacion{

    double resta;

    public Resta(double n1, double n2) {

        super(n1, n2, '-');
        this.resta = n1 - n2;
        this.setRes(this.resta);
    }
```

```
}
```

Multiplicación:

```
public class Multiplicacion extends Operacion{

    double multi;

    public Multiplicacion(double n1, double n2) {

        super(n1, n2, '*');
        this.multi = n1 * n2;
        this.setRes(this.multi);
    }
}
```

y por último División:

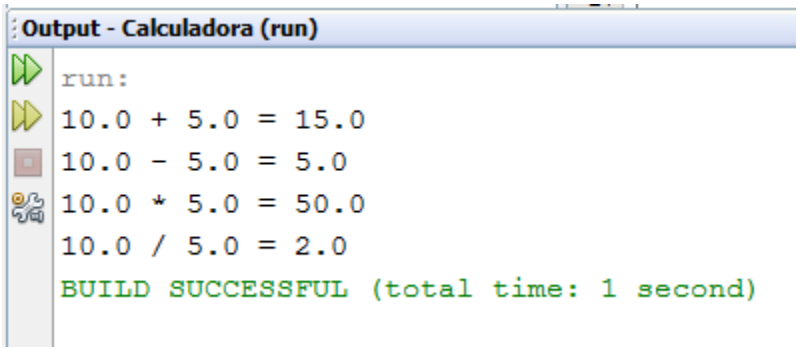
```
public class Division extends Operacion{

    double div = 0;

    public Division(double n1, double n2) {

        super(n1, n2, '/');
        if(n2==0) System.out.println("No se puede dividir entre cero");
        else this.div = n1 / n2;
        this.setRes(this.div);
    }
}
```

Como podemos apreciar en las cuatro clases utilizamos la palabra reservada extends y seguida la clase de la cual se está heredando en nuestro caso Operacion, esto permite tener acceso a los métodos y atributos de esta clase, además hacemos uso del método super que hace una instancia de la superClase Operacion enviando como parámetros los números con los cuales se realizará la operación y el símbolo de esta, luego ejecutamos la operación y enviamos el resultado a la superClase.

En nuestro programa principal instanciamos a cada una de las clases para realizar las operaciones y hacemos uso del método de la superClase Operacion para mostrar los resultados.

```
double n1 = 10;
double n2 = 5;

//suma
Suma sum = new Suma(n1,n2);
sum.mostrarResultado();

//resta
Resta res = new Resta(n1,n2);
res.mostrarResultado();

//multiplicacion
Multiplicacion mul = new Multiplicacion(n1,n2);
mul.mostrarResultado();

//division
Division div = new Division(n1,n2);
div.mostrarResultado();
```

Como resultado obtenemos el resultado de las operaciones y las podemos observar en la consola de salida de NetBeans, así que la herencia funciona correctamente.



También puedes aprender esto y mucho más en el curso de fundamentos de Java que DevCode ofrece para ti: https://devcode.la/cursos/fundamentos-java/

TIPOS DE ACCESO

Normalmente, cuando se codifica un programa, se hace con la intención de que ese programa pueda interactuar con los usuarios del mismo, es decir, que el usuario pueda pedirle que realice cosas y pueda suministrarle datos con los que se quiere que haga algo. Una vez introducidos los datos y las órdenes, se espera que el programa manipule de alguna forma esos datos para proporcionarnos una respuesta a lo solicitado.
Además, en muchas ocasiones interesa que el programa guarde los datos que se le han introducido, de forma que si el programa termina los datos no se pierdan y puedan ser recuperados en una sesión posterior. La forma más normal de hacer esto es mediante la utilización de ficheros que se guardarán en un dispositivo de memoria no volátil (normalmente un disco).
A todas estas operaciones, que constituyen un flujo de información del programa con el exterior, se les conoce como *Entrada/Salida* (*E/S).*
Existen dos tipos de E/S; la *E/S estándar* que se realiza con el terminal del usuario y la *E/S a través de fichero*, en la que se trabaja con ficheros de disco.
Todas las operaciones de E/S en Java vienen proporcionadas por el paquete estándar de la API de Java denominado *java.io* que incorpora interfaces, clases y excepciones para acceder a todo tipo de ficheros. En este tutorial sólo se van a dar algunas pinceladas de la potencia de este paquete.

Entrada/Salida estándar

Aquí sólo trataremos la entrada/salida que se comunica con el usuario a través de la pantalla o de la ventana del terminal.
Si creamos una *applet* no se utilizarán normalmente estas funciones, ya que su resultado se mostrará en la ventana del terminal y no en la ventana de la *applet.* La ventana de la *applet* es una ventana gráfica y para poder realizar una entrada o salida a través de ella será necesario utilizar el AWT.
El acceso a la entrada y salida estándar es controlado por tres objetos que se crean automáticamente al iniciar la aplicación: *System.in*, *System.out* y *System.err*

a.) System.in

Este objeto implementa la entrada estándar (normalmente el teclado). Los métodos que nos proporciona para controlar la entrada son:

- *read():* Devuelve el carácter que se ha introducido por el teclado leyéndolo del buffer de entrada y lo elimina del buffer para que en la siguiente lectura sea leído el siguiente carácter. Si no se ha introducido ningún carácter por el teclado devuelve el valor -1.
- *skip(n):* Ignora los *n* caracteres siguientes de la entrada.

b.) System.out

Este objeto implementa la salida estándar. Los métodos que nos proporciona para controlar la salida son:

- *print(a):* Imprime *a* en la salida, donde *a* puede ser cualquier tipo básico Java ya que Java hace su conversión automática a cadena.
- *println(a):* Es idéntico a *print(a)* salvo que con *println()* se imprime un salto de línea al final de la impresión de *a*.

c.) System.err

Este objeto implementa la salida en caso de error. Normalmente esta salida es la pantalla o la ventana del terminal como con *System.out*, pero puede ser interesante redirigirlo, por ejemplo hacia un fichero, para diferenciar claramente ambos tipos de salidas.
Las funciones que ofrece este objeto son idénticas a las proporcionadas por *System.out*.

Ejemplo
A continuación vemos un ejemplo del uso de estas funciones que acepta texto hasta que se pulsa el retorno de carro e informa del número de caracteres introducidos.

```
import java.io.*;
class CuentaCaracteres {
public static void main(String args[]) throws IOException {
int contador=0;
while(System.in.read()!='\n')
contador++;
System.out.println(); // Retorno de carro "gratuito"
System.out.println("Tecleados "+contador+" caracteres.");
}
}
```

Entrada/Salida por fichero

Tipos de ficheros
En Java es posible utilizar dos tipos de ficheros (de texto o binarios) y dos tipos de acceso a los ficheros (secuencial o aleatorio).
Los ficheros de texto están compuestos de caracteres legibles, mientras que los binarios pueden almacenar cualquier tipo de datos (*int*, *float*, *boolean*,...).
Una lectura secuencial implica tener que acceder a un elemento antes de acceder al siguiente, es decir, de una manera lineal (sin saltos). Sin embargo los ficheros de acceso aleatorio permiten acceder a sus datos de una forma aleatoria, esto es indicando una determinada posición desde la que leer/escribir.

Clases a estudiar
En el paquete *java.io* existen varias clases de las cuales podemos crear instancias de clases para tratar todo tipo de ficheros.
En este tutorial sólo vamos a trata las tres principales:

- *FileOutputStream*: Fichero de salida de texto. Representa ficheros de texto para escritura a los que se accede de forma secuencial.
- *FileInputStream*: Fichero de entrada de texto. Representa ficheros de texto de sólo lectura a los que se accede de forma secuencial.
- *RandomAccessFile*: Fichero de entrada o salida binario con acceso aleatorio. Es la base para crear los objetos de tipo fichero de acceso aleatorio. Estos ficheros permiten multitud de operaciones; saltar hacia delante y hacia atrás para leer la información que necesitemos en cada momento, e incluso leer o escribir partes del fichero sin necesidad de cerrarlo y volverlo a abrir en un modo distinto.

Generalidades

Para tratar con un fichero siempre hay que actuar de la misma manera:

1. Se abre el fichero. Para ello hay que crear un objeto de la clase correspondiente al tipo de fichero que vamos a manejar, y el tipo de acceso que vamos a utilizar:

TipoDeFichero obj = new TipoDeFichero(ruta);

Donde *ruta* es la ruta de disco en que se encuentra el fichero o un descriptor de fichero válido. Este formato es válido, excepto para los objetos de la clase *RandomAccessFile* (acceso aleatorio), para los que se ha de instanciar de la siguiente forma:

RandomAccessFile obj = new RandomAccessFile(ruta, modo);

Donde *modo* es una cadena de texto que indica el modo en que se desea abrir el fichero; "r" para sólo lectura o "rw" para lectura y escritura.

2. Se utiliza el fichero. Para ello cada clase presenta diferentes métodos de acceso para escribir o leer en el fichero.

3. Gestión de excepciones (opcional, pero recomendada) Se puede observar que todos los métodos que utilicen clases de este paquete deben tener en su definición una cláusula *throws IOException*. Los métodos de estas clases pueden lanzar excepciones de esta clase (o sus hijas) en el transcurso de su ejecución, y dichas excepciones deben de ser capturadas y debidamente gestionadas para evitar problemas.

4. Se cierra el fichero y se destruye el objeto. Para cerrar un fichero lo que hay que hacer es destruir el objeto. Esto se puede realizar de dos formas, dejando que sea el recolector de basura de Java el que lo destruya cuando no lo necesite (no se recomienda) o destruyendo el objeto explícitamente mediante el uso del procedimiento c*lose()* del objeto:

obj.close()