

Guía

de Programación Java

Programación
para todos.

Julian Stiven Silva Muñetones

Contenido

Introducción.....	6
Agradecimientos:.....	7
¿Qué es Java?	9
¿Para qué sirve?	9
Características y ventajas.....	9
• Es un lenguaje simple	9
• Orientado a objetos	9
• Su compilación	10
Usos más comunes:	10
• Productividad y utilidades.....	10
• Entretenimiento	10
• Educación	10
• Comunicación	10
• Aplicativos móviles	10
Instalación de IDE para java ECLIPSE	11
¿Qué es eclipse?.....	11
Package y librerías.....	21
¿Qué son?	21
¿Cómo se crea un Package en java?.....	22
Importación de las librerías y paquetes	23
JAVA.UTIL.....	23
Collections	24
JAVA.IO	25

Streams	25
ObjectInputStream y ObjectOutputStream.....	26
FileInputStream y FileOutputStream.....	28
DataInputStream y DataOutputStream	29
Tipos de lecturas e impresión.....	31
¿Qué es lectura de datos?.....	31
Entrada por teclado:	31
Clase de lectura Scanner.....	31
Clase de lectura BufferedReader e InputStreamReader	32
Clase Console	33
Ventajas	33
Desventajas	33
¿Qué es salida de datos?	33
Clase System (salida de datos)	34
• Impresión print:	34
• Impresión Println:	34
• Impresión Printf:	35
Clase PrintWriter.....	37
Clases tipo envoltorio o wrapper	38
¿Cómo es el proceso para hacer paso a un envoltorio?	38
Boxing:.....	39
Unboxing:.....	39
Clase Integer	41
Clase Character:.....	42
Clase tipo String:	45
Constantes	48

¿Qué son las constantes?	48
Operadores Aritméticos.....	49
Operadores de incremento ++ y decremento --	51
Clase Math.....	51
Ciclos y Condicionales	54
Sentencia If-Else-Elseif.....	54
If:	54
Sentencia else.....	55
Anidación if.....	55
Sentencia Switch.....	56
Ciclo while.....	58
Ciclo Do-While	59
Ciclo for.....	60
Ciclo forEach	61
Funciones y métodos	63
Métodos	63
Funciones	63
Procedimientos	63
Estructuras de datos	65
¿Qué es una estructura de datos?	66
Arrays:	66
Length:	68
ArrayList, lista o vector tamaño dinámico:.....	68
Vector Sets o vector sin repeticiones:	70
Matrices:.....	71
Diccionarios o mapa (hashmap, treemap o map)	72

Cola:	73
Stack o Pila:.....	75
Algoritmos de ordenamiento.....	79
Ordenamiento burbuja o Bubble Sort	79
Quick Sort	81
Algoritmos de búsqueda.....	82
Binary Search	83
Exponential Search.....	84
Programación Orientada a Objetos (POO).....	85
Creación de clase y objetos.....	87
Métodos y parámetros	89
Constructores	91
Herencias.....	92
Polimorfismo	96
Programación con interfaces gráficas	103
¿Qué es?	103
¿Cómo crear una interfaz?	103
¿Qué se entiende como clase contenedor?	105

Guía de aprendizaje teórico-práctica de programación para estudiantes de ingeniería

Java

Introducción

La finalidad de esta guía será dar a conocer el lenguaje, todos los IDE (Entorno de Desarrollo Integrado) donde se puede programar, también se enfocará en una guía básica para el inicio en la programación competitiva para todo aquel que quiera aprender, pero primero debemos responder la pregunta del **¿Por qué programar con java y no con otros lenguajes?**, pues bien, Java es un lenguaje de programación muy grande y muy utilizado para distintos fines, entre los más destacados que se puede observar los aplicativos y páginas web, donde si los pluggins de java no se encuentran instalados no funcionan; de la misma manera para la programación competitiva, en la cual se ha visto un crecimiento exponencial de personas que se interesan más sobre el tema; así mismo, para las personas que quieren incursionar en la programación competitiva existen ahora muchas competencias tanto a nivel nacional como internacional, donde podemos encontrar *la Colombian Collegiate Programming League (CCPL)*, la cual es la liga de programación competitiva en Colombia donde compiten todas las universidades a nivel nacional para demostrar quién es el mejor y así poder identificar una que otra falencia, buscando siempre la perfección; por otro lado, también se encuentra *la International Collegiate Programming Contest (ICPC)*, esta es la competencia internacional más grande, consta de 3 fases donde se ubica la maratón nacional de programación en la cual compiten todas las universidades del país donde esté presente la programación competitiva para escoger a los mejores y representar a su país en la segunda fase que sería la

maratón regional de programación, en ella compiten los mejores programadores de los países que conforman cada continente escogiendo a los más destacados para llevarlos a la tercera fase la cual sería la maratón internacional de programación; en donde, compiten todos los países del mundo que tiene programación competitiva, para saber quién es el equipo y país óptimo en la programación competitiva, recibiendo incentivos no solamente económicos sino también el reconocimiento por toda la comunidad de programación competitiva del mundo.

Además de la programación competitiva, java es uno de los lenguajes más utilizados para la programación orientada a objetos (POO), esta consta de la organización de códigos en clases, en donde en cada una de ellas va una parte del código que luego se relacionará para conseguir las metas del aplicativo a crear.

De acuerdo con lo dicho anteriormente la programación orientada a objetos (POO) se entiende como la forma en la que las personas se expresan en la vida real mediante la programación y en la creación de ordenes computacionales buscando la satisfacción de necesidades vistas en el día a día mediante la utilización de los conocimientos de la implementación de clases, objetos, métodos y demás cosas que implica la POO.

Agradecimientos:

Este libro principalmente es dedicado para mis familiares, seres queridos, amigos, docentes y personas que me apoyan y guían día a día en mi formación como tecnólogo e ingeniero, a estas personas les doy mis más sinceras gracias por todo el apoyo brindado, por los regaños y gritos que me han dado con el fin de que haya podido llegar a este punto, por parte de mi familia le agradezco a mi abuelita Magdalena bustos ya que gracias a ella es que me motive en estudiar y obtener los logros que he obtenido, siempre cuido y me crio y me hizo la persona que soy el día de hoy, y ahora desde el cielo me cuida y me da mucha fuerza para seguir esforzándome por ser el mejor, a mi abuelito Heli Muñetones, la guía y formación de un padre, la persona que me ha corregido y me ha brindado apoyo moral, financiero y sobre todo el amor con el que me trata día a

día, a mi madre Jackeline Muñetones por todo el esfuerzo que ha hecho por poderme sacar a delante si el apoyo de un esposo o una pareja sentimental, gracias a ella he podido llegar a este punto y obtener estos resultados, a mi hermano Rubén Silva, un ejemplo a seguir siendo un apoyo y guía en algunas materias en las cuales pueda que haya presentado dificultades, a mi tío Oscar Heli Muñetones la persona que más me ha brindado su cariño, apoyo y brindándome todo su amor como tío, amigo, y figura de padre, a mi tía Esperanza Muñetones, una tercera madre que me crio en los momentos que mi madre se encontraba trabajando, a Luis Hernando uní, mi primo amigo y apoyo que he tenido desde que yo estaba pequeño, de mis amigos a mi equipo de maratones de programación que son los cuales con los que estoy realizando este gran proyecto, Edwin Andrés Villarraga y Cesar Daniel Alayon, son los mejores amigos, compañeros de clase y de trabajos con los que me he podido cruzar, A Diego Fernando Rodríguez, una gran persona, un gran programador y un gran maestro, gracias a el este proyecto se pudo llevar a cabo y culminarse con éxito, a Ángela Sofía Alvarado Ramírez mi mejor amiga, consejera y apoyo que he tenido, le doy gracias por la ayuda, paciencia y colaboración que he tenido al momento de realizar este proyecto y en mi día a día, y en general a todos mis amigos más cercanos les doy gracias, a los docentes de la universidad ECCI, ya que sin el conocimiento de ellos, la paciencia y la dedicación no hubiera podido plasmar mis ideas, conocimientos para que esto fuese posible.

¿Qué es Java?

Java es un lenguaje de alto nivel que fue creado y comercializado en el año 1995 por Sun Microsystem luego comprada por Oracle, la necesidad de crear este lenguaje es buscar una forma más eficaz de programar, más completa, con más opciones y sobre todo con una estructura fácil y sencilla de entender.

¿Para qué sirve?

Este lenguaje al ser multiplataforma y muy completo para la programación orientada a objetos (POO), sirve para crear inmensidad de aplicativos para cualquier tipo de dispositivo y/o máquina que requiera una programación extensa completa y fácil de entender.

Java está muy presente en la programación de aplicativos móviles, de escritorio y de algunos electrodomésticos, así como en algunos aplicativos web con código HTML.

Características y ventajas

- **Es un lenguaje simple:** este lenguaje de programación no es para nada complejo ya que la curva de aprendizaje es corta y de fácil familiarización ya que la mayoría de términos se usan en otros lenguajes de programación como lo es c o c++.
- **Orientado a objetos:** como en si ningún lenguaje de programación a llegado al punto que realmente se quiere dar a entender como programación orientada a objetos, java es el que se aproxima mayormente y es uno de los más completos en esta función, en este caso en java los objetos se encargan de encapsular información, clases y funciones, que se pueden modificar, usar y llamar dentro y fuera de ellas, como también relacionarlo con otros programas.

- **Su compilación:** la compilación es tan buena, que se llega a asimilar al lenguaje ensamblador, es decir, desde la base puede ser interpretado. Esto ayuda muchísimo a la ejecución de aplicaciones compiladas en Java, pues se puede ejecutar básicamente en cualquier lugar sin mayor problema.

Usos más comunes:

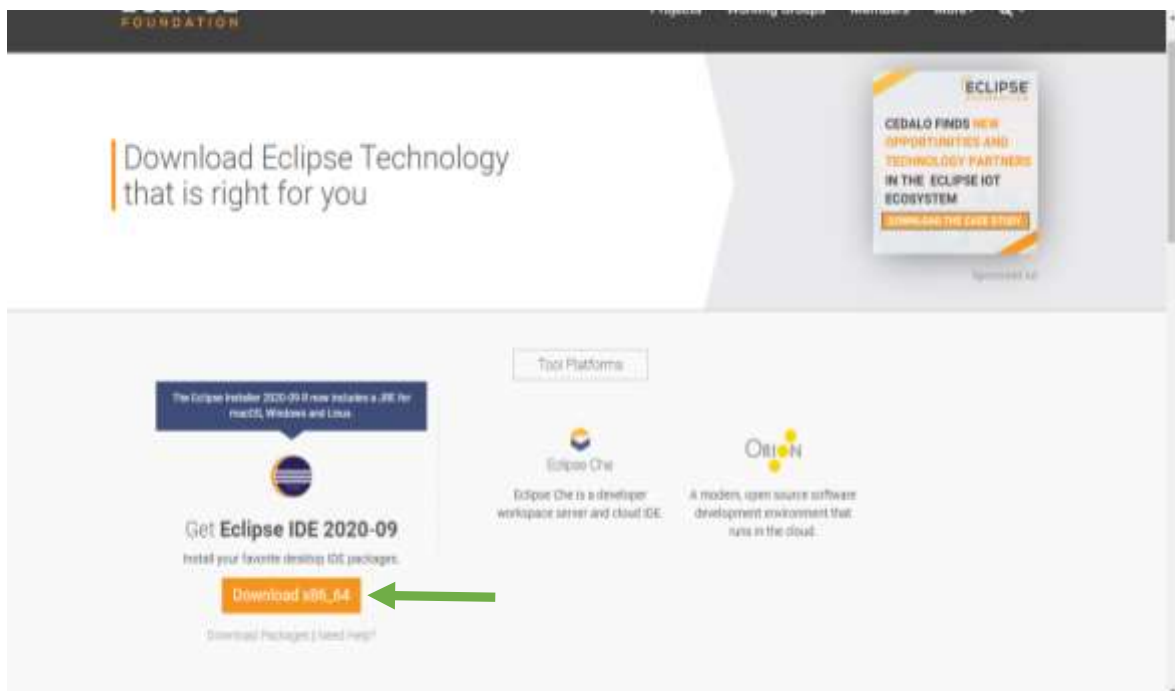
- **Productividad y utilidades:** java se usa para aplicaciones de todo tipo, como hojas de cálculo, juegos, reproductores de música, etc.
- **Entretenimiento:** en la parte de entretenimiento los juegos sobre salen en la programación con java, ya que al ser un lenguaje multiplataforma se puede programar para jugar en consolas, computadores, móviles y en general todo dispositivo que se permita jugar.
- **Educación:** en la parte educativa java es uno de los mejores lenguajes para aprender, así como en aplicativos de estudio en general, como aplicativos para hacer gráficas, para hacer cálculos, etc.
- **Comunicación:** en la parte de comunicación se encuentran los applets de java con los cuales se dio paso a la creación de aplicaciones de mensajería instantánea más importante como lo es WhatsApp, telegram, viber, line, etc.
- **Aplicativos móviles:** en la parte móvil se utilizan muchos los applets ya que son micro versiones de códigos que se puede compilar y ejecutar más fácil y rápido en estos dispositivos para que sean rápidos y eficaces.

Instalación de IDE para java ECLIPSE

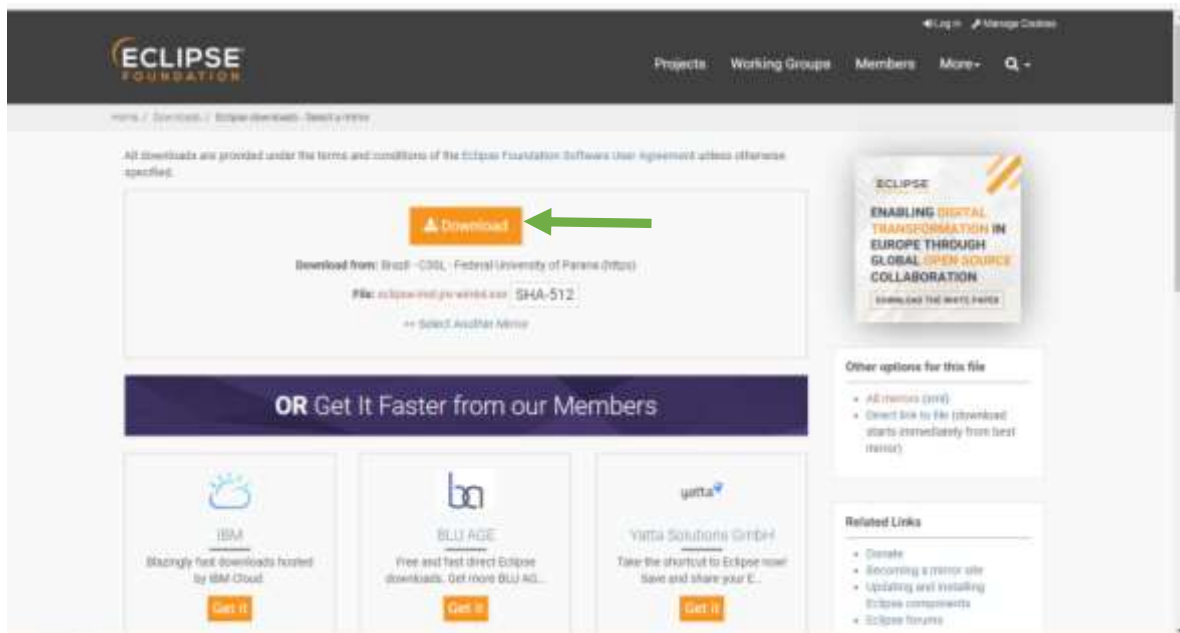
¿Qué es eclipse?

Eclipse es un entorno de desarrollo integrado, es uno de los más utilizados hoy en día para el desarrollo de software, se tiene la opción de descargar eclipse para desarrolladores de Java, C++, Python, JavaScript, eso hace que sea uno de los IDE más completos al igual que NetBeans con la pequeña diferencia de que su instalación es mucho más sencilla.

Para la instalación tenemos que tener en cuenta que versión de java vamos a usar, el presente se explicara y se enseñara en java 8 ya que contiene todo lo que se mostrara a continuación, en cambio en el java 11 ya no es necesario el JRE si no lamente el JDK para su correcto funcionamiento, y para empezar debemos entrar a la página <https://eclipse.org/downloads/eclipse-packages/>.



Tomado de: Propio



Tomado de: Propio

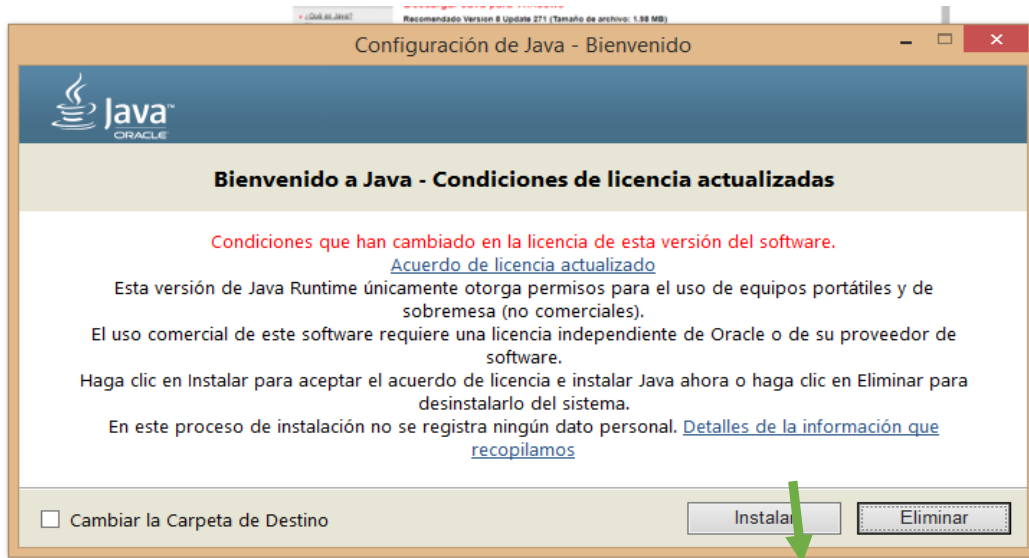
Antes de ejecutar el instalador debemos instalar el JRE y el JDK de java para que pueda funcionar correctamente, para ello debemos instalar de primeras el JRE y luego el JDK de la siguiente manera.

Entramos al siguiente link: www.java.com/es/download/ie_manual.jsp



Tomado de: Propio

Luego ejecutamos el instalador que se descargó, le damos sí para que se inicie y le damos instalar apenas se abra.



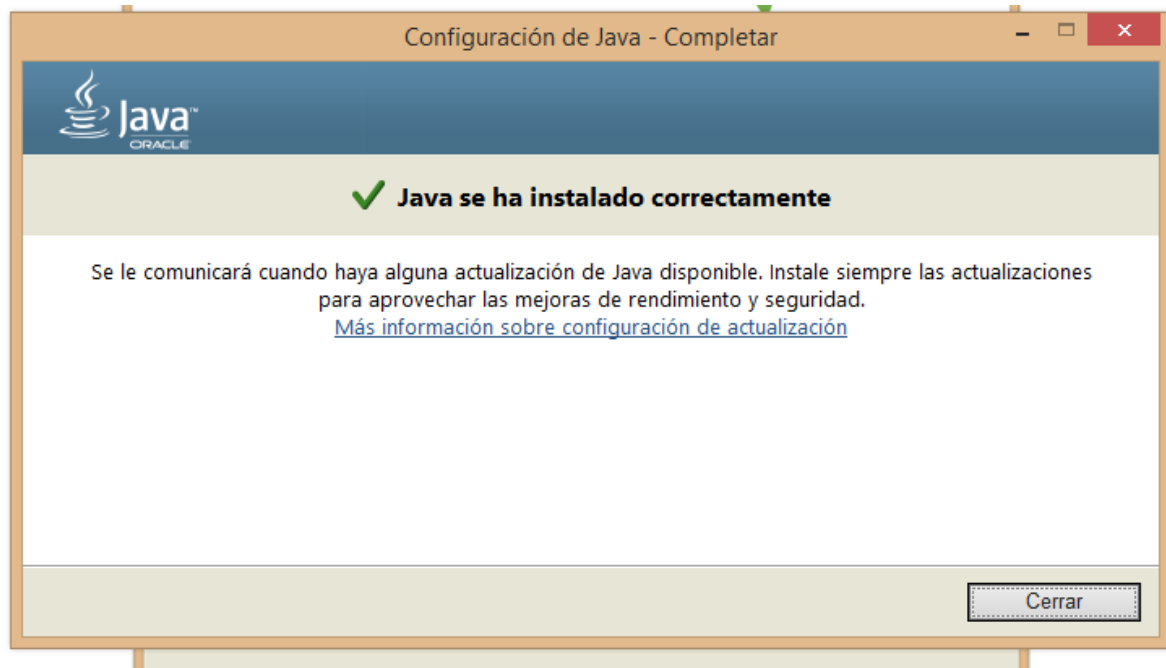
Tomado de: Propio

Después esperamos que termine de instalar.



Tomado de: Propio

Luego que termine de damos cerrar y queda instalado.



Tomado de: Propio

Luego seguimos con la instalación del JDK y lo que se tiene que hacer es entrar en la siguiente página:

www.oracle.com/technetwork/es/java/javase/downloads/index.html.

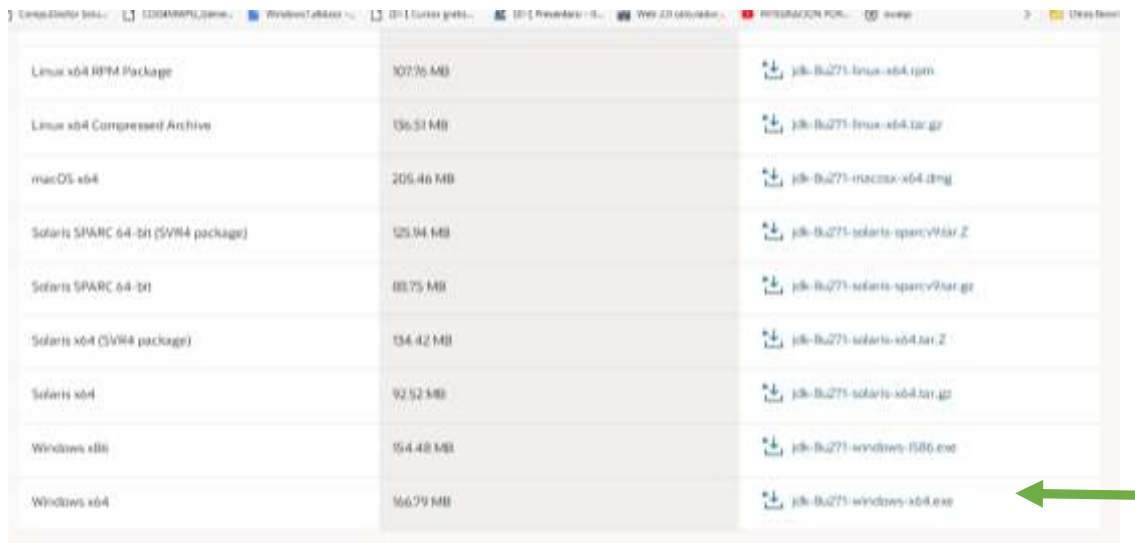
Después le damos en download en el JDK.



Tomado de: Propio

Luego de eso se nos abrirá una página con todas las opciones a descargar, lo que debemos hacer es descargar la de nuestro sistema operativo en este caso

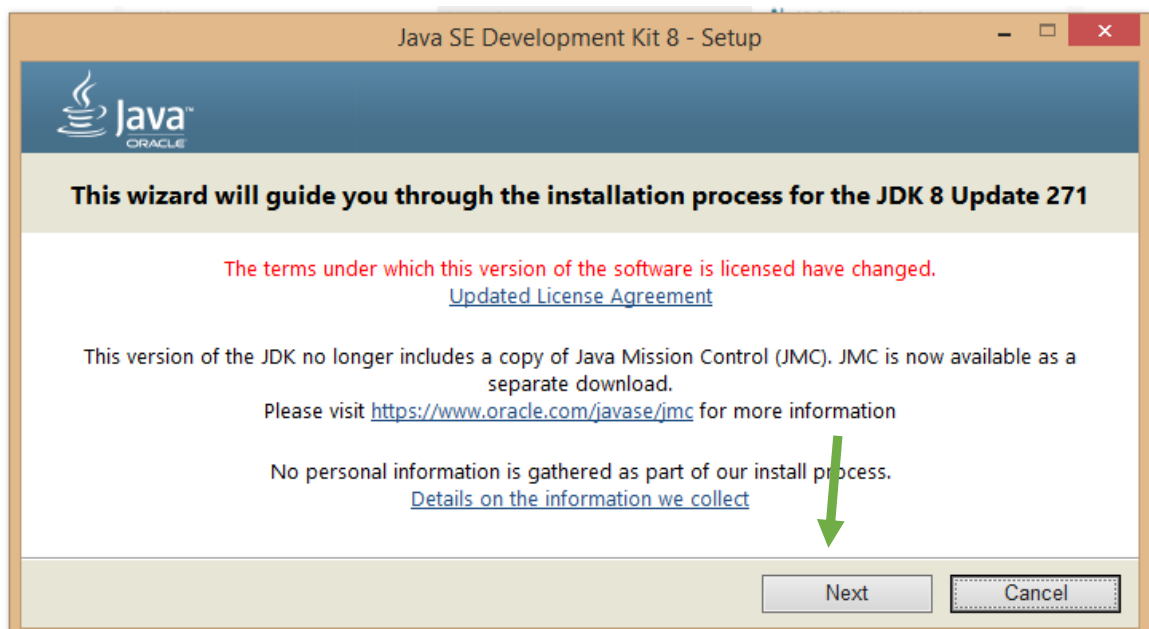
es Windows de 64 bits o x64, si fuera de 32 bits sería el de x86 (es importante saber que arquitectura tiene tu sistema operativo para descargarlo correctamente).



Linux x64 RPM Package	307.76 MB	jdk-8u271-linux-x64.rpm
Linux x64 Compressed Archive	136.31 MB	jdk-8u271-linux-x64.tar.gz
macOS x64	205.46 MB	jdk-8u271-macosx-x64.pkg
Solaris SPARC 64-bit (SVN4 package)	125.94 MB	jdk-8u271-solaris-sparcv9tar.Z
Solaris SPARC 64-bit	88.75 MB	jdk-8u271-solaris-sparcv9tar.gz
Solaris x64 (SVN4 package)	134.42 MB	jdk-8u271-solaris-x64tar.Z
Solaris x64	92.52 MB	jdk-8u271-solaris-x64tar.gz
Windows x86	154.48 MB	jdk-8u271-windows-i586.exe
Windows x64	166.79 MB	jdk-8u271-windows-x64.exe

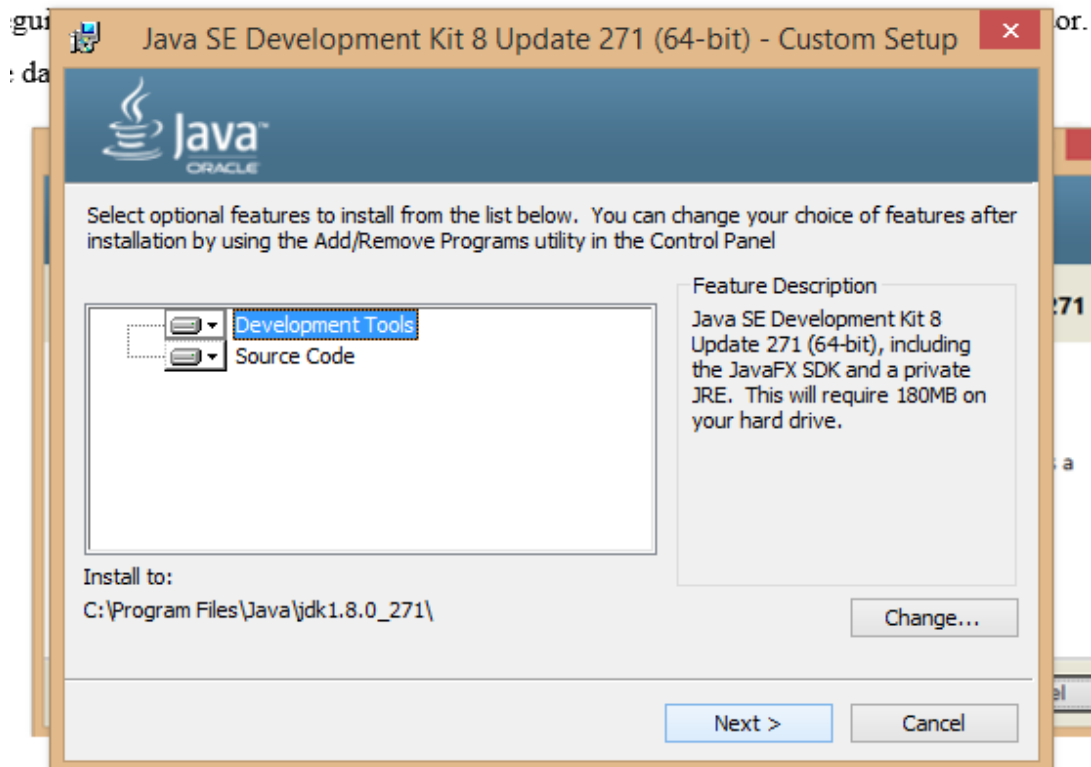
Tomado de: Propio

Seguimos todos los pasos que se nos pide y le damos descargar, y abrimos el instalador. Le damos si y se nos abrirá la aplicación.



Tomado de: Propio

Le damos next para que empiece la instalación.



Tomado de: Propio

Le damos next sin moverle nada y empezara la instalación.



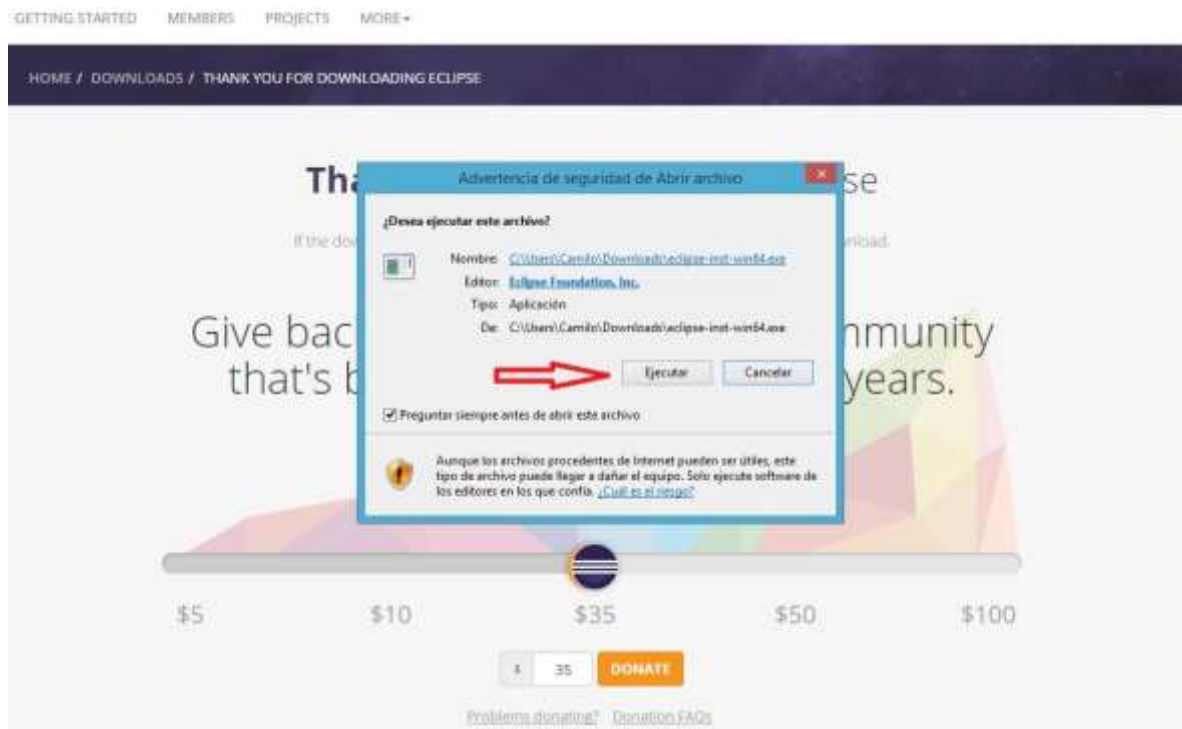
Tomado de: Propio

Luego de que cargue ya queda instalado y podremos continuar con la instalación del eclipse.



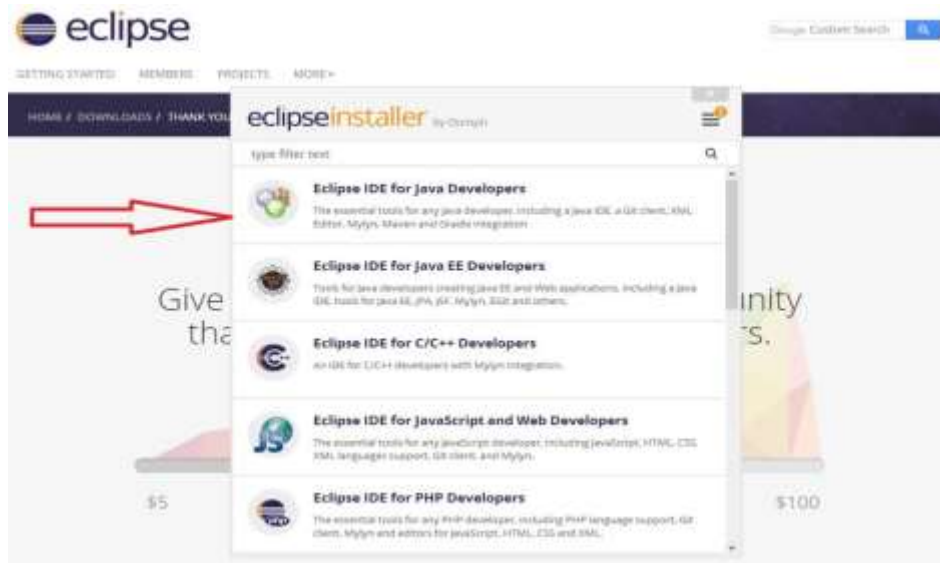
Tomado de: Propio

Después de que este instalado el JRE y el JDK ya podemos abrir el instalador y le damos en ejecutar para que entre a la aplicación



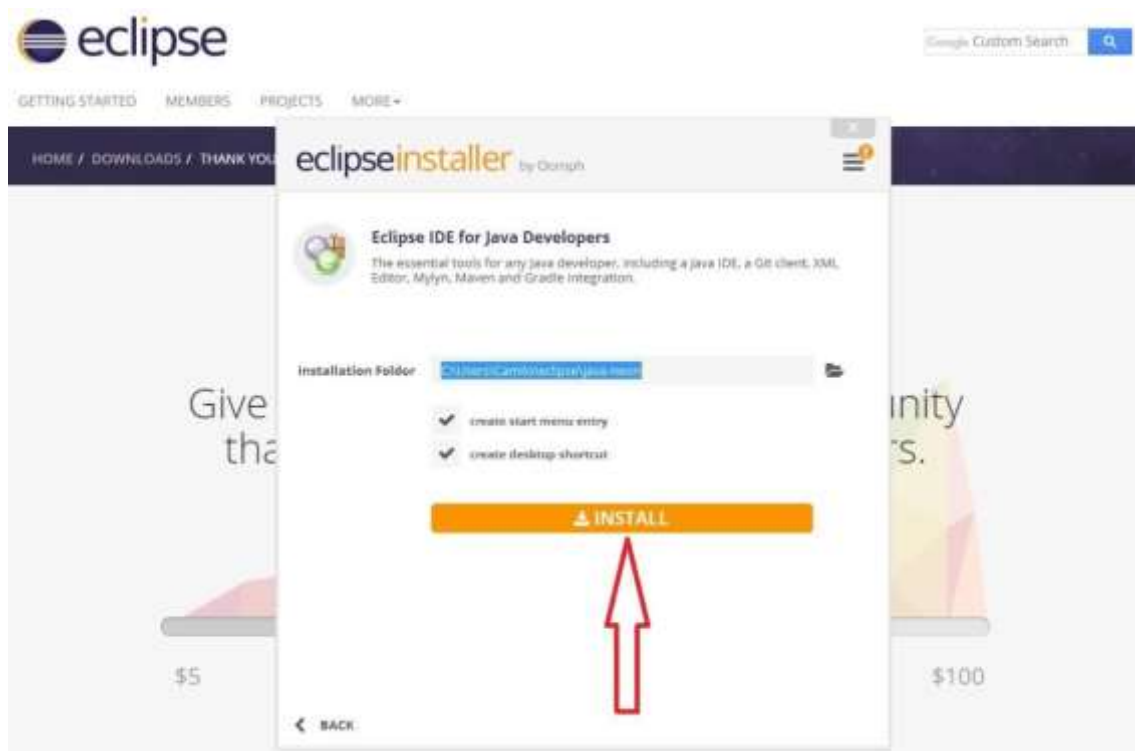
Tomado de: www.tutobasico.com/instalar-eclipse-windows/

Al momento de abrirse la aplicación se abrirán las opciones para los distintos lenguajes, nosotros le daremos a **Eclipse IDE for Java Developers**.



Tomado de: www.tutobasico.com/instalar-eclipse-windows/

Al darle clic el propio instalador señalará la carpeta en donde se instalará y guardará todos sus datos, ahí le damos install para que comience la instalación del programa.



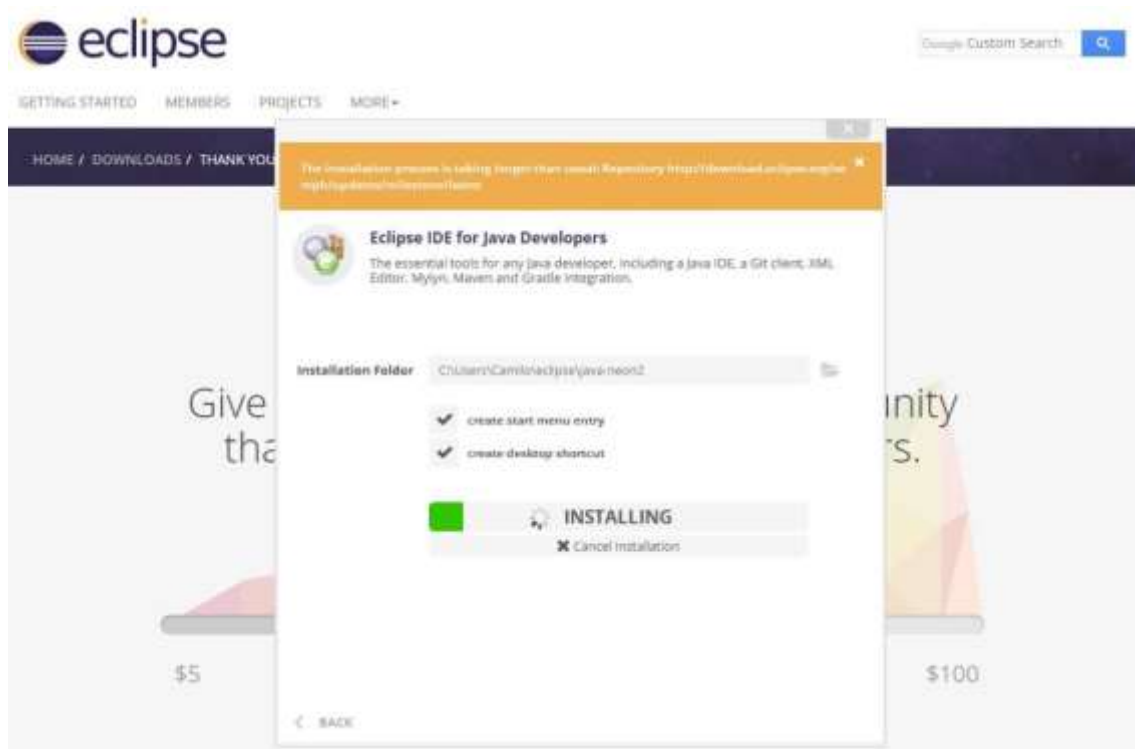
Tomado de: www.tutobasico.com/instalar-eclipse-windows/

Aceptamos términos y licencias de eclipse.



Tomado de: www.tutobasico.com/instalar-eclipse-windows/

Luego de hacer eso comenzara la descarga e instalación del programa.



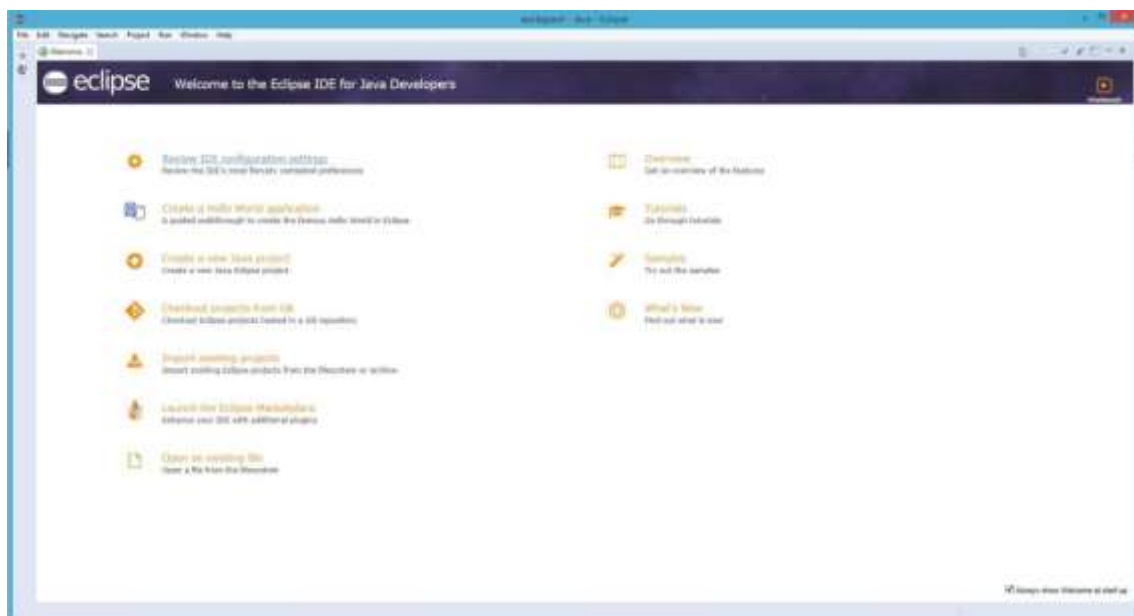
Tomado de: www.tutobasico.com/instalar-eclipse-windows/

Al terminar la instalación de le damos en launch para la primera configuración.



Tomado de: www.tutobasico.com/instalar-eclipse-windows/

Luego de hacer todo lo anterior correctamente se nos abrirá la interfaz de eclipse para java.



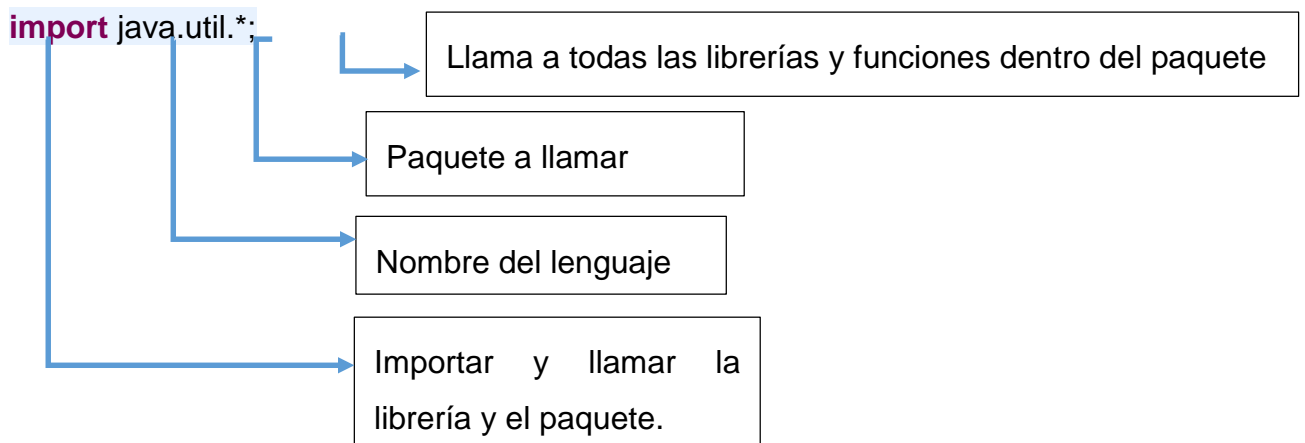
Package y librerías

¿Qué son?

Son componentes de cada lenguaje de programación con el fin de agrupar y comprimir la mayoría de funcionalidades del lenguaje, con el fin de evitar la mayor cantidad de líneas de código para crear cada librería. Para llamar a cada una se necesita poner la palabra **import**.

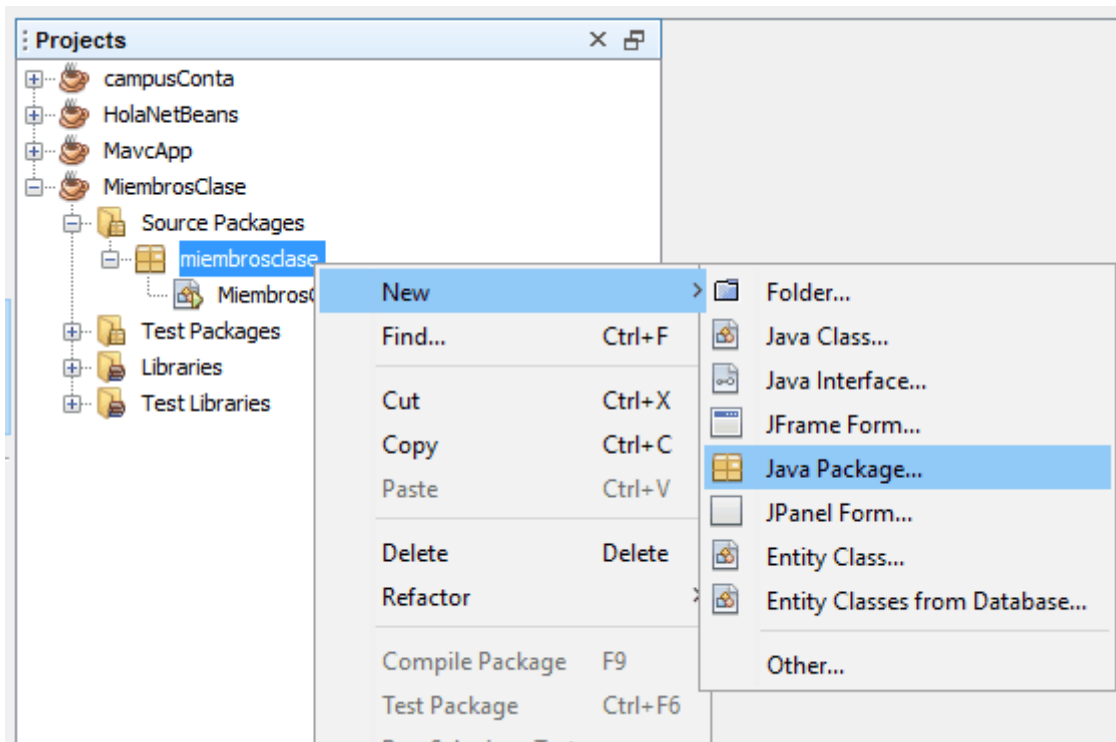
Como se sabemos en la creación de aplicaciones en java ya sea para escritorio o para móviles se necesita implementar la programación con clases, polimorfismo, etc. Cada uno de ellos se redondeará con la palabra **Package** y dentro de él conlleva cada clase que se necesite crear para el aplicativo, y dentro de cada clase es donde se hace el llamado de los **import** para que cada línea de código funcione correctamente con el fin de que la codificación sea limpia y correcta, Pero en programación competitiva son máximo dos o tres librerías que se llaman, pero dentro de ella se llaman muchas agregándole un asterisco al final.

Un ejemplo de una llamada de una librería sería.



¿Cómo se crea un Package en java?

En NetBeans la creación de paquetes empieza con la creación de un java application, y luego de ello daremos doble clic sobre el nombre del java application con el fin que se despliegue las carpetas predefinidas dentro de él, luego en el source Package hacemos clic derecho y lanzara todas las opciones y luego le daremos en new y en java Package, lo nombramos y luego de ello en el paquete creado hacemos lo mismo damos clic derecho, new pero le daremos a javaMainclass que se encuentra en other por si es la primera vez que usas NetBeans.



Tomado de: <https://www.campusmvp.es/recursos/post/paquetes-en-java-que-son-para-que-se-utilizan-y-como-se-usan.aspx>

Dentro de cada paquete pueden existir subpaquetes que cumplirán una función específica dentro del proyecto o aplicación que se esté creando todas estas tienen que tener una interrelación entre todas, pero no se puede hacer lo mismo de un paquete con otro y luego tratar de juntarlos y hacerlos funcionar.

Importación de las librerías y paquetes

Como se sabe las librerías son una parte esencial que traen componentes que se pueden utilizar en cualquier momento y para cualquier trabajo, esto hace que para crear un tipo de algoritmo y sea para realizar una simple suma o una concatenación como al momento de crear toda una aplicación ya que estas librerías contienen cada uno de las partes importantes para que funcione correctamente, un ejemplo básico quería para la lectura en donde se llama el paquete útil y la librería Scanner, esto permitirá el uso de esta línea de código para leer cada variable y/o datos que se necesite que entren por consola.

Los paquetes son:

- java.util
- java.lang
- java.io
- java.awt
- java.math

JAVA.UTIL

Este al ser un paquete de los más usados en cualquier software creado con java o por crear, contiene muchas clases e interfaces las cuales cada una de ellas tiene su funcionalidad, en donde si no se importan cada uno de ellas no servirán algunas funciones y/o líneas de códigos, las principales y más utilizadas interfaces y clases dentro del paquete son

INTERFACES EN .UTIL

CLASES EN .UTIL

COLLECTION	CALENDAR
COMPARATOR	DATE
ITERATOR	HASHMAP
LIST	HASHSET

MAP	LINKENDLIST
SET	TREEMAP
SORTEDMAP	TREESET
SORDEDSET	

Collections

Esta es una de las interfaces más utilizadas en cada uno de los programas y/o aplicaciones creadas con el lenguaje java, esta contiene una gran cantidad de operaciones, pero dentro de estas operaciones al momento de resolverlas se resuelven en una colección, con el fin de que se optimice y se resuelvan de manera rápida, dentro de estas operaciones se pueden añadir, borrar, comparar objetos de una colección.

Los tipos de Collections más utilizados son:

- **Set:** Este hace parte de la interfaz Collections en donde no se permite datos duplicados, aparte tiene métodos heredados de Collections y por eso se restringe los datos duplicados.
 - **HashSet:** Este es un conjunto que agrupa los elementos y no permite elementos repetidos.
 - **TreeSet:** Este es un conjunto que agrupa elementos en forma de árbol con el fin de tener los datos más estructurados.
 - **LinkedHashSet:** Este es un conjunto que agrupa elementos en una lista doble lazo con el fin de que se puede recorrer de la misma forma de adelante hacia atrás o viceversa.
- **List:** Son una sucesión de elementos en donde si se admite datos duplicados.

Dentro de una list se pueden hacer muchas operaciones, como lo es tener un acceso a una posición del elemento, buscar un elemento en la list, iterar sobre todo el elemento.

Hay dos tipos de list que podemos utilizar en nuestros programas, son los siguientes:

- **ArrayList**: Podemos ver un ArrayList como un tipo de arreglo dinámico, el cual crece con respecto a los datos que se van añadiendo o eliminando, de esta manera se puede hacer todo tipo de operaciones, búsquedas y arreglos que se necesiten en algún momento.
- **LinkedList**: aumenta el rendimiento y eficacia de los algoritmos, esta se basa en una lista doble lazo, en donde se puede hacer muchas operaciones con él ya que al ser doble lazo se tiene acceso a la cabeza y a la cola.

JAVA.IO

Este paquete es el encargado gestionar la mayoría de entrada y salida de datos, ya que como se sabe los datos se leen por parte del objeto InputStream y la salida de datos por el PrintStreams ya sea por consola y/o pantalla.

OBJETOS DEL PAQUETE IO

STREAM

InputStream y OutputStream

ObjectInputStream y ObjectOutputStream

FileInputStream y FileOutputStream

DataInputStream y DataOutputStream

Streams

Se pueden entender como la forma en la que se pueden agregar elementos y datos de una forma secuencial y paralela en una colección.

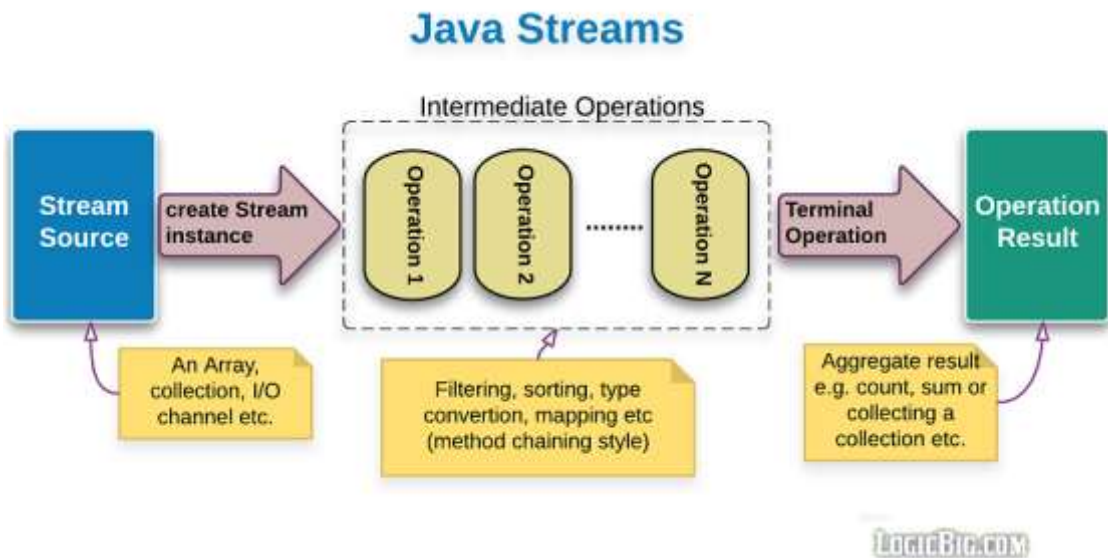


Imagen tomada de: www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/stream-api-intro.html

ObjectInputStream y ObjectOutputStream

Son dos tipos de flujo de datos y su principal función es modificar la lectura y escritura de objetos, y se derivan de los flujos de datos primitivos `DataInputStream` y `DataOutputStream`.

Clase:

```

package archivo4;

public class Lista implements java.io.Serializable {
    private int[] x; // array de datos
    private int n; // dimensión

    public Lista(int[] x) {
        this.x = x;
        n = x.length;
        ordenar();
    }

    public double valorMedio() {
        int suma = 0;
        for (int i = 0; i < n; i++) {
            suma += x[i];
        }
        return (double) suma / n;
    }
}
  
```

```

    }

    public int valorMayor() {
        return x[n - 1];
    }

    public int valorMenor() {
        return x[0];
    }

    private void ordenar() {
        int aux;
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                if (x[i] > x[j]) {
                    aux = x[j];
                    x[j] = x[i];
                    x[i] = aux;
                }
            }
        }
    }

    public String toString() {
        String texto = "";
        for (int i = 0; i < n; i++) {
            texto += "\t" + x[i];
        }
        return texto;
    }
}

```

App:

```

package archivo4;

import java.io.*;
import java.util.*;

public class app {
    static PrintWriter pw = new PrintWriter(System.out);

    public static void main(String[] args) {
        Lista lit = new Lista(new int[] { 12, 15, 11, 4, 32
    });

        try {
            ObjectOutputStream impre = new
ObjectOutputStream(new FileOutputStream("media.obj"));
            impre.writeObject("guardar este string y un
objeto\n");
            impre.writeObject(lit);
            impre.close();
        }
    }
}

```

```

        ObjectInputStream entrada = new
ObjectInputStream(new FileInputStream("media.obj"));
        String str = (String) entrada.readObject();
        Lista obj1 = (Lista) entrada.readObject();
        pw.println("Valor medio " + obj1.valorMedio());
        pw.println("-----");
        pw.println(str + obj1);
        pw.println("-----");
        entrada.close();
//se puede fundir en una catch Exception
    } catch (IOException ex) {
        pw.println(ex);
    } catch (ClassNotFoundException ex) {
        pw.println(ex);
    }

    try {
//espera la pulsación de una tecla y luego RETORNO
        System.in.read();
    } catch (Exception e) {
    }
    pw.close();
}
}

```

FileInputStream y FileOutputStream

Este tipo de entrada y salida de datos consta en la lectura y la compilación de datos en tipo byte, de esta manera obtiene los ficheros y los procesa en binario con el fin de lograr una pequeña encriptación de los datos.

Tomada de: www.discoduroderoer.es/clases-fileinputstream-y-fileoutputstream-para-ficheros-binarios-en-java/

DataInputStream y DataOutputStream

Esta clase cumple la función de leer, captar e imprimir datos primitivos de una manera rápida y eficaz.

La clase de lectura cuenta con las siguientes formas de leer los datos primitivos:

- boolean readBoolean();
- byte readByte();
- int readUnsignedByte();
- short readShort();
- int readUnsignedShort();
- char readChar();
- int readInt();

- String readLine();
- long readLong();
- float readFloat();
- double readDouble();

y en la clase la parte de impresión tiene las siguientes formas para cada tipo de dato primitivo:

- void writeBoolean(boolean v);
- void writeByte(int v);
- void writeBytes(String s);
- void writeShort(int v);
- void writeChars(String s);
- void writeChar(int v);
- void writeInt(int v);
- void writeLong(long v);
- void writeFloat(float v);
- void writeDouble(double v);

```
import java.io.*;
public class app {
    static PrintWriter pw = new PrintWriter(System.out);

    public static void main(String[] args) throws IOException {
        DataInputStream sc = new DataInputStream(System.in);
        String str;
        int n, aux = 0;
        pw.print("\n[" + aux + "] ");
        while ((str = sc.readLine()) != null) {
            n = Integer.parseInt(str);
            aux += n;
            pw.print("\n[" + aux + "] ");
        }
        pw.println("\n[" + aux + "]\n");
        pw.close();
    }
}
```

Tipos de lecturas e impresión

¿Qué es lectura de datos?

A diferencia de otros lenguajes la entrada de entrada de datos es un poco complicadas, las lecturas de datos son la forma en que el usuario interactúa o ingresa los datos para el procesamiento de los mismo con el fin de obtener un resultado esperado con el algoritmo creado.

En los métodos de lectura de datos hay 4 que son los mas utilizados o los más conocidos para el procesamiento de datos, lo cuales serían, lectura por teclado, lectura por consola, lectura por consola (fast) y procesamiento de textos.

Entrada por teclado:

Esta entrada consta en directamente en una operación en donde se ingrese directamente en el software que se este creado, con el fin de no crear o llamar ninguna clase de lectura.

```
import java.util.*;
import java.io.*;

public class codigos {
    public static void main(String arg[]) {
        String str;
        System.out.print("como te llamas: ");
        str = System.console().readLine();
        System.out.println("Hola " + str +
            ", ¡bienvenido al curso de java para principiantes!");
    }
}
```

Clase de lectura Scanner

Este método es uno de los más utilizados y conocidos por los programadores, este analiza todos los tipos de datos primitivos y expresiones regulares aunque por medio de comandos se puede hacer la lectura de datos.

```
import java.util.*;
```

```
import java.io.*;

public class codigos {
    public static void main(String arg[]) {
        PrintWriter pw = new PrintWriter(System.out);
        Scanner sc = new Scanner(System.in);
        String a = sc.nextLine();
        pw.println("el string es: " + a);
        int x = sc.nextInt(), y = sc.nextInt();
        int c = x + y;
        pw.println("los numeros ingresados son: " + x + ", "
+ y + " y la suma de ambos es: " + c);
        pw.close();
        sc.close();
    }
}
```

Clase de lectura BufferedReader e InputStreamReader

La lectura del BufferedReader consiste en un tipo de entrada por lectura de String, básicamente hace una lectura de las funciones principales de los datos primitivos en forma de String, con el fin de que cada lectura será más rápida y se pueda ejecutar y compilar de manera efectiva y completa, la parte del InputStreamReader lo que cumple es que hace que los datos y/o datos que se vayan a ingresar sean captados por teclado y procesados por el Buffered, una de las características es que toca colocarle las excepciones si es en caso numérico o de error como lo sería el IOException o el NumberFormatException.

```
import java.util.*;
import java.io.*;

public class codigos {
    public static void main(String arg[]) throws
NumberFormatException, IOException {
        PrintWriter pw = new PrintWriter(System.out);
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        int a= Integer.parseInt(br.readLine());
        int b= Integer.parseInt(br.readLine());
        int c=a*b;
        int d=a+b;
        pw.println("La suma de los dos numero leidos con
BufferedReader es: "+d+" y la multiplicación es: "+c);
    }
}
```


Clase Console

Esta clase es la preferida para leer los datos de usuarios ya que su seguridad no permite en ciertos casos caracteres repetidos, un ejemplo puede ser en una contraseña de un login para un sistema de ingreso.

```
import java.util.*;
import java.io.*;

public class codigos {
    public static void main(String arg[]){
        PrintWriter pw = new PrintWriter(System.out);
        String str = System.console().readLine();
        pw.println(str);
        pw.close();
    }
}
```

Ventajas

- En la clase console no hay repetición de caracteres.
- El `BufferedReader` permite lectura de datos contenedores.
- El `BufferedReader` permite guardar la lectura eficiente por un cierto tiempo.
- El `Scanner` permite la lectura de los datos primitivos haciéndolo mucho más entendible y menos tedioso de programar.

Desventajas

- En el `BufferedReader` la programación es muy tediosa y difícil de recordar.
- Al momento de leer no se sincroniza correctamente y puede haber pérdida de datos.
- La clase console tiene un gran problema y es que no puede ser compilado y/o ejecutado en un IDE.

¿Qué es salida de datos?

La salida de datos es una forma de mostrar en pantalla el objetivo que cumplió el software creado, esto se hace para cumplir una necesidad y/o testear que el software este cumpliendo su función correctamente, la mayoría de veces no se tiene que importar librerías ya que son propias del lenguaje, en cambio si se va

a usar la impresión de `PrintWriter` si se necesita importar la librería “`java.io.PrintWriter;`”, con el fin de llamar a todas sus funciones y sus propiedades.

Clase System (salida de datos)

La salida de datos de la clase `System` es la más común y la más utilizada entre los programadores o desarrolladores que utilicen este lenguaje en su día a día, esta salida tiene tres métodos de impresión, `print()`, `println()` y `printf()`, cada una de ellas cumple una función distinta y se pueden utilizar para resolver muchas situaciones en las que se pida una impresión específica.

- Impresión `print`:

Este formato de impresión muestra los que se quiere imprimir, pero sin el formateo de un salto de línea, por ejemplo, si se quiere imprimir “`Hola mundo`” sin salto de línea usamos este método y el resultado será el mismo sin ningún salto de línea.

```
public class codigos {  
    public static void main(String arg[]) {  
        String str = "Hola mundo";  
        System.out.print(str);  
        System.out.print(" como estas");  
    }  
}
```

Este algoritmo imprimirá las dos frases en la misma línea.

- Impresión `Println`:

Este formato de impresión es muy similar al `print` con la pequeña diferencia del (`ln`) que cumple la función de aplicar un salto de línea, esto con el fin de hacer una impresión más limpia o simplemente cumplir con los requerimientos dados.

```
public class codigos {  
    public static void main(String arg[]) {  
        String str = "Hola mundo";  
        System.out.println(str);  
        System.out.println("como estas");  
    }  
}
```

Este algoritmo imprimirá cada uno en línea separada.

- **Impresión Printf:**

Este formato de impresión es un formato “**print Formatted**” o impresión formateada, este recibe parámetros uno que es fijo y otro que es la cadena de formato que se va a imprimir.

El símbolo “%” es el que identifica el formateo y luego sigue una letra pegada con la que se identifica el tipo de dato a imprimir, la siguiente tabla muestra las letras y su función.

Letra	Función
%c	Imprime el carácter correspondiente
%d, %i	Hace una conversión para imprimir un entero
%x	Impresión de un hexadecimal sin signo
%p	Impresión de un puntero o dirección de memoria
%e	Impresión de un flotante con notación científica
%f	Impresión un flotante con signo y usando el punto decimal
%g	Impresión de un flotante usando la notación que no tenga tanto espacio
%o	Impresión de un octal
%u	Impresión decimal sin signo
%s	Impresión de un String

%%	Imprime el símbolo %
----	----------------------

Ejemplo de impresión formato printf

```
import java.util.*;

public class codigos {
    public static void main(String arg[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Estas son las distintas formas
del printf");
        int x=10;
        System.out.printf("%d %n", x);

        System.out.println("_____
");
        double a=12.3698;
        System.out.printf("%.2f %n", a);

        System.out.println("_____
");
        double n = 1.25036;
        System.out.printf("%.3f %n", n);

        System.out.println("_____
");
        double v = 1.25036;
        System.out.printf("%+.3f %n", v);

        System.out.println("_____
");
        double z = 1.25036;
        int y = 10;
        System.out.printf("z = %.2f y = %d %n", z, y);

        System.out.println("_____
");
        double c = 1.25036;
        System.out.println("Aquí se hace un corrimiento de
los espacios que se especifique");
        System.out.printf("%+10.2f %n", c);

        System.out.println("_____
");
        sc.close();
    }
}
```

Clase PrintWriter

Esta clase permite hacer una impresión con tipos de formateos en una salida de stream de texto, esto quiere decir que se puede crear un ficho en donde se puede guardar cada dato de entrada y de salida con esta forma, si se necesita un respaldo de datos de entrada y salida para una siguiente prueba, esta puede ser una forma más fácil de guardarlos mientras los va probando. Si se necesita una impresión por consola se puede hacer una modificación en la invocación del objeto.

```
import java.io.*;

public class codigos {
    public static void main(String arg[]) throws IOException {
        FileWriter Escrituradedatos = new
        FileWriter("c:/ficheros/datos.txt");
        PrintWriter impresiondedatos = new
        PrintWriter(Escrituradedatos);
        impresiondedatos.close();
    }
}
```

Este ejemplo es una escritura de ficheros y una impresión de los mismos.

```
import java.io.*;
import java.util.*;
public class codigos {
    public static void main(String arg[]) throws IOException {
        Scanner sc= new Scanner(System.in);
        PrintWriter pw = new PrintWriter(System.out);
        int x=sc.nextInt();
        int z=sc.nextInt();
        pw.println(x*z);
        pw.close();
        sc.close();
    }
}
```

En este ejemplo podemos observar la forma en la que podemos ver la salida por consola, esto hace que se puedan usar en competencias de programación y/o solo para testear el algoritmo.

Clases tipo envoltorio o wrapper

Este tipo de clases se encargan de como dice su nombre envolver a los datos primitivos como los son las variables estándar, tiene la opción de manipular los primitivos, con el fin de envolverlos y convertirlos de una manera en la cual le aplican todos sus contenidos.

En muchos casos o situaciones los datos primitivos son necesarios para resolver un problema en particular con sus distintas operaciones, pero le es imposible realizarlas al tener un tipo de dato así, entonces es cuando se llaman las clases envoltorios con el fin de poder utilizar ese tipo de dato con su envoltorio y las funciones totales que este tiene, como consiguiente se mostrara los tipos de datos primitivos con sus respectivos envoltorios.

Tipo primitivo	Clase contenedora
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

como se muestra en la anterior tabla cada uno tiene su envoltorio específico con el fin de utilizar el tipo de dato primitivo a su máximo uso con una mayor cantidad de eficiencia, todas estas clases se encuentran en el paquete interno **java.lang.***, en donde el * hace que se llamen todas las clases dentro de el para una mayor efectividad y reducción de código.

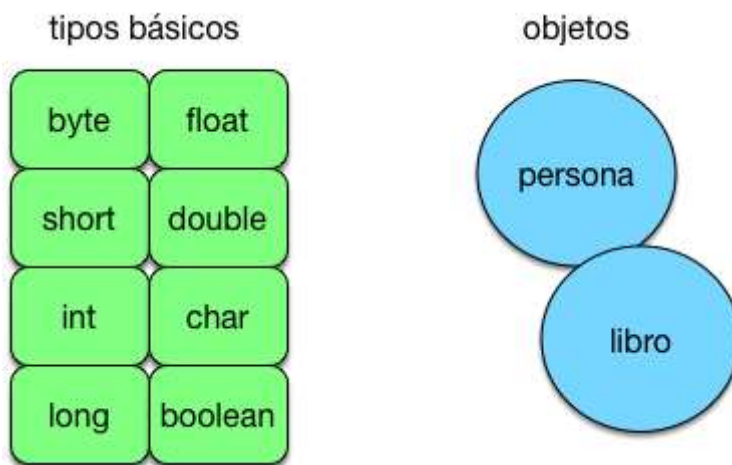
¿Cómo es el proceso para hacer paso a un envoltorio?

Las clasificaciones Boxing y Unboxing en los códigos y algoritmos en java son unos conceptos muy abstractos que al momento de empezar a codificar en este lenguaje no se tiene en claridad y mucho menos como sería el funcionamiento al cambiar de tipo primitivo a clase envoltorio, para la clasificación y proceso de estos conceptos hay que tener la claridad de que es un objeto y de cuáles son

los tipos de datos preestablecidos de java, los tipos de datos básicos ya preestablecidos son 8 los cuales son los enteros, booleanos, chars y flotantes, todos los demás que lleguen ya son objetos.

Boxing:

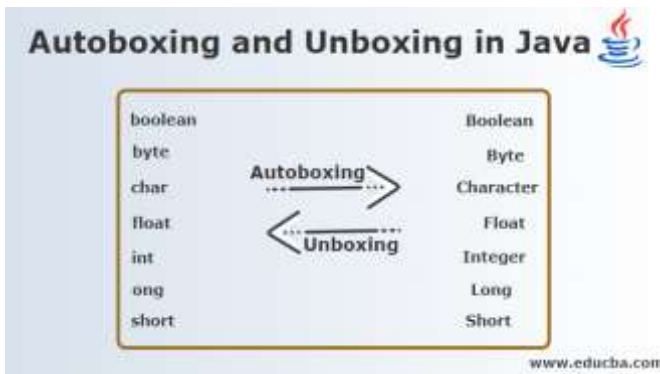
Se puede entender como Boxing en java como la entrada de un tipo de dato primitivo en un tipo de dato objeto, con el fin de crear una herencia entre clase para una codificación más limpia clara y concreta, de manera que otros programadores puedan entender tu código e interpretarlo de la manera correcta.



Tomado de: <https://www.arquitecturajava.com/wp-content/uploads/JavaBoxingTipos-1.gif>

Unboxing:

Este método busca extraer los datos que hayamos creados en una clase objeto con el fin de poder llamarlo en cualquier momento en cualquier parte del código ya que es de forma heredada creada en otra clase externa.

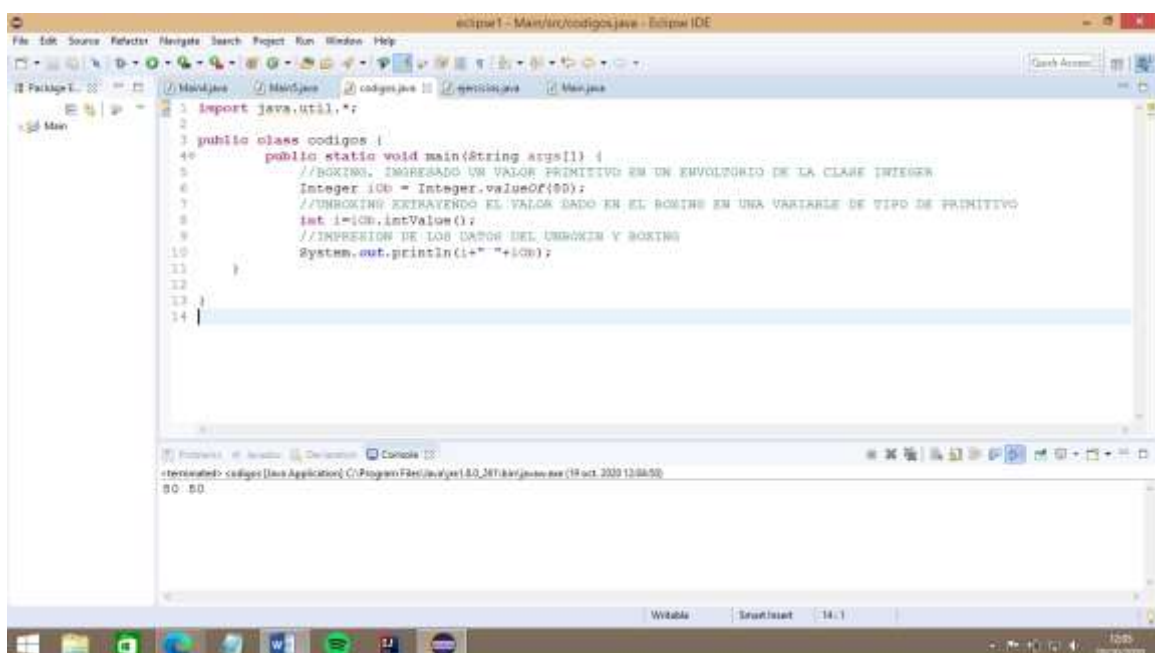


Tomado de: <https://cdn.educba.com/academy/wp-content/uploads/2019/07/Autoboxing-and-Unboxing-in-Java.png>

Un ejemplo de código de Boxing y Unboxing sería el siguiente:

```
import java.util.*;

public class Main {
    public static void main(String args[]) {
        //BOXING, INGRESADO UN VALOR PRIMITIVO EN UN
        ENVOLTORIO DE LA CLASE INTEGER
        Integer iOb = Integer.valueOf(80);
        //UNBOXING EXTRAYENDO EL VALOR DADO EN EL BOXING
        EN UNA VARIABLE DE TIPO DE PRIMITIVO
        int i=iOb.intValue();
        //IMPRESION DE LOS DATOS DEL UNBOXIN Y BOXING
        System.out.println(i+ " "+iOb);
    }
}
```



Clase Integer

Esta clase es la que se encarga de los valores numéricos que envuelve al dato primitivo `int` el cual tiene una restricción de memoria de $2^{31} - 1$ para valores positivos y -2^{31} para valores negativos, por lo tanto, este se usa muchas veces para la creación de estructuras básicas no tan extremas como lo podría ser una calculadora básica para hacer cálculos básicos de aritmética básica.

Hay formas de creación o estructuras básicas estándar en las cuales uno no puede crearlas o llamarla con el tipo de dato primitivo, un ejemplo de ello es crear un `ArrayList` (Es una forma de crear arrays dinámicos lo que significa que no tiene necesidad de decirle un tamaño como un array fijo), en donde sí se coloca en la creación del mismo un `int` dará un error de compilación ya que en su librería estándar se predefine que necesita el envoltorio de dicha variable de la cual se vaya a crear.

```
import java.util.*;
public class Main {

    public static void main(String[] args) {
        //array creado con envoltorio Integer
        ArrayList<Integer> arr= new ArrayList<>();
        //array creado con el dato primitivo int el cual
        genera error y fallo del código
        ArrayList<int> arr2= new ArrayList<>();
    }

}
```

En la mayoría de casos al llamar al envoltorio para una utilización de muchas veces de él es necesario crear unos constructores para saber de qué tipo va a ser el envoltorio, ya que puede ser el envoltorio “Integer” y puede tomar directamente los valores de un `int` o también se puede manejar de tipo “String”.

```
import java.util.*;

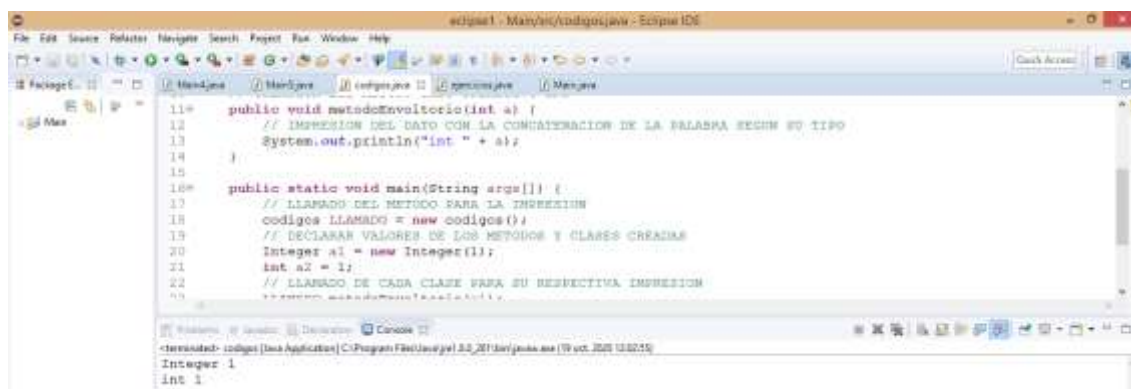
public class códigos {
    // CREACION DEL METODO ENVOLTORIO INTEGER
    public void metodoEnvoltorio(Integer a) {
        // IMPRESION DEL DATO CON LA CONCATENACION DE LA
        PALABRA SEGUN SU TIPO
        System.out.println("Integer " + a);
    }
}
```

```

// CREACION DEL METODO DATO PRIMITIVO INT
public void metodoEnvoltorio(int a) {
    // IMPRESION DEL DATO CON LA CONCATENACION DE LA
PALABRA SEGUN SU TIPO
    System.out.println("int " + a);
}

public static void main(String args[]) {
    // LLAMADO DEL METODO PARA LA IMPRESION
    códigos LLAMADO = new códigos();
    // DECLARAR VALORES DE LOS METODOS Y CLASES CREADAS
    Integer a1 = new Integer(1);
    int a2 = 1;
    // LLAMADO DE CADA CLASE PARA SU RESPECTIVA IMPRESION
    LLAMADO.metodoEnvoltorio(a1);
    LLAMADO.metodoEnvoltorio(a2);
}
}

```



Tomado de: Propio

Clase Character:

La clase Character es la que se encarga de la parte de los caracteres en general, allí se pueden clasificar los caracteres numéricos, los caracteres alfabéticos y especial, siempre y cuando sean del lenguaje en español y se encuentren en la tabla del ASCII (es un código que utiliza 7 bits para representar cada carácter con el fin de darle un valor a cada uno para que se facilite el trabajo al momento de codificar String o chars).

El tipo de dato primitivo es el char que habitualmente se usa y su clase envoltorio es Character, al ser un tipo carácter tiene sus propios métodos específicos para

hallar un dato o un carácter en específico, algunos de estos métodos son tipos booleanos para saber si es verdadero o falso, los métodos son:

Nombre de la función	Descripción
isLetter();	Verifica si el char dado es una letra.
isDigit();	Verifica si el char dado es un dígito.
isWhitespace();	Verifica si el char es un espacio en blanco.
isUpperCase();	Verifica si el char dado se encuentra en mayúsculas.
isLowerCase();	Verifica si el char dado se encuentra en minúsculas.
toUpperCase();	Convierte el char en mayúsculas.
toLowerCase();	Convierte el char en minúsculas
toString();	Convierte el char en un tipo cadena de caracteres.

```
import java.io.PrintWriter;  
import java.util.*;  
  
public class códigos {  
    public static void main(String args[]) {  
        PrintWriter pw = new PrintWriter(System.out);  
        // CREACION DEL OBJECJO CHARACTER ESPECIFICANDO SU  
        VALOR  
        Character ch = new Character('C');  
        pw.println("ISLETTER");  
        // IMPRESIONES PARA VERIFICAR SI ES UN CHARACTER
```

```

        pw.println(ch.isLetter(ch));
        pw.println(Character.isLetter('8'));

        pw.println("_____
        ");

        pw.println("ISDIGIT");
        pw.println(ch.isDigit(ch));
        pw.println(Character.isDigit('1'));

        pw.println("_____
        ");

        pw.println("ISWHITESPACE");
        pw.println(ch.isWhitespace(ch));
        pw.println(Character.isWhitespace(' '));

        pw.println("_____
        ");

        pw.println("ISUPPERCASE");
        pw.println(Character.isUpperCase(ch));
        pw.println(Character.isUpperCase('N'));

        pw.println("_____
        ");

        pw.println("ISLOWERCASE");
        pw.println(Character.isLowerCase(ch));
        pw.println(Character.isLowerCase('n'));

        pw.println("_____
        ");

        pw.println("TOUPPERCASE");
        pw.println(Character.toUpperCase('b'));
        pw.println(Character.toUpperCase('c'));

        pw.println("_____
        ");

        pw.println("TOLOWERCASE");
        pw.println(Character.toLowerCase('B'));
        pw.println(Character.toLowerCase('C'));

        pw.println("_____
        ");

        pw.println("TOSTRING");
        pw.println(Character.toString('t'));
        pw.println(Character.toString('H'));

        pw.close();
    }

```

Como podemos observar la clase Character puede contener cualquier tipo de carácter que se encuentre en la tabla ASCII y se puede hacer modificaciones y verificaciones para facilitar muchos trabajos que toque realizar con este tipo de variable.



La clase String es una de las más importantes por no decir la más importante, ya que esta trabaja con cadenas de caracteres haciéndola la más utilizada en las aplicaciones hechas con java, son tan utilizados ya que con estas cadenas se pueden almacenar texto el cual también se puede procesar de manera que se puede capturar lo que ingresa un usuario o también lo que traigan de otro lado y sea una cadena de texto se puede guardar e igual procesar con el fin de hacer muchas operaciones. Esta clase tiene una serie de funciones muy útiles para acceder, modificar todos los caracteres que se encuentren en dicha cadena, cabe aclarar que en un String se pueden incluir caracteres alfabéticos, numéricos, especiales y signos de puntuación.

length()	Devuelve todo el tamaño de la cadena
indexOf('caracter')	Devuelve la posición de la primera aparición de <i>carácter</i> dentro del String. Devuelve -1 si no lo encuentra.
lastIndexOf('caracter')	Devuelve la posición de la última aparición de <i>carácter</i> dentro del String. Devuelve -1 si no lo encuentra.
charAt(n)	Muestra el carácter de la posición n
substring(n1,n2)	Devuelve la subcadena desde la posición n1 hasta n2 – 1
toUpperCase()	Convierte la cadena en Mayúscula
toLowerCase()	Convierte la cadena en minúscula
equals(otroString)	Compara dos String y devuelve true si son iguales
equalsIgnoreCase(otroString)	Igual que equals pero sin considerar mayúsculas y minúsculas
compareTo(OtroString)	Devuelve 0 si las dos cadenas son iguales. <0 si la primera es alfabéticamente menor que la segunda ó >0 si la primera es alfabéticamente mayor que la segunda.
compareToIgnoreCase(OtroString)	Igual que compareTo pero sin considerar mayúsculas y minúsculas.
valueOf(N)	Convierte la variable N a String. N puede ser de cualquier tipo de dato.

Un código de ejemplo para una de estas funciones seria el siguiente:

```
import java.util.*;
import java.io.*;
public class codigos {
    public static void main(String[] args) {
        PrintWriter pw = new PrintWriter(System.out);
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        while (sc.hasNextLine()) {
            s = sc.nextLine();
            s = s.replaceAll("_", "");
            if (s.equals("")) {
                break;
            }
            s = s.replace("|", "");
            s = s.replace(".", "");
            s = s.replace(" ", "0");
            s = s.replace("o", "1");
            int a = Integer.parseInt(s, 2);
            char imp = (char) a;
            pw.print(imp);
        }
    }
}
```

```

        pw.close();
        sc.close();
    }
}

```

Lo que hace el código es reemplazar los caracteres que se le indiquen por el especificado en el replace, a diferencia que fuera el replaceAll que cambia todo lo que se le indique en el String.

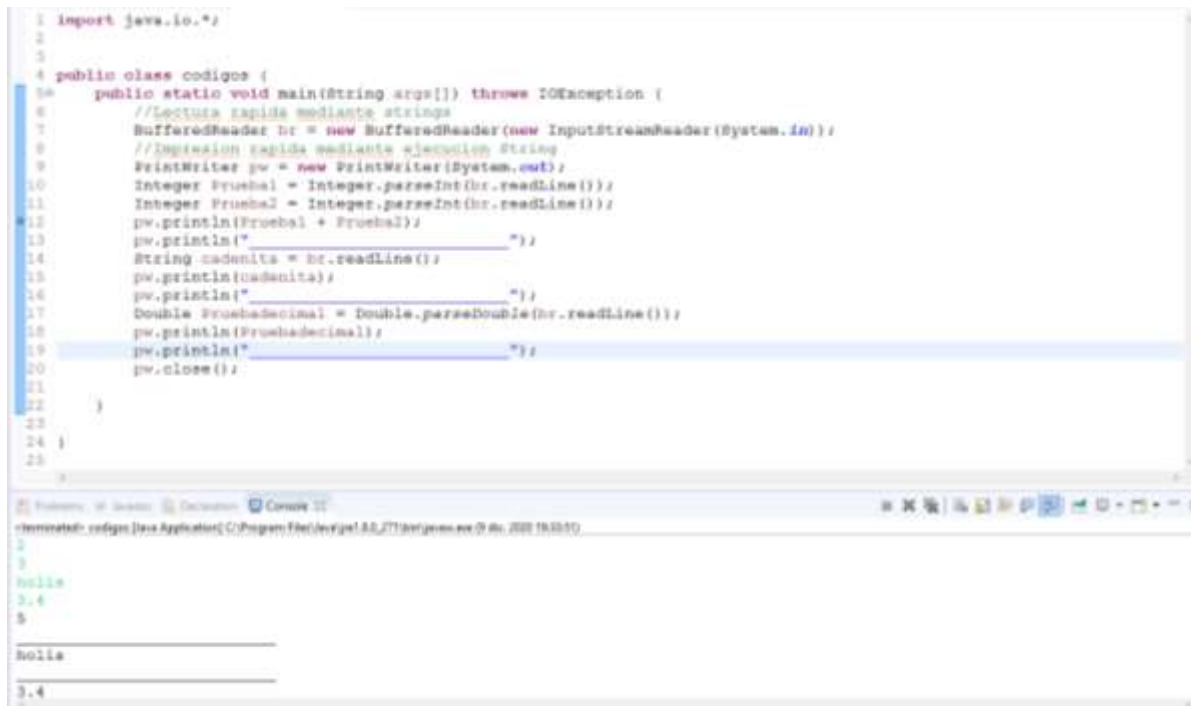
Al momento de usar los String como se pueden llamar datos de tipo carácter y datos de tipos numéricos, pueden llamar cada instancia cuando se necesite ya que cada desarrollador el utilizar String lo que hará es reducir el tiempo y reducción de líneas de código al no tener que crear muchas variables de otros tipos ya que en esta se pueden hacer directamente, un ejemplo de esto sería el comento de utilizar una lectura rápida de tipo String para el procesamiento de datos ingresados por teclado, como lo sería el BufferedReader.

```

import java.io.*;

public class códigos {
    public static void main(String args[]) throws IOException {
        //Lectura rápida mediante String
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        //Impresión rápida mediante ejecución String
        PrintWriter pw = new PrintWriter(System.out);
        Integer Prueba1 = Integer.parseInt(br.readLine());
        Integer Prueba2 = Integer.parseInt(br.readLine());
        pw.println(Prueba1 + Prueba2);
        pw.println("_____");
        String cadenita = br.readLine();
        pw.println(cadenita);
        pw.println("_____");
        Double Pruebadecimal =
Double.parseDouble(br.readLine());
        pw.println(Pruebadecimal);
        pw.println("_____");
        pw.close();
    }
}

```



```
1 import java.io.*;
2
3
4 public class codigos {
5     public static void main(String args[]) throws IOException {
6         //Lectura rápida mediante strings
7         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
8         //Impresión rápida mediante el escritor String
9         PrintWriter pw = new PrintWriter(System.out);
10        Integer Prueba1 = Integer.parseInt(br.readLine());
11        Integer Prueba2 = Integer.parseInt(br.readLine());
12        pw.println(Prueba1 + Prueba2);
13        pw.println(" ");
14        String cadenita = br.readLine();
15        pw.println(cadenita);
16        pw.println(" ");
17        Double Pruebadecimal = Double.parseDouble(br.readLine());
18        pw.println(Pruebadecimal);
19        pw.println(" ");
20        pw.close();
21    }
22 }
23
24
25
```

hola

3.4

Tomado de: Propio

Esto sería un ejemplo de una lectura de datos mediante la lectura rápida por String, esto hace que su ejecución y compilación sea más rápida.kop

Constantes

¿Qué son las constantes?

Las constantes se pueden definir como un tipo de dato no modificable, ya que al momento de su creación se asigna un espacio de memoria con su valor, se puede usar las veces que se requiera en el programa, pero nunca se puede alterar su valor ya que como dice su nombre es constante siempre mantiene el mismo valor.

```
import java.io.*;
import java.util.*;
```



```

public class codigos {
    public static void main(String arg[]) throws IOException {
        Scanner sc = new Scanner(System.in);
        PrintWriter pw = new PrintWriter(System.out);
        int a = 20; // variable modificable
        final int x = 90; // constante fija sin opcion de
modificar.
        pw.close();
        sc.close();
    }
}

```

Operadores Aritméticos

Los operadores aritméticos son funciones que te permiten hacer distintas operaciones matemáticas dentro de nuestros algoritmos, como bien sabemos los principales operadores son la suma, la resta, la multiplicación, la división y el residuo o en programación el modular, cada uno de estos operadores se representan mediante sus símbolos de operación igual que como lo hacemos en matemáticas.

El símbolo de + cumple la función de adición o suma.

El símbolo de – cumple la función de sustracción o resta.

El símbolo de * cumple la función de multiplicación.

El símbolo de / cumple la función de división.

El símbolo de % cumple la función de residuo o modular.

```

import java.io.*;
import java.util.*;

public class codigos {
    public static void main(String arg[]) throws IOException {
        Scanner sc = new Scanner(System.in);
        PrintWriter pw = new PrintWriter(System.out);
        System.out.println("Digite dos numeros");
        float a = sc.nextFloat(), b = sc.nextFloat(), suma,
resta, multi, divi, mod;
        suma = a + b;
        resta = a - b;
        multi = a * b;
        divi = a / b;
        mod = a % b;
        pw.println("el resultado de la suma es: " + suma);
        pw.println("el resultado de la resta es: " + resta);
    }
}

```

```

        pw.println("el resultado de la multiplicacion es: " +
multi);
        pw.println("el resultado de la division es: " +
divi);
        pw.println("el resultado del residuo es: " + mod);
        pw.close();
        sc.close();
    }
}

```

Nota:

¡Muchas personas no saben que es y cómo funciona el operador modular, la función de este es obtener el residuo de una división, por ejemplo, si dividimos el 2 entre el mismo dos y su residuo será 0 esa es la función principal del modular!

En los operadores aritméticos se pueden hacer las mismas variaciones con la misma variable una y otra vez, digamos podemos sumarle 5 a una variable inicializada en lo que quiera, un ejemplo de esto sería lo siguiente.

```

import java.io.*;
import java.util.*;

public class codigos {
    public static void main(String arg[]) throws IOException {
        Scanner sc = new Scanner(System.in);
        PrintWriter pw = new PrintWriter(System.out);
        System.out.println("Digite un numero");
        int x=sc.nextInt();
        x+=10;
        pw.println(x);
        System.out.println("Digite un numero");
        int y=sc.nextInt();
        y-=20;
        pw.println(y);
        System.out.println("Digite un numero");
        int z=sc.nextInt();
        z*=10;
        pw.println(z);
        pw.close();
        sc.close();
    }
}

```

En donde el `x+=10` cumple la función `x= x+ 10`, esto quiere decir que podemos modificar las variables de esta manera o hacer una operación básica.

Ejercicio de practica

(Puedes practicar el tema de operadores básicos con el ejercicio “Add Two Numbers” de la plataforma de entrenamiento de programación competitiva CodeChef y el código del problema es FLOW001.)

Operadores de incremento ++ y decremento --

Los operadores de incremento y decremento cumplen una función básica de aumentar en uno su valor o disminuirlo, esto hace que podamos usar estos operadores como un contador o una variable autoincremento.

```
import java.io.*;
import java.util.*;

public class codigos {
    public static void main(String arg[]) throws IOException {
        Scanner sc = new Scanner(System.in);
        PrintWriter pw = new PrintWriter(System.out);
        System.out.println("Digite dos numeros para
incrementar y disminuir");
        int x=sc.nextInt();
        int z=sc.nextInt();
        x++;
        z--;
        pw.println(x);
        pw.println(z);
        pw.close();
        sc.close();
    }
}
```

Clase Math

Es una de la librerías mas importantes y más utilizadas en java, con el fin poder lograr hacer muchas operaciones sin necesidad de utilizar algún tipo de constructor o propiedades, algunas de las funciones matemáticas son:

Funciones Matemáticas	Significado	Ejemplo de uso	Resultado
abs	Valor absoluto	int x = Math.abs(2.3);	x = 2;
atan	Arcotangente	double x = Math.atan(1);	x = 0.78539816339744;
sin	Seno	double x = Math.sin(0.5);	x = 0.4794255386042;
cos	Coseno	double x = Math.cos(0.5);	x = 0.87758256189037;
tan	Tangente	double x = Math.tan(0.5);	x = 0.54630248984379;
exp	Exponenciación neperiana	double x = Math.exp(1);	x = 2.71828182845904;
log	Logaritmo neperiano	double x = Math.log(2.7172);	x = 0.99960193833500;
pow	Potencia	double x = Math.pow(2.3);	x = 8.0;
round	Redondeo	double x = Math.round(2.5);	x = 3;
random	Número aleatorio	double x = Math.random();	x = 0.20614522323378;

Tomado de:

www.dis.um.es/~lopezquesada/documentos/IES_1415/IAW/curso/UT3/ActividadesAlumnos/java7/paginas/pag13.html

En general las más utilizadas son:

- Math.abs: Retorna el valor absoluto del parámetro dado.
- Math.floorDiv: Retorna el valor más bajo en entero del resultado de una división.
- Math.floorMod: Retorna el valor más bajo en entero del resultado de un residuo.
- Math.max: Retorna el valor del máximo de dos valores.
- Math.min: Retorna el valor del mínimo de dos valores.
- Math.round: Retorna en un valor entero el redondeo de un número decimal.
- Math.E: Retorna el valor de la constante Euler.
- Math.Pi: Retorna el valor de la constante PI.
- Math.floor: Retorna el valor más debajo del resultado de una división.
- Math.sqrt: Retorna el valor del resultado de una raíz cuadrada.
- Math.pow: Retorna como valor el resultado de la potencia.

```
import java.io.*;
import java.util.*;

public class codigos {
```

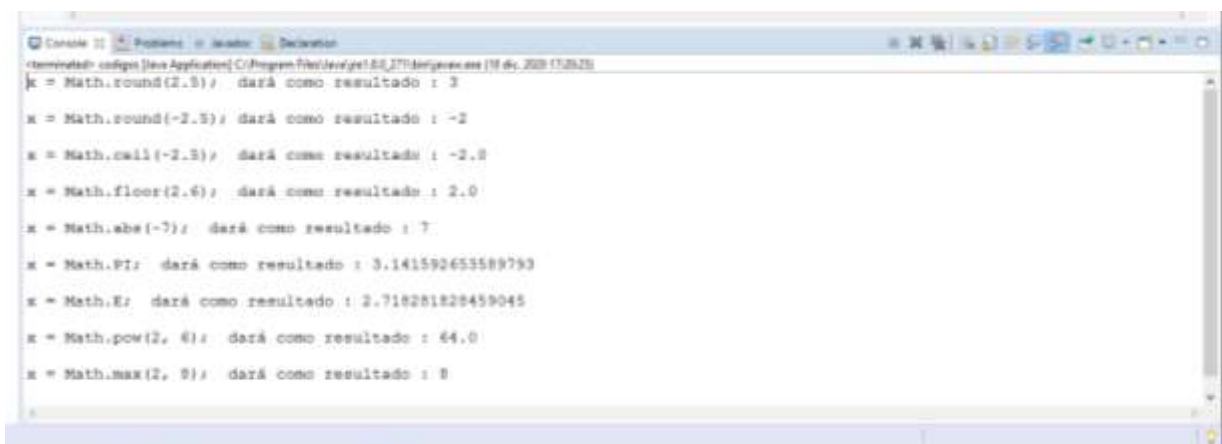
```

double x = 0;

public static void main(String arg[]) throws
IOException {
    Scanner sc = new Scanner(System.in);
    PrintWriter pw = new PrintWriter(System.out);
    pw.println("x = Math.round(2.5); dará como
resultado : " + Math.round(2.5) + "\n");
    pw.println("x = Math.round(-2.5); dará como
resultado : " + Math.round(-2.5) + "\n");
    pw.println("x = Math.ceil(-2.5); dará como
resultado : " + Math.ceil(-2.5) + "\n");
    pw.println("x = Math.floor(2.6); dará como
resultado : " + Math.floor(2.6) + "\n");
    pw.println("x = Math.abs(-7); dará como
resultado : " + Math.abs(-7) + "\n");
    pw.println("x = Math.PI; dará como resultado : "
+ Math.PI + "\n");
    pw.println("x = Math.E; dará como resultado : "
+ Math.E + "\n");
    pw.println("x = Math.pow(2, 6); dará como
resultado : " + Math.pow(2, 6) + "\n");
    pw.println("x = Math.max(2, 8); dará como
resultado : " + Math.max(2, 8) + "\n");
    pw.close();
    sc.close();

}
}

```



```

C:\Program Files\Java\jdk-1.8.0_202\bin\java.exe [JVM args]
x = Math.round(2.5); dará como resultado : 3
x = Math.round(-2.5); dará como resultado : -2
x = Math.ceil(-2.5); dará como resultado : -2.0
x = Math.floor(2.6); dará como resultado : 2.0
x = Math.abs(-7); dará como resultado : 7
x = Math.PI; dará como resultado : 3.141592653589793
x = Math.E; dará como resultado : 2.718281828459045
x = Math.pow(2, 6); dará como resultado : 64.0
x = Math.max(2, 8); dará como resultado : 8

```

Tomado de: Propio

Ciclos y Condicionales

Sentencia If-Else-Elseif

If:

Esta es una de las sentencias más utilizadas y simples, esta se usa mas con el fin de declarar si se cumple una condición permite seguir el algoritmo y realizar una operación que este dentro de esta condicional.

Su estructura básica es la siguiente:

```
if (condición) {  
    declaracion1;  
    declaracion2;  
}
```

La condicional retorna true si es verdadero o false si no es así, esto significa que si tenemos una variable declarada en 10 podemos verificar en un if si es mayor o menor a un numero x, y se le da la orden de impresión, si es verdadero o true este imprimirá, si no lanzara un error advirtiéndolo que algo está mal.

```
import java.util.*;  
import java.io.*;  
  
public class Main4 {  
    public static void main(String[] args) {  
        PrintWriter pw = new PrintWriter(System.out);  
        Scanner sc = new Scanner(System.in);  
        int a = sc.nextInt();  
        if (a < 16) {  
            pw.println("la condicional if retorno  
true");  
        }  
        pw.close();  
    }  
}
```

```
}
```

Sentencia else

Esta sentencia es un complemento de la sentencia if, esta cumple la función de verificar si no se cumplió la condición que se declaró en el if, se saltará y se ejecutará la orden que se le haya dado, por ejemplo, si se tiene una condición en un if de que una variable a es menor a una variable x, si se cumple se ejecuta la orden dada dentro de él, pero si no, lo que sucede es que se salta a ejecutar la condicional else.

```
import java.util.*;
import java.io.*;

public class Main4 {
    public static void main(String[] args) {
        PrintWriter pw = new PrintWriter(System.out);
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        if (a < 16) {
            pw.println("la condicional if se ejecuto");
        } else {
            pw.println("la condicional if no se cumplio
y el else se ejecuto");
        }
        pw.close();
    }
}
```

Anidación if

Esta anidación if nos permite verificar una condicional dentro de otra con el fin de verificar distintas instrucciones al mismo tiempo para no usar muchos if individuales, esto hace que se pueda optimizar tiempo de ejecución y de procesamiento del algoritmo.

```
import java.io.*;
import java.util.*;

public class app {
    static PrintWriter pw = new PrintWriter(System.out);
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) throws IOException {
        int i = sc.nextInt();
    }
}
```

```

        if (i < 10) {
            if (i < 15) {
                pw.println("soy menor que 15");
            }
            if (i < 12) {
                pw.println("yo también soy menor que 12");
            } else {
                pw.println("soy mayor que 15");
            }
        }
        pw.close();
    }
}

```

Sentencia Switch

Este tipo de sentencias son muy utilizadas cuando se necesitan muchas operaciones pero se necesita que sea algo muy corto, en estos casos se utilizan los switch, este funciona con los tipos de datos primitivos ya vistos.

La estructura básica de un switch es la siguiente:

```

switch(expresión){
    // declaración case

    // los valores deben ser del mismo tipo de la expresión

    case valor1 :
        // Declaraciones
        break; // break es opcional

    case valor2 :
        // Declaraciones
        break; // break es opcional

    // Podemos tener cualquier número de declaraciones de casos o case

    default :
        // Declaraciones
}

```


Como se puede observar la sentencia switch consta de varios case los cuales serán las opciones que se ejecutaran y se elegirá con fin de ejecutar lo que este dentro de estos case, un ejemplo muy claro de esto, es cuando entramos a las opciones de un operador móvil en donde tenemos diferentes opciones a elegir.

Limitaciones del switch:

- La expresión puede ser de tipo int, char o una enumeración. Y en las últimas actualizaciones de java se pueden usar string.
- Las declaraciones de un case no se pueden duplicar.
- La declaración predeterminada es opcional.
- La declaración es opcional. Si se omite, la ejecución continuará en el siguiente case.

```
import javax.swing.JOptionPane;

public class app {
    public static void main(String[] args) {

        String menu = new String("Elija una opción: 1, 2, 3 o
4 \n");
        for (int i = 1; i <= 4; i++) {
            menu = menu + " opción " + i + "\n";
        }
        String option =
JOptionPane.showInputDialog(menu.toString());
        switch (option) {
            case "1":
                JOptionPane.showMessageDialog(null,
"Felicitaciones, ha elegido la opción 1");
                break;
            case "2":
                JOptionPane.showMessageDialog(null,
"Felicitaciones, ha elegido la opción 2");
                break;
            case "3":
                JOptionPane.showMessageDialog(null,
"Felicitaciones, ha elegido la opción 3");
```

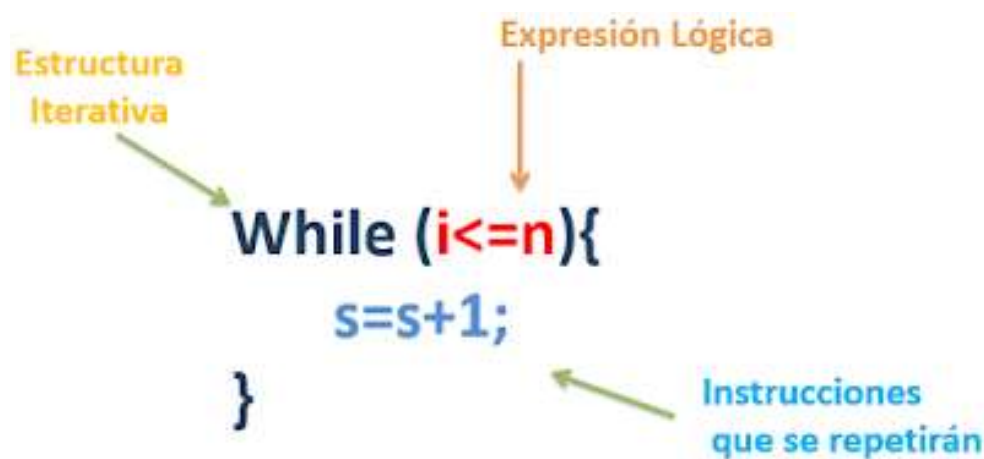
```

        break;
    case "4":
        JOptionPane.showMessageDialog(null,
"Felicitaciones, ha elegido la opción 4");
        break;
    default:
        JOptionPane.showMessageDialog(null, option + "
no es una opción válida");
        break;
    }
}
}

```

Ciclo while

Este ciclo funciona de una manera muy distinta a lo que sería el ciclo for, la forma de funcionamiento de este es una repetición siempre y cuando se cumpla una condición, y se deja de ejecutar y sale hasta que se rompa la condición.



Tomado de: www.incanatoit.com/2016/03/estructura-iterativa-bucle-while-curso.html

```

public class app {
    public static void main(String[] args) {
        int i = 0;

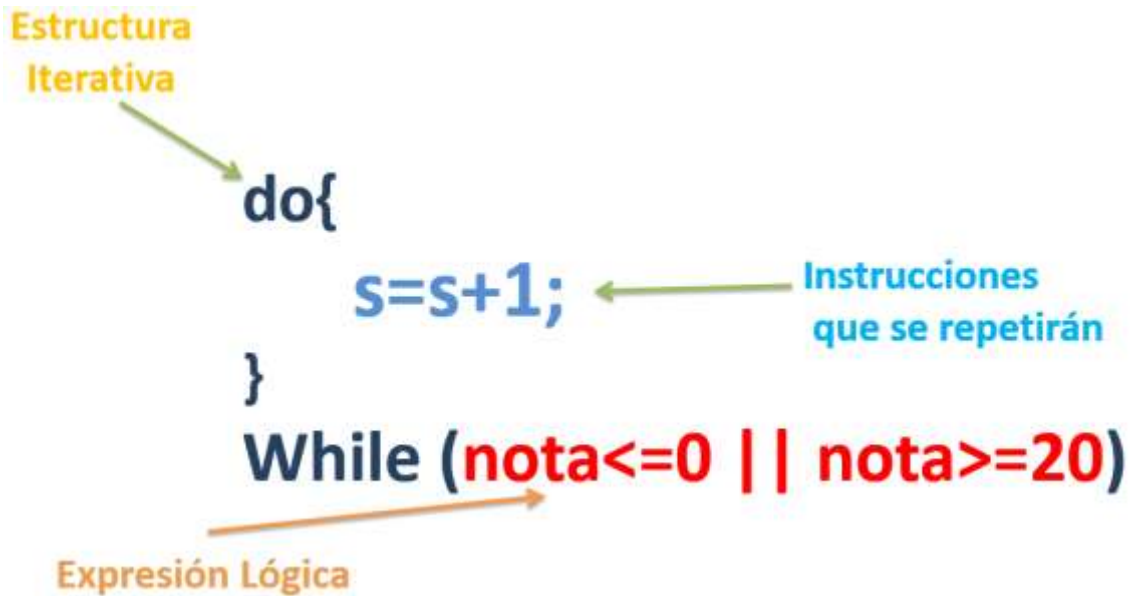
        while (true) { //el true quiere decir que no se rompe
            hasta que sé que le indique.
            i++;
            System.out.println("Valor de i: " + i);
            if (i == 9) {
                break;
            }
        }
    }
}

```

}

Ciclo Do-While

Este ciclo es muy similar al while con la ligera diferencia que primero se ejecutan las instrucciones y luego verifica si la condición en el while es verdadera o no.

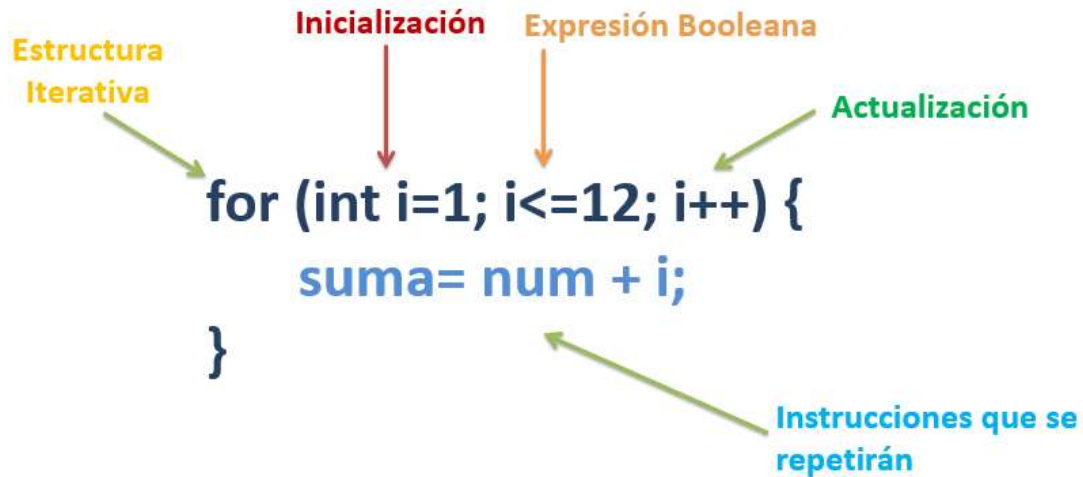


Tomado de: www.incanatoit.com/2016/01/estructura-iterativa-bucle-do-while-programacion-csharp-net.html

```
public class app {  
    public static void main(String[] args) {  
        int cont = 0;  
  
        do {  
            System.out.println("Aumentando... " + (cont +  
1));  
  
            cont += 1;  
        } while (cont < 10);  
    }  
}
```

Ciclo for

Este ciclo es una estructura de control que nos permite ejecutar una orden cierta cantidad de veces requerida o dada en la sentencia interna del for.

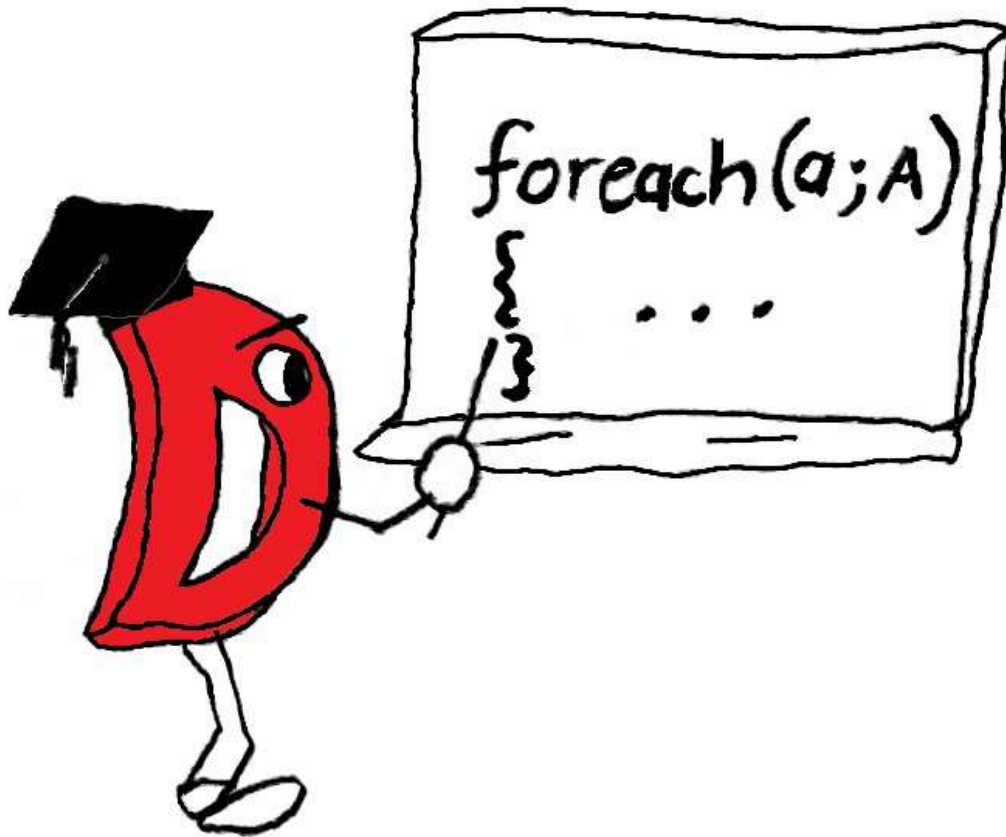


Tomado de: www.incanatoit.com/2014/12/estructura-iterativa-bucle-for-programacion-csharp-net.html

```
public class app {  
    public static void main(String[] args) {  
        for (int i = 500; i <= 1000; i+=2) {  
            System.out.println(i);  
        }  
    }  
}
```

En el ejemplo anterior realizamos un ciclo for que se ejecutará e imprimirá todos los números pares desde el numero 500 hasta el 1000.

Ciclo forEach



Tomado de: www.tour.dlang.org/tour/es/basics/foreach

El bucle o ciclo foreach o for mejorado, es otro tipo de ciclo que principalmente se utiliza para el recorrido y llenado de un arreglo de una forma más corta y sencilla.

Su sintaxis básica es la siguiente:

```
for (tipo variable: array) {  
    declaraciones usando variable;  
}
```

Generalmente en un algoritmo muy grande y con grandes arreglos se usan los dos for, el normal y el foreach con el sentido de que el algoritmo sea más rápido y optimo, ya que el foreach carece de algunas características del for normal, en el siguiente ejemplo se mostrara un código con un for normal y luego su equivalencia en un foreach.

FOR:

```
public class Main {

    public static void main(String[] args) {
        int [] vec= {1,2,3,4,5};
        int suma=0;
        for (int i = 0; i < vec.length; i++) {
            suma+=vec[i];
        }

    }

}
```

FOREACH:

```
public class Main {

    public static void main(String[] args) {
        int [] vec= {1,2,3,4,5};
        int suma=0;
        for (int x : vec) {
            suma+=x;
        }

    }

}
```

Teniendo en cuenta lo anterior haremos un ejemplo haciendo iteraciones con un foreach en un arreglo.

```
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<String> jugadores = new ArrayList<>();
        jugadores.add("Jordan");
        jugadores.add("Gasol");
        jugadores.add("Ronaldo");
        jugadores.add("Messi");
        int cont = 0;
        System.out.println("Los jugadores agregados en el
arreglo son: ");

        for (String str : jugadores) {
            System.out.println((cont + 1) + ".- " + str);
            cont++;
        }

    }

}
```

```
}  
}
```

Funciones y métodos

Métodos

Estos pueden hacer lo mismo que una función, son totalmente parecidos con la diferencia de que los métodos se relacionan mucho con objetos o con una clase, en estos casos se denotan con unos nombres en específico que harán que cumplan una función, el public o private, esto no afecta en nada, en si se pueden hacer operaciones, recibir valores y demás cosas y lo hará, aunque en java se debería hablar de métodos y no de funciones ya que en java se usan mucho lo que son los objetos, aunque para que sirva la función debe ser static y retornar algún valor si lo necesita.

Funciones

Son una serie de instrucciones que se escriben en un bloque de código con el fin de optimizar el código, esto se utiliza en la programación orientada a objetos (POO), estas funciones hacen que el código se vea más limpio y presentable en un proyecto conjunto, muchas veces las funciones son la manera más fácil de resolver algún problema en específico ya que pueden resolverlos allí y solamente con llamarlos en la función main se resuelve todo, hay distintos tipos de funciones, hay una que retornan un valor, otras que no retornan nada y otras que llaman a ellas mismas para volverlas recursivas.

Procedimientos

Se entiende que los procedimientos en programación son un conjunto de instrucciones que se ejecutan y no retornan valor y en si a veces no reciben ningún dato o valor, la notación de este es un void que quiere decir que no es necesario colocar la sentencia return.

Códigos de ejemplo:

Métodos:

```
public String metodoString(int n) { // método con un parámetro
    if (n == 0) { // verificamos el parámetro
        return "a"; // si se cumple retorna una a y si no
        retorna una b
    }
    return "b";
}
```

Método con dos parámetros:

```
static boolean metodoBoolean(boolean n, String mensaje) { //
Método con dos parámetros

    if (n) { // Usamos el parámetro en el método

        System.out.println(mensaje); // Mostramos el
mensaje
    }
    return n; // Usamos el parámetro como valor a
retornar
}
```

Procedimientos:

```
void procedimiento(int n, String nombre) { // Notar el void {
    if (n > 0 && !nombre.equals("")) { // usamos los dos
parámetros {
        System.out.println("hola " + nombre);
        return; // Si no ponemos este return se mostraría
hola y luego adiós
    }
    // También podríamos usar un else en vez del return
    System.out.println("adiós");
}
```

Ejemplo en conjunto

```
public class app {
    public static String saludar(String nombre) {

        String saludo = "Hola. Bienvenido " + nombre;

        return saludo;
    }
}
```



```

        public static String error(String nombre) {
            String error = "Ups. No pudimos validar tus datos. "
+ nombre + " es tu usuario?";

            return error;
        }

        public static void verificar(String usuario, String
contrasenia) {
            String usuarioValido = "Juliansilvit";

            String contraseniaValida = "contraseña";

            if (usuarioValido.equals(usuario) &&
contraseniaValida.equals(contrasenia)) {

                System.out.println(saludar(usuario));
                return;
            }

            System.out.println(error(usuario));
        }

        public static void main(String[] args) {
            String usuario = "Juliansilvit";
            String contrasenia = "contraseña";

            verificar(usuario, contrasenia);
        }
    }
}

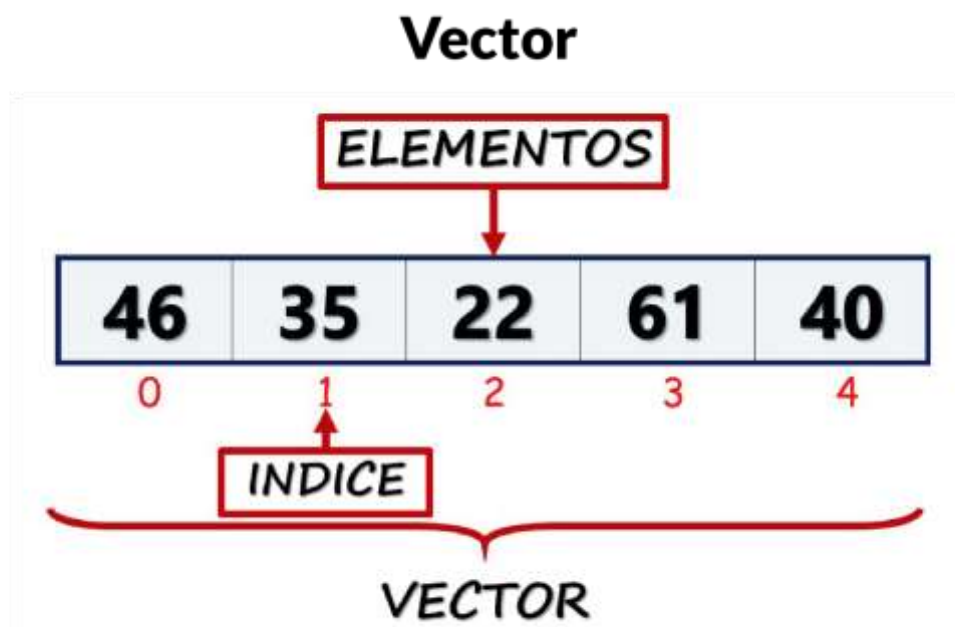
```

Estructuras de datos



¿Qué es una estructura de datos?

Como su nombre lo indica son unas estructuras de organización de datos que se pueden usar de una manera eficaz y optima en un algoritmo, generalmente estas estructuras de datos se utilizan en la mayoría de aplicativos con el fin de optimizarlos y darles orden según los requerimientos que estos necesiten, son muy usadas en las bases de datos que reciben grandes cantidades de datos, de esta manera se organizan y categorizan para una mejor búsqueda y planificación de la información.



Guía del programador competitivo

Tomado de: Guía del programador competitivo

Arrays:

Se conoce como array, vector o arreglo, a una de las estructuras de datos que permite hacer una colección de elementos de un mismo tipo, este vector o array tiene un tamaño fijo el cual una vez establecido no se puede modificar ni disminuir ni agrandar, al ser una colección de elementos de un mismo tipo este permite ordenar, enlazar, buscar y mostrar el elemento que se necesite buscar,

de esta manera también se permite hacer ciertas operaciones matemáticas entre ellos, al juntar dos y más arrays de cierta manera podemos formar una matriz de datos la cual nos permitirá entrar a otro tipo de procedimientos posibles.

Un arreglo siempre va a empezar desde 0 ya que es la posición inicial desde cualquier ciclo, esto es de suma importancia tenerlo en cuenta para que luego al crear un algoritmo no se generen errores.

Su sintaxis básica es:

Tipo_de_variable **[** **]** **Nombre_del_array** **=**
new Tipo_de_variable[tamaño_vec];

Un ejemplo de comprensión de la anterior sintaxis seria el siguiente:

```
public class Main {
    public static void main(String[] args) {
        int[] vec = new int[6];
        // o tambien se le pueden dar los datos
directamente
        int [] vec_con_datos= {1,5,2,5,3};
    }
}
```

Cuando requerimos llenar, buscar o decirle a nuestro algoritmo que nos muestre algún tipo de datos que este dentro el arreglo, recurrimos a la ayuda de los ciclos para que sea mucho mas sencillo.

```
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a=10;
        int vec [] = new int [a];
        for (int i = 0; i < vec.length; i++) {
            vec[i]=a--;
        }
        for (int i = 0; i < vec.length; i++) {
            System.out.println(vec[i]);
        }
    }
}
```

La función del algoritmo anterior es declarar 10 posiciones de memoria en el arreglo en el cual se almacenarán en orden descendente desde el numero 10 que es el que está almacenado en la variable a, hasta contar las 10 posiciones, pero también hay una palabra nueva en este algoritmo la cual es LENGTH.

Length:

Esta palabra reservada nos quiere decir o indicar el tamaño completo del vector, de esta manera podemos hacer un llenado más completo y rápido del mismo, ya que si en alguna situación se te olvida la cantidad de datos que tenía el vector con él "length" se puede saber, un ejemplo es el siguiente:

```
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        int vec []= new int [a];
        System.out.println("El tamaño del vector es de: "+
vec.length+" espacios de memoria");
    }
}
```

ArrayList, lista o vector tamaño dinámico:

En este tipo de vectores su mayor característica es que el tamaño varía según los datos que se agregan o se eliminan, siempre que se agrega un dato este se coloca en la cola del vector.

Estos arrays contienen métodos de búsqueda para ellos, los mas utilizados son los siguientes:

Agregar datos:

Add(objeto);

Eliminar datos:

Remove(objeto);

Mirar tamaño del array:

Size();

Código de ejemplo:

```
import java.util.ArrayList;

public class arraylist {
    public static void main(String[] args) {
        // se crea el array
        ArrayList arraisito = new ArrayList();

        // se verifica el tamaño de la lista antes de ser
        // llenado
        System.out.println("Tamaño de ArrayList en la
        creación:" + arraisito.size());

        // elemento que se agregaran al array
        arraisito.add("JULIAN");
        arraisito.add("EDWIN");
        arraisito.add("DANIEL");
        arraisito.add("LUNA");

        // verificacion de la lista despues de agregar datos
        System.out.println("Tamaño de ArrayList después de
        agregar elementos:" + arraisito.size());

        // se muestran todos los elementos dentro de la lista
        System.out.println("Lista de todos los elementos:" +
        arraisito);

        // eliminar contenido de la lista
        arraisito.remove ("DANIEL");
        System.out.println("Ver contenido después de eliminar
        un elemento:" + arraisito);

        // eliminar contenido de la lista por indice
        arraisito.remove (1);
        System.out.println("Ver contenido después de eliminar
        elemento por índice:" + arraisito);

        // verificacion de tamaño de la lista despues de
        // eliminar elementos
        System.out.println("Tamaño de arrayList después de
        eliminar elementos:" + arraisito.size());
        // mostrar los elementos que quedaron en la lista
        System.out.println("Lista de todos los elementos
        después de eliminar elementos:" + arraisito);

        // Verifica si la lista contiene "LUNA"
        System.out.println(arraisito.contains("LUNA"));
    }
}
```

Vector Sets o vector sin repeticiones:

Este tipo de vector se maneja como una colección de objetos que no tienen ningún orden en especial y la principal característica de este es que no permite repeticiones de datos dentro de él.

```
import java.io.*;
import java.util.*;

public class Sets {

    public static void main(String[] args) throws Throwable,
    IOException {
        Scanner sc = new Scanner(System.in);
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        HashSet<Integer> set = new HashSet<Integer>();
        for (int i = 0; i < 10; i++) {
            if (set.add(Integer.parseInt(br.readLine())) {

                } else {
                    System.out.println("Este dato ya existe en
el set");
                }

            }
        Iterator<Integer> it = set.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }

    }
}
```

Matrices:

Matriz

Llenando la fila 0 y las columnas del 0 al 3:
MatrizDeEnteros [0] [0] = 1;
MatrizDeEnteros [0] [1] = 3;
MatrizDeEnteros [0] [2] = 5;
MatrizDeEnteros [0] [3] = 7;

Llenando la fila 1 y las columnas del 0 al 3:
MatrizDeEnteros [1] [0] = 5;
MatrizDeEnteros [1] [1] = 4;
MatrizDeEnteros [1] [2] = 1;
MatrizDeEnteros [1] [3] = 16;

Llenando la fila 2 y las columnas del 0 al 3:
MatrizDeEnteros [2] [0] = 7;
MatrizDeEnteros [2] [1] = 9;
MatrizDeEnteros [2] [2] = 61;
MatrizDeEnteros [2] [3] = 13;

		Columnas			
		0	1	2	3
Filas	0	1	3	5	7
	1	5	4	1	16
	2	7	9	61	13

MatrizDeEnteros [0] [3]

Guía del programador competitivo

Tomado de: Guía del programador competitivo

Las matrices o arreglos bidimensionales, son unos arreglos ordenados por filas y columnas que se usan para almacenar datos en cada espacio de memoria.

La forma de creación de estos arreglos es similar a un array normal con la diferencia que se le agrega otro corchete cuadrado, de la siguiente manera:

Tipo_de_variable[] []Nombre_de_la_matriz=

new Tipo_de_variable[tamaño_filas][tamaño_columnas];

y la forma de recorrer la matriz es similar con la diferencia de que se agrega otro ciclo for para las columnas.

Un ejemplo de ello seria las matrices 2D y 3D, el siguiente código es de una matriz 3x3:

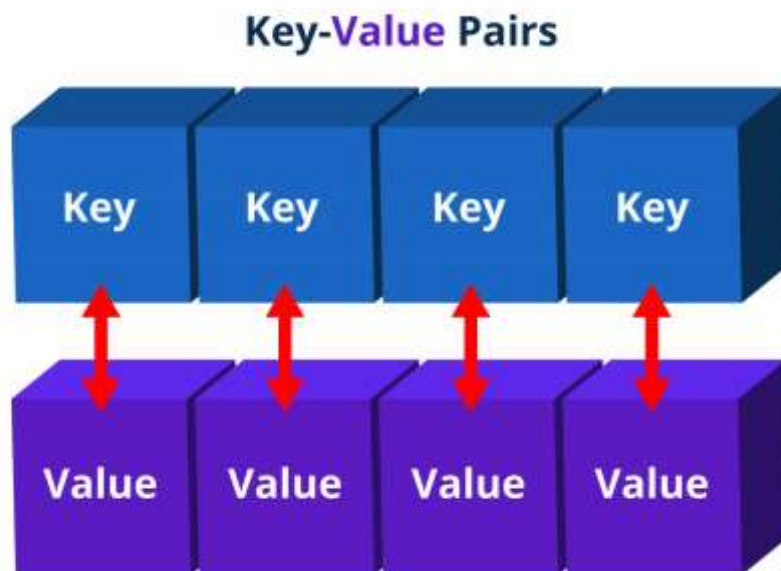
```
import java.util.*;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int matrix[][] = new int[3][3];
```

```

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}

```

Diccionarios o mapa (hashmap, treemap o map)



tomado de: www.edureka.co/blog/java-hashmap/

Los diccionarios o maps son estructuras de datos que ayudan a almacenar valores y claves, de esta manera un valor solamente tiene una clave, este tipo de estructuras se usan mayormente en los arboles binarios ya que permite agregar, borrar y modificar elementos.

Ejemplo:


```

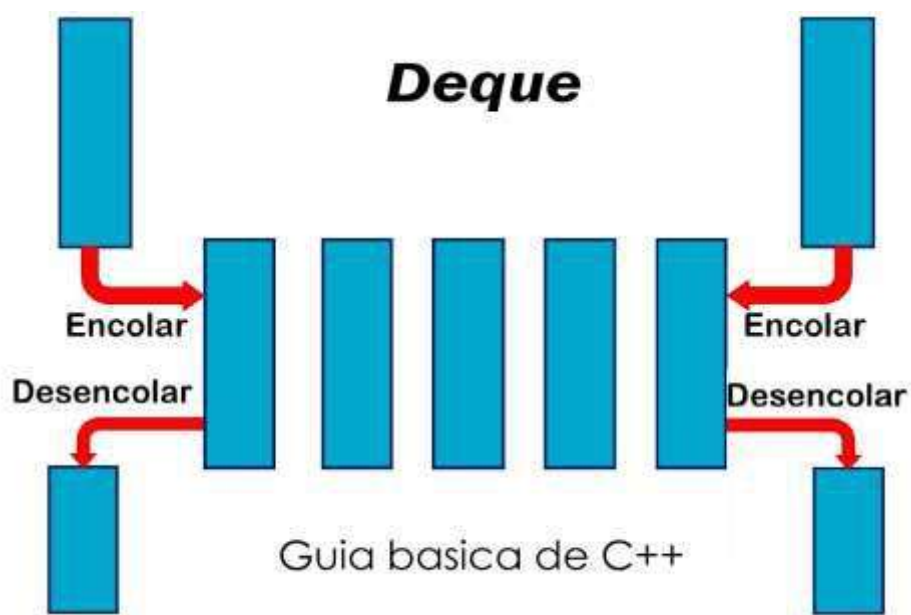
import java.util.*;

public class Diccionarios {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> dicc = new HashMap<String,
String>();
        String a = sc.nextLine(), b = sc.nextLine();
        dicc.put(a, b);
        dicc.put("futbol", "deporte con una pelota y arcos");
        dicc.put("vainazo", "lo que se tiran en el grupo de
discord");
        dicc.put("impostor", "among us");
        dicc.put("edwin", "parlon de miedo");
        for (String concepto : dicc.keySet()) {
            String key = concepto;
            String value = dicc.get(concepto);
            System.out.println(key + "->" + value);
        }
    }
}

```

Cola:



Tomado de: Guía del programador competitivo

La estructura de datos COLA o DEQUE es una lista de varios elementos que estos se ingresan en la parte final llamada cola, y si se necesita sacar o eliminar se hace por el otro lado llamada cabeza, esta estructura maneja una forma llamada fifo (First in, First Out), Primero en entrar es el primero en salir.

Esta estructura tiene sus propios métodos de para el manejo de ella, son los siguientes:

Para insertar datos:

-add(objeto)

-offer(objeto)

Para eliminar datos:

-remove()

-poll()

Para consultar datos del frente:

-element()

-peek()

Un código de ejemplo mostrando todos sus métodos de modificación de datos es:

```
import java.util.*;

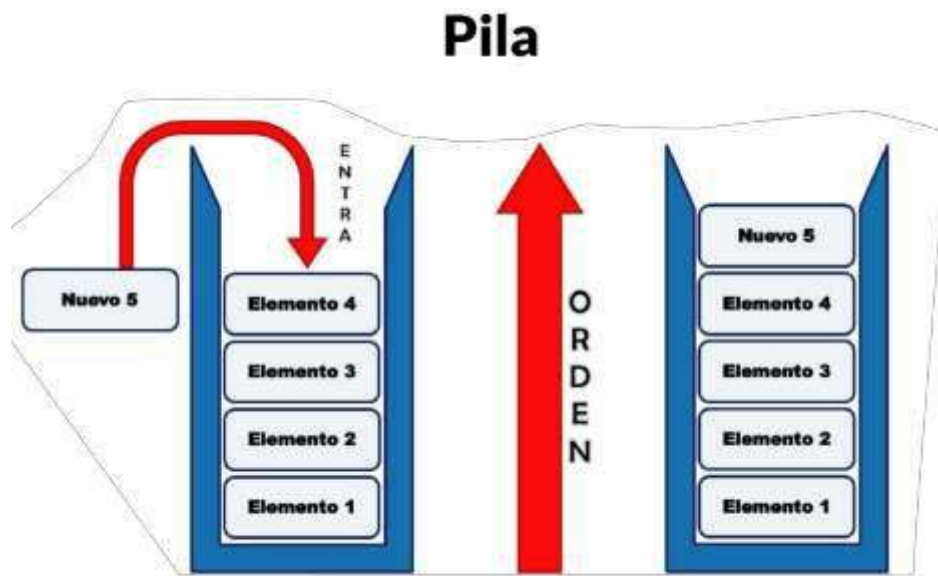
public class Cola {
    public static void main(String[] args) {
        // se crea la cola y se indica que tipo de dato
        es
        Queue<Integer> cola = new LinkedList();
        // se ingresan datos mediante los metodos ya
        enseñados
        cola.offer(3);
        cola.add(14);
        cola.offer(12);
        cola.add(7);
        cola.offer(10);
        //se imprime con los datos ingresados
        System.out.println("Cola llena: " + cola);
        // Esta es la estructura basica para sacar datos
        while (cola.poll() != null) { // se saca un dato y
        se compara con null a ver si era
            System.out.println(cola.peek()); //nos
        muestra la nueva posicion del frente
        }
    }
}
```

```

        //Se nos mostrara un null ya que la cola esta
        vacia
        System.out.print(cola.peek());
    }
}

```

Stack o Pila:



Guía del programador competitivo

Tomado de: Guía del programador competitivo

La pila o Stack es una estructura de datos de manejo lineal, a esta estructura solamente se le pueden ingresar, eliminar o consultar elementos por un punto fijo de la lista, este tipo de estructura maneja un tipo de manejo llamado (LIFO) Last in, First out.

```

import java.util.*;
import java.io.*;

public class Pila {
    static Scanner sc = new Scanner(System.in);
    static PrintWriter pw = new PrintWriter(System.out);
}

```

```

public static void main(String[] args) {

    Random r = new Random();
    Stack<Integer> pila = new Stack<>();
    int a = sc.nextInt();
    for (int i = 0; i < a; i++) {
        int num = r.nextInt();
        pw.println(num);
        pila.push(num);
    }
    pw.println("-----");
    while (!pila.isEmpty()) {
        pw.print(pila.pop() + "\t");
    }
    pw.close();
    sc.close();
}
}

```

Código de repaso de las anteriores estructuras:

```

import java.util.*;

public class Repaso_Estructuras {

    static Random r = new Random();
    static int i, j;
    static Scanner sc = new Scanner(System.in);
    public static void main(String args[]) {

        System.out.println("Inserte cantidad de datos");
        int n = sc.nextInt();
        System.out.println("Que estructura quiere
utilizar? (Los números son aleatorios");
        System.out.println("1) Vector fijo ");
        System.out.println("2) Vector dinámico");
        System.out.println("3) Vector sin repetición");
        System.out.println("4) Matriz");
        System.out.println("5) Diccionario");
        System.out.println("6) cola");
        System.out.println("7) pila");
        int aux = sc.nextInt();
        switch (aux) {
            case 1:
                vectorfijo(n);
                break;
            case 2:

```

```

        vectordinamico(n);
        break;
    case 3:
        vectorsinrepeticion(n);
        break;
    case 4:
        matriz(n);
        break;
    case 5:
        Diccionario(n);
        break;
    case 6:
        cola(n);
        break;
    case 7:
        pila(n);
        break;
    }
}

public static void vectordinamico(int n) {
    ArrayList<Integer> arreglo = new ArrayList<>();
    for (i = 0; i < n; i++) {
        arreglo.add(r.nextInt());
    }
    for (i = 0; i < n; i++) {
        System.out.println(arreglo.get(i));
    }
}

public static void vectorfijo(int n) {
    int[] números = new int[n];
    for (i = 0; i < n; i++) {
        números[i] = r.nextInt();
    }
    for (i = 0; i < n; i++) {
        System.out.println(números[i]);
    }
}

public static void matriz(int n) {
    int[][] matrix = new int[n][n];
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            matrix[i][j] = r.nextInt(50);
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            System.out.print(matrix[i][j] + " ");
        }
    }
}

```

```

        }
        System.out.println("");
    }
}

public static void vectorsinrepeticion(int n) {
    HashSet<Integer> sinrep = new HashSet<>();
    for (i = 0; i < n; i++) {
        sinrep.add(r.nextInt(50));
    }
    Iterator it = sinrep.iterator();
    while (it.hasNext()) {
        System.out.println(it.next());
    }
}

public static void Diccionario(int n) {
    TreeMap<String, String> dicc = new TreeMap<>();
    dicc.put("Programar", "Utilizacion de codigo para
ejecutar un programa");
    dicc.put("Futbol", "Deporte con una pelota y
arcos");
    dicc.put("helado", "crema helada");
    dicc.put("Sargento mayor Johnson", "Practicar con
palos y piedras");
    for (String concepto : dicc.keySet()) {
        String key = concepto;
        String value = dicc.get(concepto);
        System.out.println(key + "->" + value);
    }
}

public static void pila(int n) {
    Stack<Integer> mipila = new Stack<>();
    for (i = 0; i < n; i++) {
        mipila.push(r.nextInt(50));
    }
    while (!mipila.isEmpty()) {
        System.out.println(mipila.pop());
    }
}

public static void cola(int n) {
    Queue<Integer> micola = new LinkedList<>();
    for (i = 0; i < n; i++) {
        micola.offer(r.nextInt(50));
    }
    while (!micola.isEmpty()) {
        System.out.println(micola.poll());
    }
}

```

```

    }
}

```

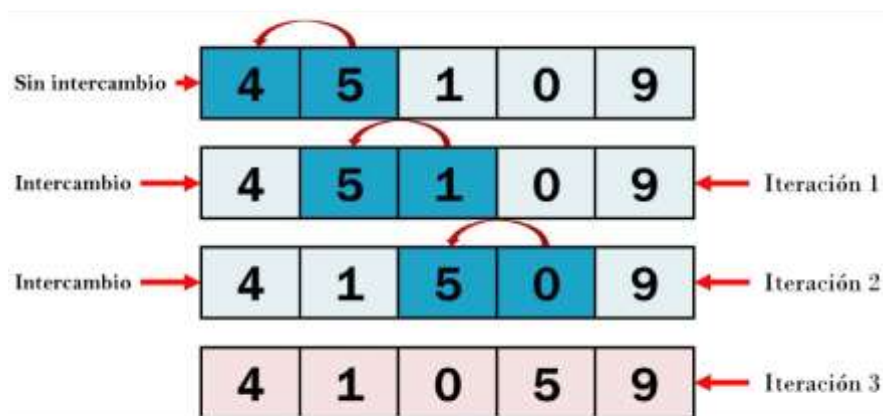
Código tomado del libro guía del programador competitivo.

Algoritmos de ordenamiento

Los algoritmos de ordenamientos como su nombre lo dicen su prioridad es ordenar los datos de una estructura de datos, estos se usan mayor mente para ordenar información de mayor a menor o viceversa con el fin de optimizar la búsqueda de los mismos por parte del usuario, existen varios tipos de ordenamiento, unos más eficaces que otros y más completos.

Ordenamiento burbuja o Bubble Sort

Bubble Sort



Guia del programador competitivo

Tomado de: Guía del programador competitivo

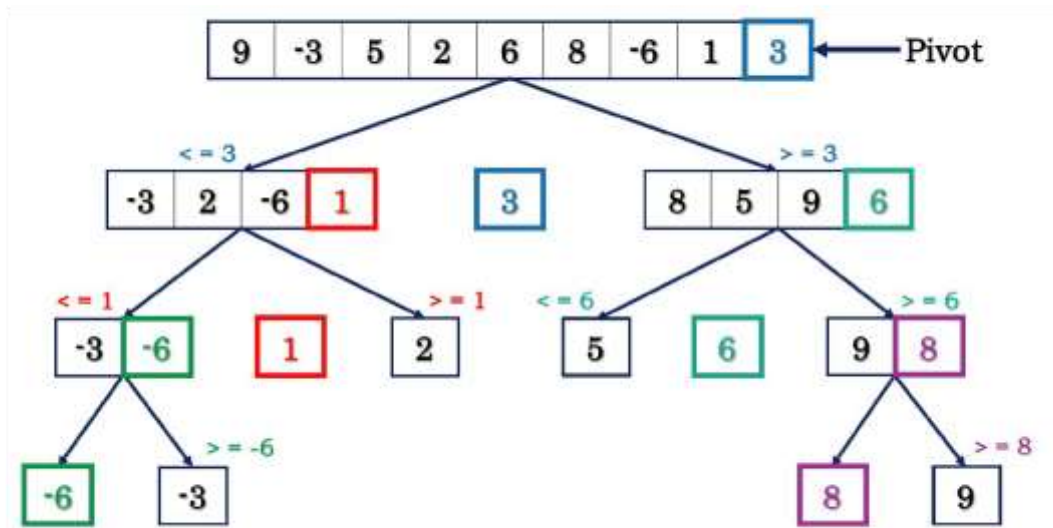
Este algoritmo es uno de los más sencillos y posiblemente más conocido de los ordenamientos ya que es el primero que se ve y se explica para dar apertura a la serie de algoritmos de ordenamientos, a diferencia de los demás este puede ser muy demorado por la cantidad de recursiones que tiene que hacer para llevar acabo su función.

```
import java.util.Arrays;
```

```
public class RecursiveBubbleSort {  
  
    static void bubbleSort(int arr[], int n) {  
        if (n == 1) {  
            return;  
        }  
  
        for (int i = 0; i < n - 1; i++) {  
            if (arr[i] > arr[i + 1])  
                int temp = arr[i];  
                arr[i] = arr[i + 1];  
                arr[i + 1] = temp;  
        }  
        bubbleSort(arr, n - 1);  
    }  
  
    public static void main(String[] args) {  
        int arr[] = {64, 34, 25, 12, 22, 11, 90};  
        bubbleSort(arr, arr.length);  
        System.out.println("Array ordenado : ");  
        System.out.println(Arrays.toString(arr));  
    }  
}
```


Quick Sort

Quicksort



Guía del programador competitivo

Tomado de: Guía del programador competitivo

El algoritmo de Quick Sort se basa en los algoritmos de divide y conquiste, para hacer esta división, se toma un costo del array como pivote, y se mueven todos los recursos menores que este pivote a su izquierda, y los más grandes a su derecha. Luego se aplica el mismo procedimiento a todas ambas piezas en las que queda dividido el array. Otra de derecha a izquierda, intentando encontrar un factor menor que el pivote. Una vez que se han encontrado ambos recursos anteriores, se cambian, y se sigue llevando a cabo la averiguación hasta que ambas búsquedas se hallan.

```
public class QuickSort {  
  
    static int partition(int arr[], int low, int high) {  
        int pivot = arr[high];  
        int i = (low - 1);  
        for (int j = low; j < high; j++) {  
            if (arr[j] <= pivot) {  
                i++;  
                int temp = arr[i];  
                arr[i] = arr[j];  
            }  
        }  
        arr[i+1] = pivot;  
        return i+1;  
    }  
}
```

```

        arr[j] = temp;
    }
}

int temp = arr[i + 1];
arr[i + 1] = arr[high];
arr[high] = temp;

return i + 1;
}
static void sort(int arr[], int low, int high) {
    if (low < high) {

        int pi = partition(arr, low, high);
        sort(arr, low, pi - 1);
        sort(arr, pi + 1, high);
    }
}
static void printArray(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n; ++i) {
        System.out.print(arr[i] + " ");
    }
    System.out.println();
}

public static void main(String args[]) {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = arr.length;
    sort(arr, 0, n - 1);
    System.out.println("Array ordenado");
    printArray(arr);
}
}

```

Algoritmos de búsqueda

Los algoritmos de búsqueda son uno de los mas utilizados en conjunto con los algoritmos de ordenamiento, estos mayoritariamente sirven para encontrar cierto dato en un índice indicado, principalmente se trabajan estos algoritmos en las estructuras de datos, se puede hacer un trabajo extremadamente completo entre una estructura un ordenamiento y las búsquedas, de esta manera lograr una excelente calidad de software, hay distintos tipos de búsquedas por lo que unas nos más eficientes en tiempo y otras más completas pero más demoradas.

Búsqueda binaria

0	1	2	3	4	5	6	7	8	9
4	6	10	12	17	25	29	30	41	44

25	29	30	41	44
----	----	----	----	----

25	29
----	----

Buscando el 28

29

Se determino que el elemento no esta haciendo solo 4 comparaciones!

Guia del programador competitivo

Tomado de: Guía del programador competitivo

Si partimos de que los recursos de la matriz permanecen almacenados en orden ascendente, el proceso de averiguación podría describirse en los próximos pasos:

- Se selecciona el componente del centro o alrededor de del centro de la matriz.
- Si el costo a buscar no coincide con el componente seleccionado, y es más grande que él, se continúa la averiguación en la segunda mitad de la matriz. Si, por otro lado, el costo a buscar es menor que el costo del componente seleccionado, la averiguación continúa en la primera mitad de la matriz.
- En los dos casos, se encuentra otra vez el componente central, que corresponde al nuevo intervalo de averiguación, repitiéndose el periodo.
- El proceso se repite hasta que esté el costo buscado, o bien hasta que el intervalo de averiguación sea nulo, lo cual querrá mencionar que el componente buscado NO FIGURA en la matriz.

```

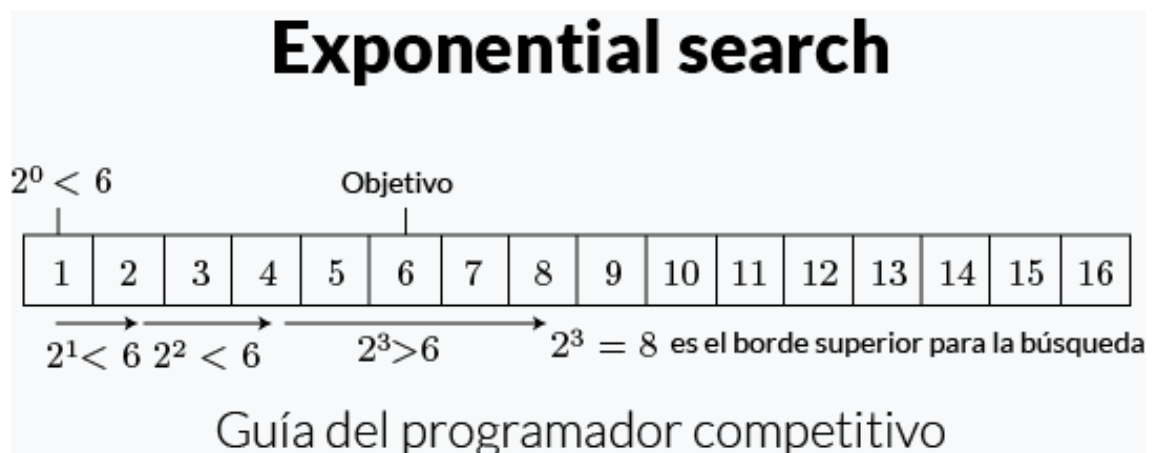
public class BinarySearch {

    static int binarySearch(int arr[], int l, int r, int x){
        if (r >= l) {
            int mid = l + (r - l) / 2;
            if (arr[mid] == x) {
                return mid;
            }
            if (arr[mid] > x) {
                return binarySearch(arr, l, mid - 1, x);
            }
            return binarySearch(arr, mid + 1, r, x);
        }
        return -1;
    }

    public static void main(String args[]) {
        int arr[] = {2, 3, 4, 10, 40};
        int n = arr.length;
        int x = 10;
        int result = binarySearch(arr, 0, n - 1, x);
        if (result == -1) {
            System.out.println("Elemento no presente");
        } else {
            System.out.println("Element encontrado en la
posición " + result);
        }
    }
}

```

Exponential Search



Tomado de: Guía del programador competitivo

Este algoritmo sirve para buscar en listas ordenadas infinitas, Hay varias formas de llevar a cabo esto con el ser más común para decidir un rango en el cual

radica la clave de averiguación y hacer una averiguación binaria en aquel rango. Esto toma $O(\log i)$ donde i es la postura de la clave de averiguación en la lista, si la clave de averiguación está en la lista, o la postura donde debería estar la clave de averiguación, si la clave de averiguación no está en la lista.

```
import java.util.Arrays;

public class ExponentialSearch {
    static int exponentialSearch(int arr[],int n, int x) {
        if (arr[0] == x) {
            return 0;
        }

        int i = 1;
        while (i < n && arr[i] <= x) {
            i = i * 2;
        }

        return Arrays.binarySearch(arr, i / 2,
            Math.min(i, n), x);
    }

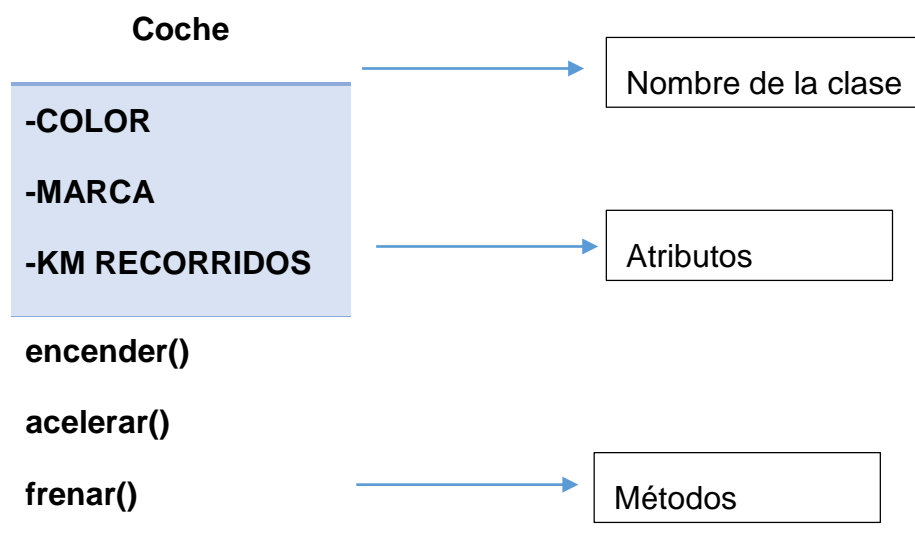
    public static void main(String args[]) {
        int arr[] = {2, 3, 4, 10, 40};
        int x = 10;
        int result = exponentialSearch(arr, arr.length, x);
        System.out.println((result < 0)
            ? "El elemento no está presente en el array"
            : "Elemento encontrado en : "
            + result);
    }
}
```

Programación Orientada a Objetos (POO)

La programación orientada a objetos (POO), es una forma muy peculiar de programar, es mucho más parecido a la forma en la que nos expresamos día a día aplicándolo a diferentes tipos de programación, este tipo de programación

contiene unos aspectos muy importantes como lo serían los objetos, donde los objetos tienen que tener Atributos (CARACTERISTICAS) y sus métodos (ACCIONES), un ejemplo básico sería que un auto es el objeto y sus atributos podrían ser el color, la marca y el kilometraje, y los métodos podrían ser encender, acelerar y frenar, para pensar en cómo sería un objeto es tener en cuenta que puede ser cualquier cosa siempre y cuando cumpla los dos pilares importantes que tengan Atributos y Métodos.

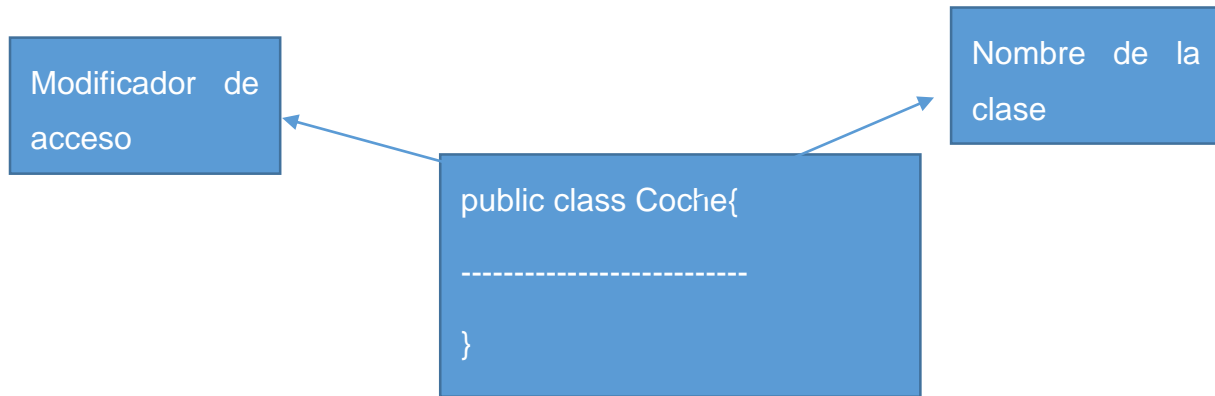
A parte de los objetos en la programación orientada a objetos (POO) también se habla de clases, que simplemente son un conjunto de objetos que comparten muchas características y ayuda a que sea más eficiente y completo al mismo tiempo, utilizando el ejemplo anterior de los coches, un ejemplo más simple para entender cómo funcionan las clases en java sería el siguiente:



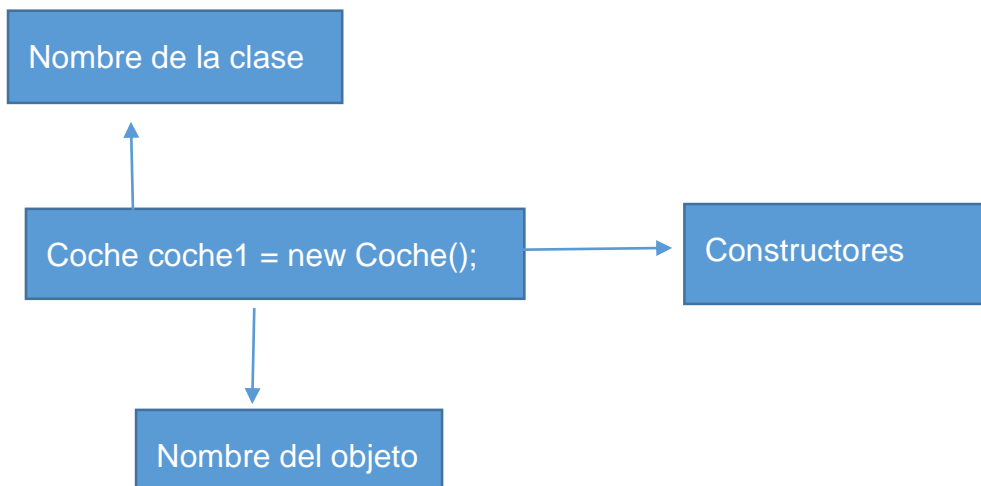
Como se ve en el gráfico una clase tiene los atributos y métodos que tenían el ejemplo de un objeto, ahora muchas veces se pueden preguntar ¿de una clase solo puedo crear un objeto?, y la respuesta es no sencillamente se pueden crear tantos objetos como se quiera, pero siempre y cuando se respeten el tipo de objetos a crear y sus atributos y métodos.

Creación de clase y objetos

Creación de la clase



Creación del objeto:



Para la creación de estas clases y objetos es necesario utilizar el package en donde se almacenará la clase y por ende sus objetos relacionados con el fin de la relación es mucho más sencilla de hacer.

```
public class Coche {

    //Atributos
    String color;
    String marca;
    int km;

    //Metodo
    public static void main(String[] args) {
        Coche coche1 = new Coche();

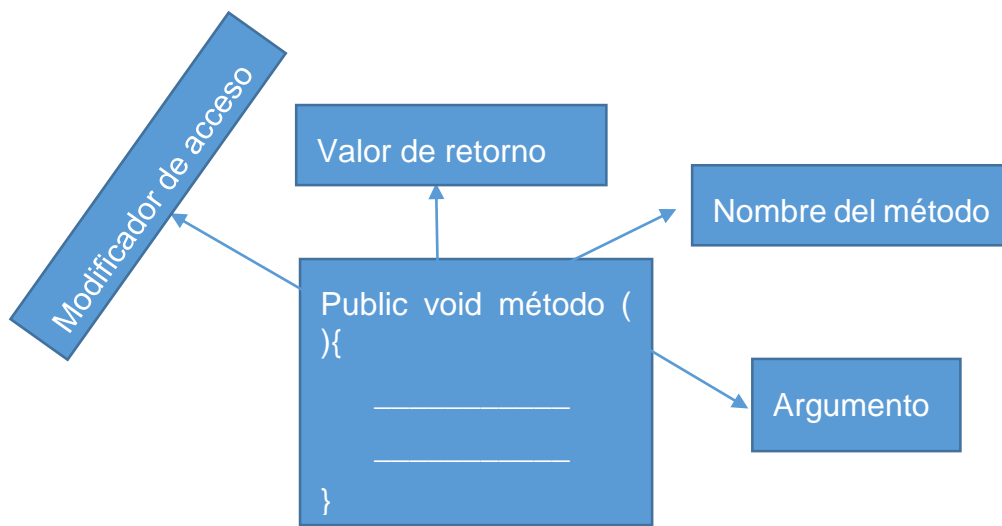
        coche1.color = "Rojo";
        coche1.marca = "Audi";
        coche1.km = 0;

        System.out.println("El color del auto numero 1 es: " +
coche1.color);
        System.out.println("La marca del auto 1 es: " + coche1.marca);
        System.out.println("El kilometraje del auto 1 es: " +
coche1.km);

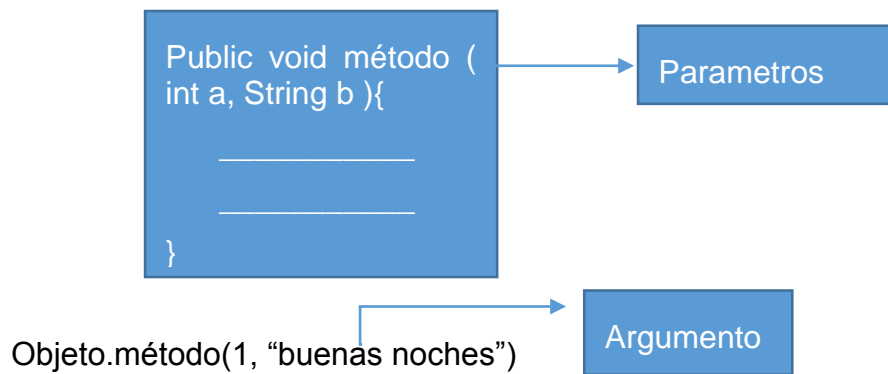
        System.out.println("_____
_____");
        Coche coche2 = new Coche();

        coche2.color = "Blanco";
        coche2.marca = "Mercedes-Benz";
        coche2.km = 20000;
        System.out.println("El color del auto numero 2 es: " +
coche2.color);
        System.out.println("La marca del auto 2 es: " + coche2.marca);
        System.out.println("El kilometraje del auto 2 es: " +
coche2.km);
    }

}
```

Métodos y parámetros:



Un método como se ha hablado es la acción que realizan los objetos, y un parámetro es una declaración que se hace dentro de la declaración del método, y un argumento son los datos que van en los parámetros creados, estos argumentos van en la invocación del método.

Código de ejemplo:

CLASE:

```
package Librito;

import javax.swing.JOptionPane;

public class Operacion {

    //Atributos
    int numerol;
    int numero2;
```

```

int suma;
int resta;
int multiplicacion;
int division;

//Metodos

//Metodos para pedirle al usuario que digite los numeros
public void leerNumeros() {
    numero1
Integer.parseInt(JOptionPane.showInputDialog("Digite el primer
numero: "));
    numero2
Integer.parseInt(JOptionPane.showInputDialog("Digite el segundo
numero: "));
}
//Metodo para sumar
public void sumar() {
    suma = numero1 + numero2;
}
//Metodo para restar
public void resta() {
    resta = numero1 - numero2;
}
//Metodo para multiplicar
public void multiplicar() {
    multiplicacion = numero1 * numero2;
}
//Metodo para dividir
public void dividir() {
    division = numero1 / numero2;
}

public void mostrarResultado(){
    System.out.println("La suma es: "+ suma);
    System.out.println("La resta es: "+ resta);
    System.out.println("La      multiplicacion      es:      "+
multiplicacion);
    System.out.println("La divison es: "+ division);
}

```

```
    }  
}
```

MAIN:

```
package Librito;  
  
public class Main {  
    public static void main(String[] args) {  
        Operacion op = new Operacion();  
  
        op.leerNumeros();  
        op.sumar();  
        op.resta();  
        op.multiplicar();  
        op.dividir();  
        op.mostrarResultado();  
    }  
}
```

Constructores

En Java es un método particular en una clase, que se denomina automáticamente cada vez que se crea un objeto de dicha clase, estos constructores tienen el mismo nombre de la clase, no permite devolver ningún valor y siempre se debe declarar como público.

CLASE:

```
package Constructor;  
  
public class Persona {  
    //Atributos  
  
    String nombre;  
  
    int edad;
```

```

//Metodos

//constructor

public Persona(String nombre, int edad) {

    this.nombre = nombre;

    this.edad = edad;

}

public void mostrarDatos() {

    System.out.println("El nombre es: " + nombre);

    System.out.println("La edad es: " + edad);

}

}

```

MAIN:

```

package Constructor;

public class Main {

    public static void main(String[] args) {

        Persona personal = new Persona("Julian", 20);

        personal.mostrarDatos();

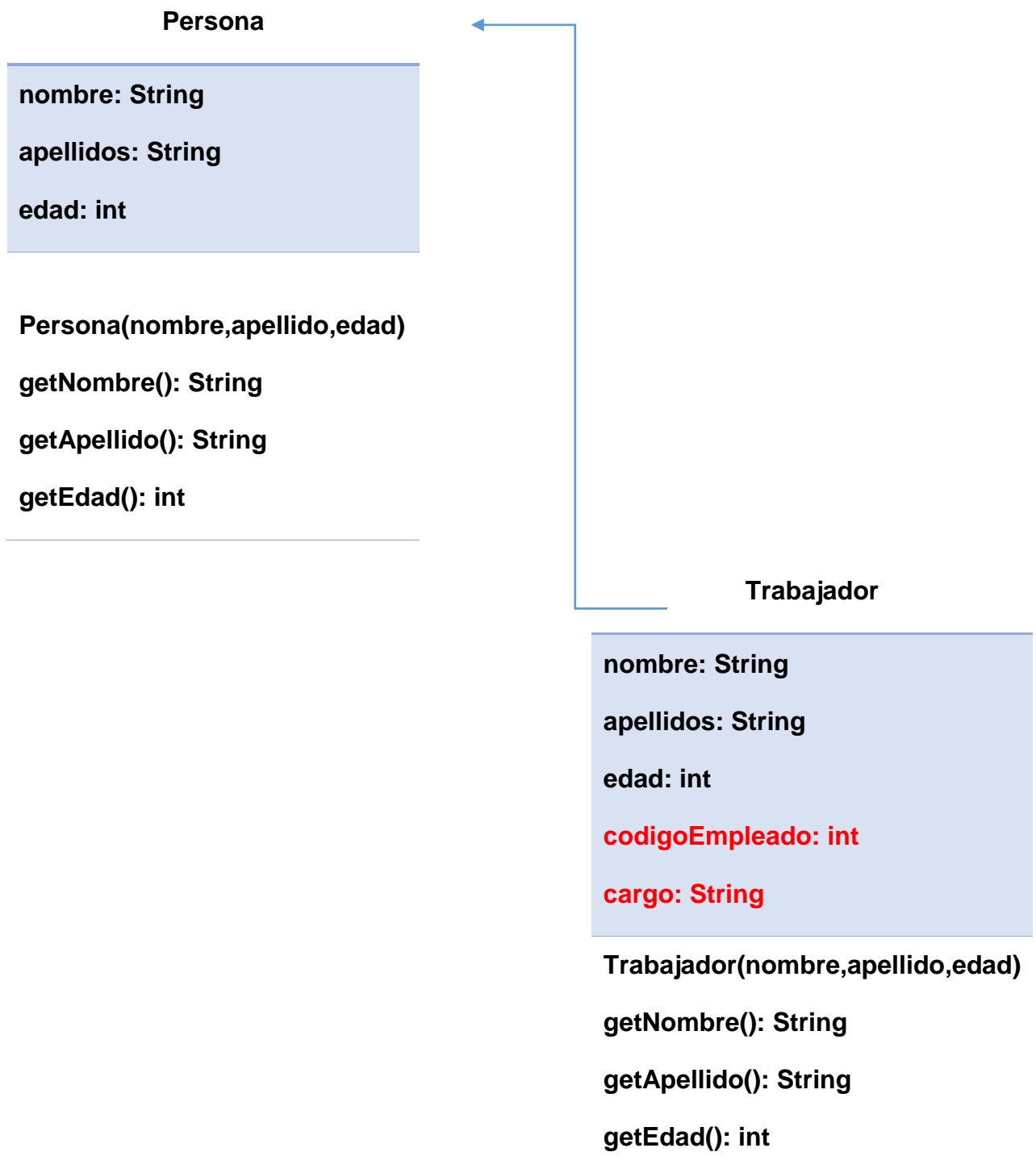
    }

}

```

Herencias

La herencia se puede entender como a manera de reutilizar parte del software o hacer una recursión en donde se crea una nueva clase que obtendrá los miembros de otra que ya se haya creado y utilizado.



Codigo:

CLASE PERSONA:

```
package Herencia;
```

```
public class Persona {  
    private String nombre;  
    private String apellido;  
    private int edad;  
    /*Podriamos utilizar elprotected cumple la funcion de que los  
datos  
    pueden ser privados pero tambien los elementos de la misma  
    clase los pueden utilizar, pero en este caso utilizaremos los  
getters*/
```

```
    public Persona(String nombre,String apellido, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
        this.apellido=apellido;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
    public String getApellido() {  
        return apellido;  
    }  
    public int getEdad() {
```

```
        return edad;
    }
}
```

CLASE TRABAJADOR:

```
package Herencia;

public class Trabajador extends Persona {

    private String codigoTrabajador;
    private String cargo;

    //Constructor especial
    public Trabajador(String nombre, String apellido, int edad,
String codigo, String cargo) {
        super(nombre, apellido, edad);
        this.codigoTrabajador = codigoTrabajador;
        this.cargo = cargo;
    }

    public void mostrarDatos() {
        System.out.println("El nombre del trabajador es: " +
getNombre());

        System.out.println("El apellido del trabajador es: " +
getApellido());

        System.out.println("La edad del trabajador es: " + getEdad());

        System.out.println("El código del trabajador es: " +
codigoTrabajador);
    }
}
```

```
        System.out.println("El cargo del trabajador es: " +
cargo);
    }
}
```

CLASE PRINCIPAL(MAIN):

```
package Herencia;

public class Principal {

    public static void main(String[] args) {

        Trabajador trabajador1 = new Trabajador("Stiven",
"Silva", 20, "80806", "Desarrollador");

        trabajador1.mostrarDatos();

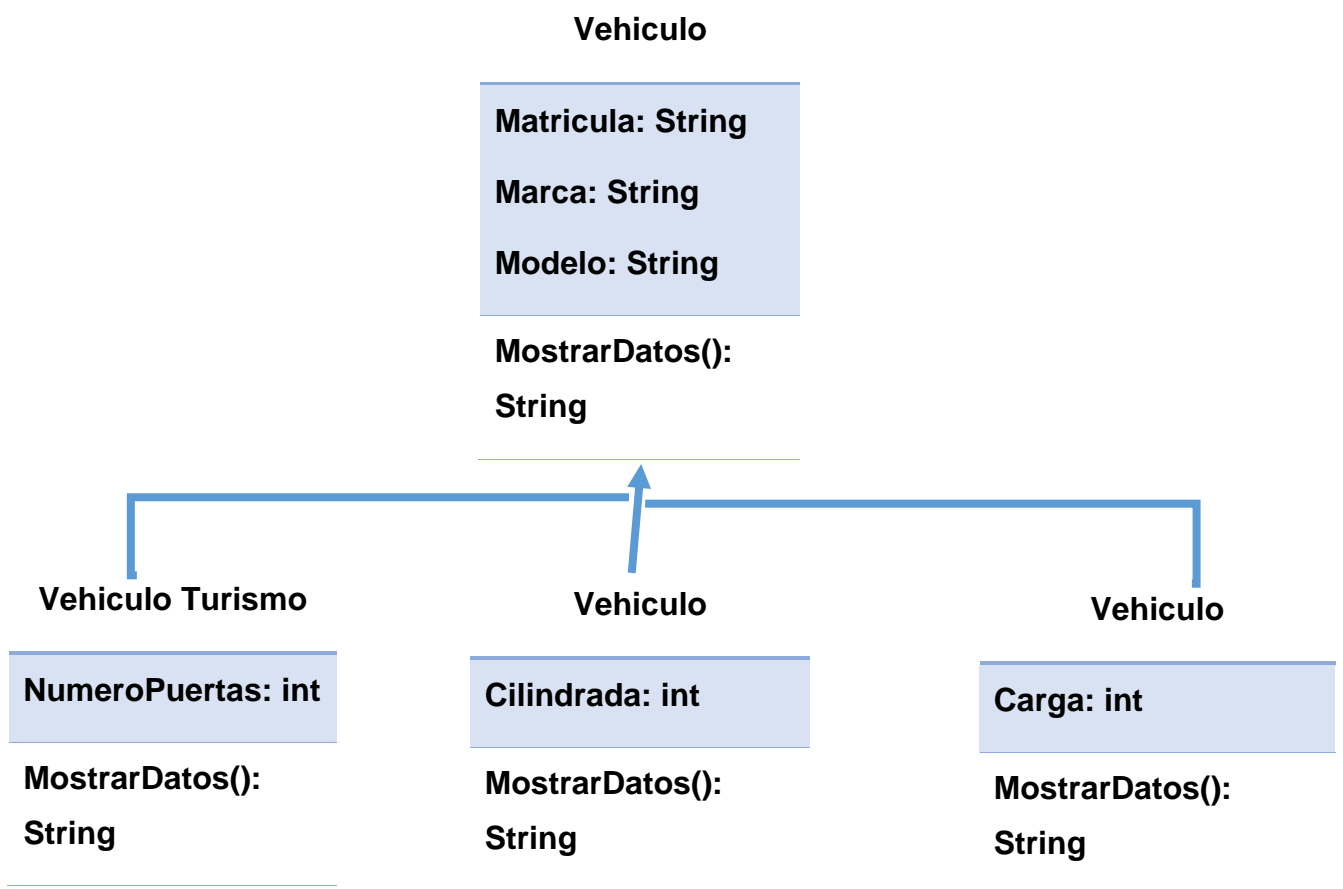
    }

}
```

Polimorfismo

Se entiende como polimorfismo a una habilidad que tiene una función, variables, etc. Pueden poseer distintas formas y propiedades, entonces para entender mejor se puede decir que un objeto que comparte sus propiedades, podría tener distintos significados y elementos.

Para una mejor explicación tomemos el ejemplo de los vehículos.



Código:

CLASE VEHICULO

```
package Polimorfismo;
```

```
public class Vehiculo {

    protected String matricula;

    protected String marca;

    protected String modelo;

    public Vehiculo(String matricula, String marca, String
modelo) {

        this.matricula = matricula;

        this.marca = marca;

        this.modelo = modelo;

    }

    public String getMatricula() {

        return matricula;

    }

    public String getMarca() {

        return marca;

    }

    public String getModelo() {

        return modelo;

    }

}
```

```

        public String mostrarDatos() {
            return "Matricula: " + matricula + "\nMarca: " + marca +
"\nModelo: " + modelo;
        }
    }
}

```

CLASE VEHICULO_DEPORTIVO

```
package Polimorfismo;
```

```

public class Vehiculo_Deportivo extends Vehiculo {

    private int cilindrada;

    public Vehiculo_Deportivo(int cilindrada, String matricula,
String marca, String modelo) {

        super(matricula, marca, modelo);

        this.cilindrada = cilindrada;
    }

    public int getCilindrada(){
        return cilindrada;
    }

    @Override
    public String mostrarDatos() {
        return "Matricula: "+matricula +"\nMarca:
"+marca+"\nModelo: "+modelo+
"\nCilindrada: "+cilindrada;
    }
}

```

CLASE VEHICULO_TURISMO

```
package Polimorfismo;

public class Vehiculo_Turismo extends Vehiculo{

    private int nPuertas;

    public Vehiculo_Turismo (int nPuertas, String matricula,
String marca, String modelo){

        super(matricula, marca, modelo);

        this.nPuertas=nPuertas;

    }

    public int getnPuestas(){

        return nPuertas;

    }

    @Override

    public String mostrarDatos() {

        return "Matricula: "+matricula +"\nMarca: "+marca+"\nModelo: "+modelo+

            "\nNumero de puertas: "+nPuestas;

    }

}
```

CLASE VEHICULO_FURGONETA

```
package Polimorfismo;

public class Vehiculo_Furgoneta extends Vehiculo {

    private int carga;

    public Vehiculo_Furgoneta(int carga, String matricula, String
marca, String modelo) {

        super(matricula, marca, modelo);

        this.carga = carga;

    }

    public int getCarga() {

        return carga;

    }

    @Override

    public String mostrarDatos() {

        return "Matricula: " + matricula + "\nMarca: "
+ marca + "\nModelo: " + modelo +
        "\nCarga: " + carga;

    }

}
```

CLASE PRINCIPAL(MAIN)

```
package Polimorfismo;

public class Principal {

    public static void main(String[] args) {

        Vehiculo carritos[] = new Vehiculo[4];

        carritos[0] = new Vehiculo("EWH925", "Renault", "12
break");

        carritos[1] = new Vehiculo_Turismo(4, "FJH763",
"Chevrolet", "Sail");

        carritos[2] = new Vehiculo_Deportivo(670, "MKH424",
"Chevrolet", "Camaro");

        carritos[3] = new Vehiculo_Furgoneta(200, "HGS450",
"Audi", "Q8");

        for (Vehiculo carrito : carritos) {

            System.out.println(carrito.mostrarDatos());

            System.out.println("");

        }

    }

}
```

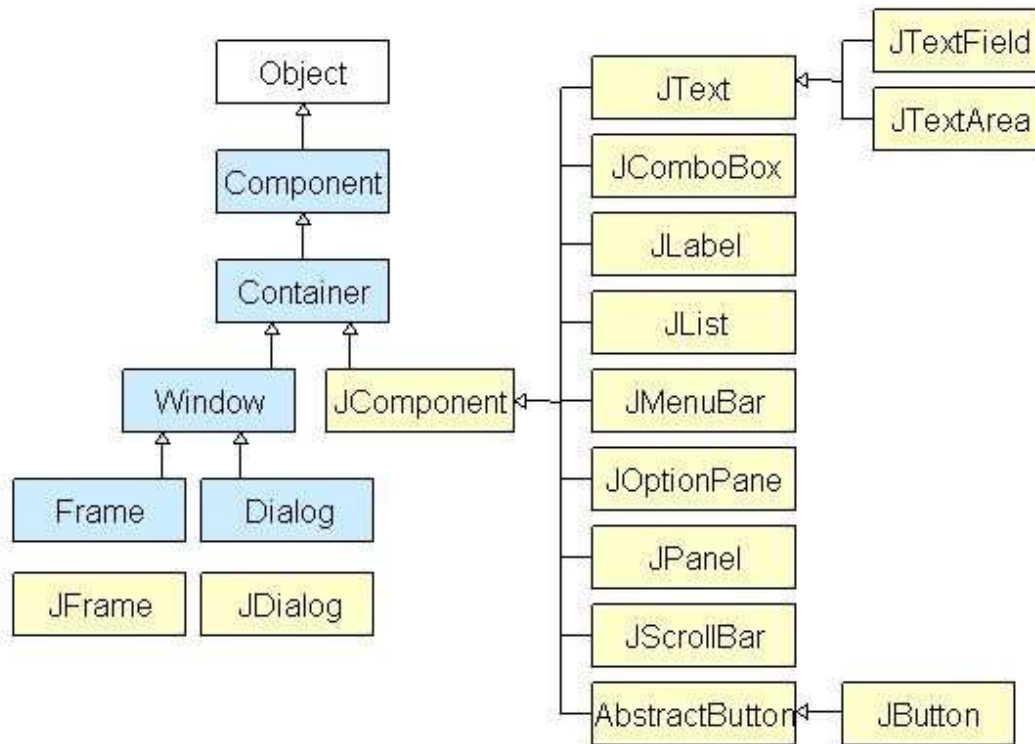
Programación con interfaces gráficas

¿Qué es?

Una interfaz gráfica es un entorno en donde interactúa un usuario con el software que ha adquirido o se le ha realizado, se conoce además como GUI (del inglés graphical user interface), usa un grupo de imágenes y objetos gráficos para representar la información y ocupaciones accesibles en la interfaz.

¿Cómo crear una interfaz?

Para crear una interfaz debemos saber que para la utilización y creación de dichos elementos debemos usar la librería “swing”, la librería swing es un conjunto de interfaces gracias de usuario(GUI), aunque esta es algo mas nueva, presenta una relación con otra librería mas antigua de interfaces como lo es la AWT.



Tomado de: www.guru99.es/java-swing-gui

Dentro de swing podemos encontrar diferentes componentes que lo completan y hacen que podamos desarrollar de manera sencilla y rápida, dichos componentes se dividen de la siguiente manera, en el objeto hay componentes, dentro de los componentes se encuentran los contenedores, así mismo en los contenedores hay ventanas y los conocidos JComponent's, en las ventanas encontramos los Frame y los Dialog que se usan para empezar a crear la interfaz y en los JComponent's están todos los objetos que podemos utilizar para poder darle forma y diseño a nuestra interfaz, podemos encontrar los JText, JComboBox, JLabel, JList, JMenuBar, JOptionPane, JPanel, JScrollBar, AbstractButton, en donde los JText contienen los JTextField y los JTextArea y en el AbstractButton los JButton's.

¿Qué se entiende como clase contenedor?

La clase contenedor como su nombre lo indica contiene otras clases que complementan el GUI y sirven para darle diseño, dentro de él están las siguientes clases:

1. Marco: En donde se puede colocar el título del GUI.
2. Diálogo: Este es una ventana emergente en donde se van a mostrar mensajes según lo indicado por el desarrollador.
3. Panel: Este es el encargado de organizar los componentes que estén dentro de una ventana.

Un ejemplo de un GUI en donde se puede observar lo que es una ventana, los objetos y sus componentes sería el siguiente:

```
import javax.swing.*;

public class Interfaces {

    public static void main(String args[]) {

        JFrame frame = new JFrame("Mi primera GUI");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setSize(300, 300);

        JButton button1 = new JButton("Presionar");

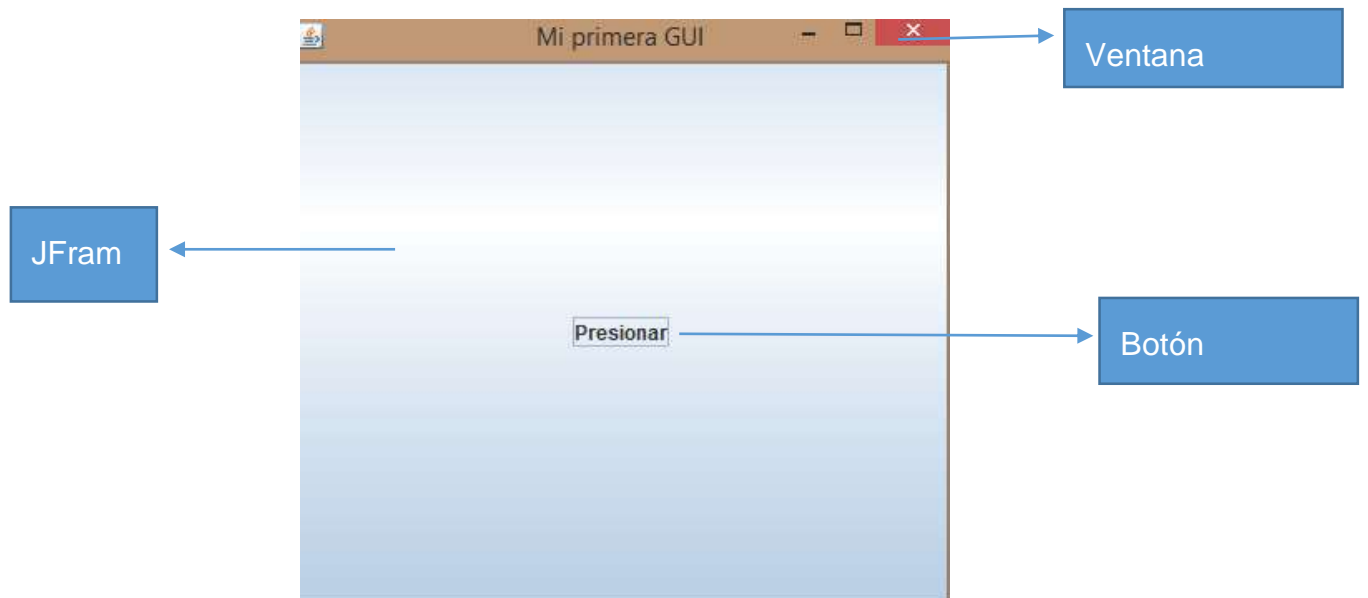
        frame.getContentPane().add(button1);

        frame.setVisible(true);

    }

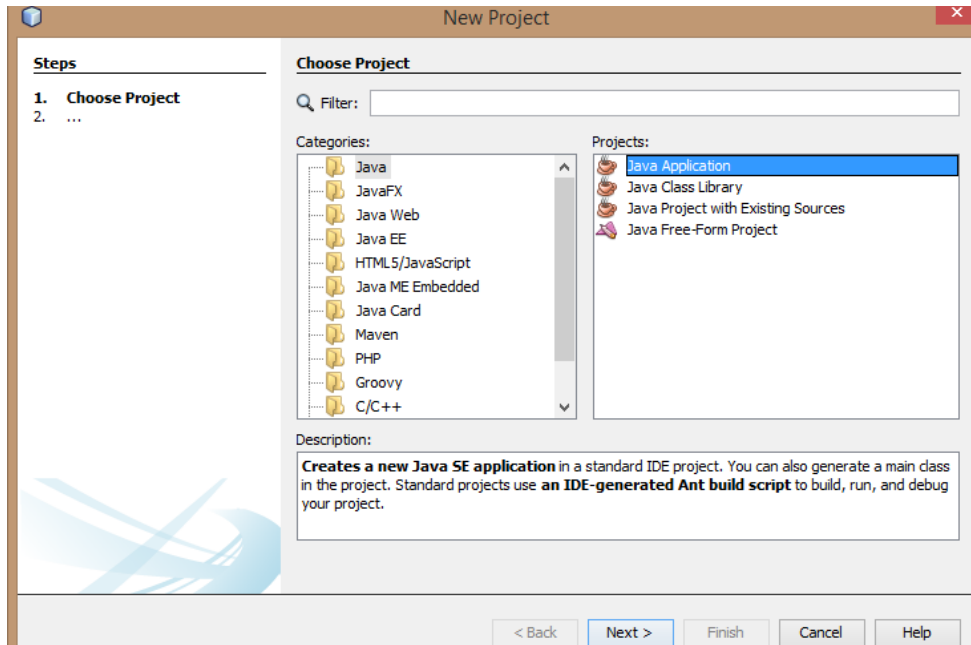
}
```

Donde el resultado es el siguiente:

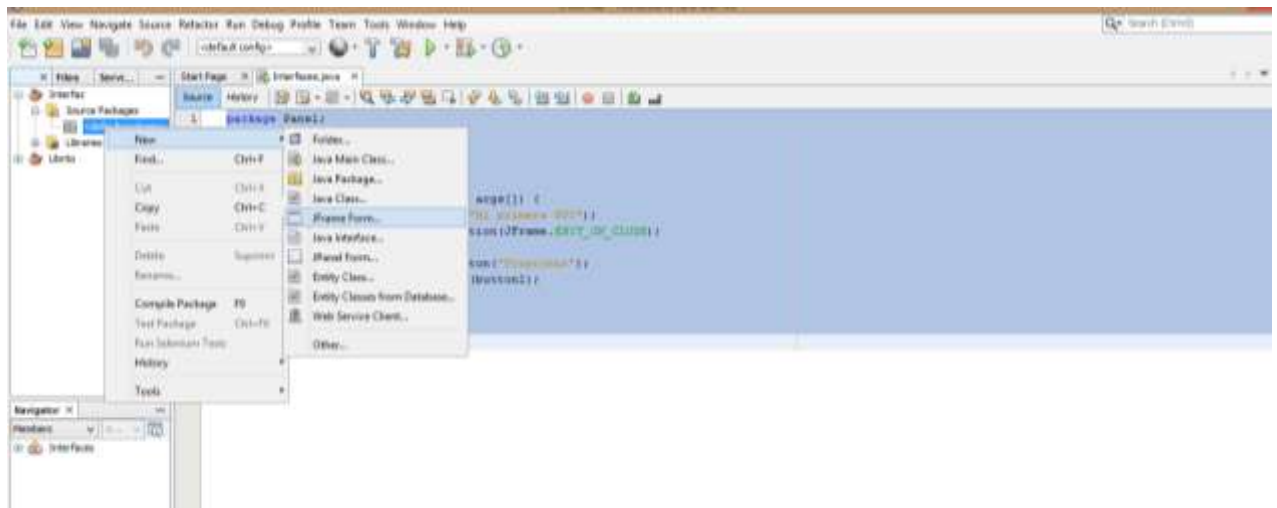


A parte de la creación de lo GUI por medio de código, también se podría crear físicamente, en este caso utilizaremos el IDE NetBeans 8.2 para crearlo.

Primero creamos un proyecto, seleccionamos el Java Application y le colocamos el nombre:



Luego en el Source Package, el en default package le damos clic derecho y new y seleccionamos el JFrame Form y colocamos el nombre que queramos.



A penas hayamos realizado los pasos anteriores se nos abrirá todas las opciones para empezar a crear nuestra interfaz.

