Introducción a la programación con Java

Salvador Uriel Aguirre Andrade, Miguel Aguilar Hilario
Grupo GCCyC, FES Acatlán, UNAM

10 de octubre de 2019

Licencia

Este documento está creado con fines educativos. La información contenida está sometida a cambios y a revisiones constantes, por lo que se sugiere no imprimirlo.

Puedes compartir el documento con quien desees; sin embargo debes respetar la autoría original. Puedes citar el material y considerarlo como referencias en tus proyectos.

Índice general

Introducción **3**
0.1. ¿Java? . . . 3
0.2. Entornos de desarrollo . . . 3
0.3. Compilar y Correr el código . . . 20

1. Bases **28**
1.1. Programación . . . 28
1.2. Paradigmas de programación . . . 28
1.3. Paradigma Orientado a Objetos (POO). . . . 29
1.4. Empezando a programar en java . . . 29
1.5. Tipos de datos básicos . . . 31

2. Operadores **34**
2.1. Operadores aritméticos . . . 35
2.2. Operadores unarios . . . 36
2.3. Operadores relacionales y los booleanos . . . 38
2.4. Operadores lógicos . . . 39
2.5. Operadores bit a bit . . . 40
2.6. Operadores de desplazamiento de bits o shift . . . 42
2.7. Operadores de asignación . . . 43
2.8. Operador ternario . . . 44

3. Palabras reservadas **45**

4. Secuencias de control básicas **47**

5. Pensando en código **50**
5.1. Pseudocódigo . . . 50
5.2. Usando POO en java . . . 51
5.3. Memoria . . . 53

6. Sección Yeyecoa **55**
6.1. Secuencias de escape 55
6.2. Ahora Yeyecoa 56

7. Expandiendo el conocimiento **60**
7.1. Por diversión 60
7.2. Por profesión 60
7.3. Sugerencias 61

Motivación final **62**

Introducción (en construcción)

0.1. ¿Java?

"Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por *Sun Microsystems*" [**?**]. Se seleccionó **Java** por:

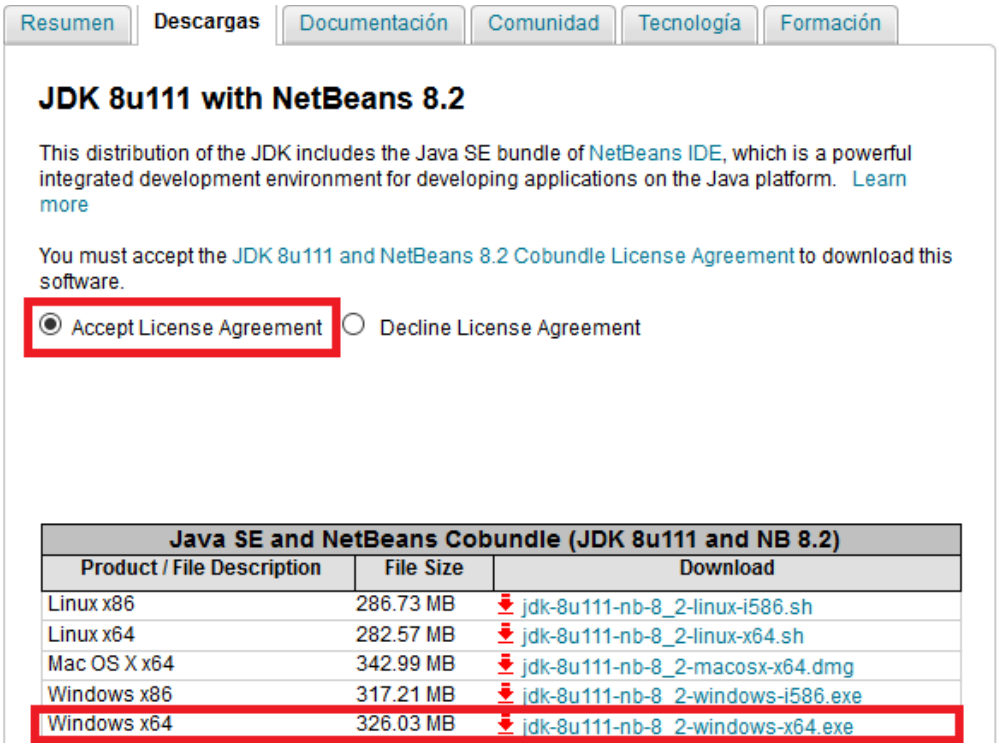
- Su popularidad en aplicaciones de dispositivos inteligentes, bases de datos, juegos, etc.
- La abundancia de fuentes, guías y manuales para aprender Java u otros tópicos que hacen uso del lenguaje.
- Facilidad de uso.

0.2. Entornos de desarrollo

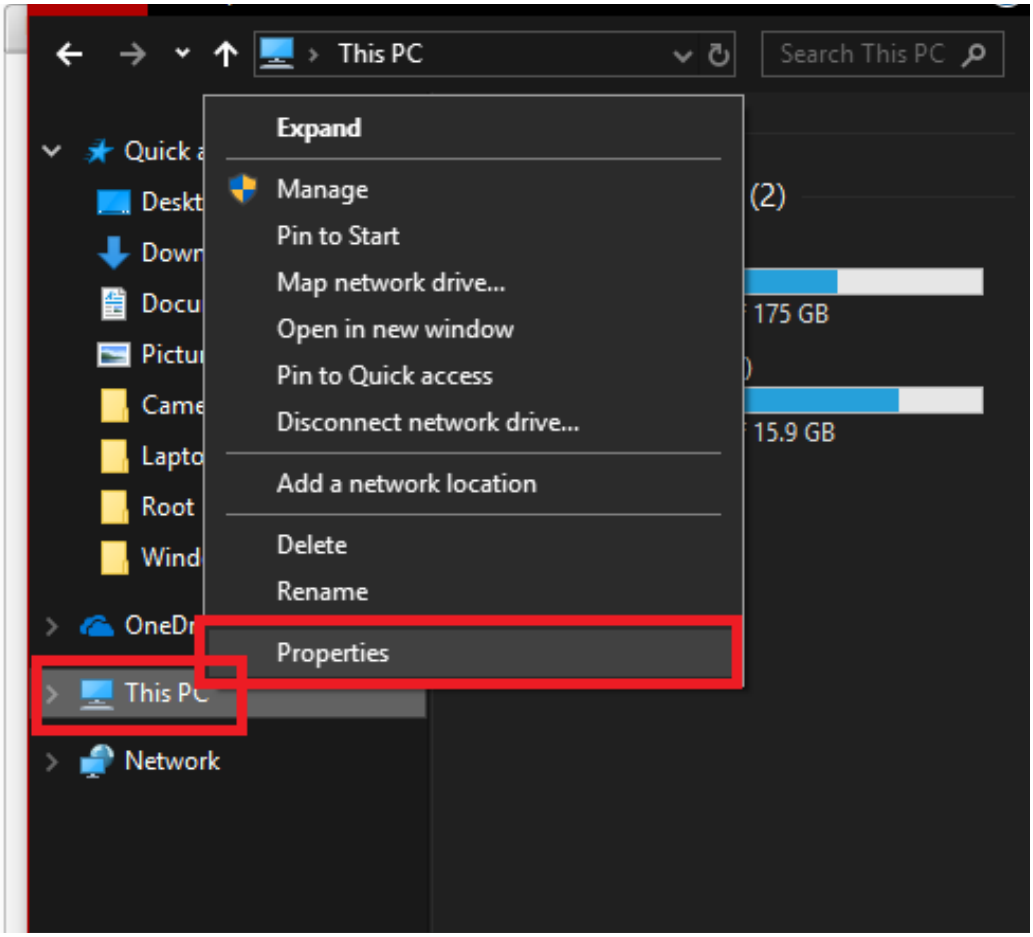
Para empezar a desarrollar nuestros proyectos necesitamos un **IDE** *(Integrated Development Environment)* que en español sería *(Entorno de Desarrollo Integrado)* quiere decir que es más completo y eficiente que un editor de texto.
Hay muchos tipos de IDE, los más conocidos son Netbeans, Eclipse, JGrasp, IntelliJ, IDEA, entre otros.
Para este curso usaremos Netbeans, pero puedes usar cualquier otro IDE que te guste, debido que las lecciones que veremos están enfocadas en el lenguaje JAVA y no depende del IDE.
Si no deseas (o no puedes) instalar nada inclusive puedes usar un compilador en linea, a lo largo del manual se usa el siguiente: https://www.compilejava.net/. Aunque se recomienda familiarizarse con un IDE ya que estos tienen más funcionalidades.
Instalación de Netbeans (Windows)
. Entramos a nuestro navegador y ponemos el siguiente enlace.

https://www.oracle.com/technetwork/es/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html

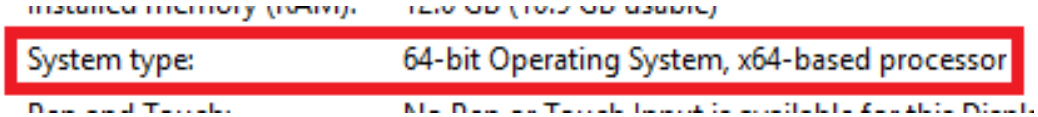
Inmediatamente se nos direccionara a la página de Oracle en la parte de descargas.
Aceptamos el acuerdo de licencia y seleccionamos el sistema operativo que estaremos usando, en mi caso es Windows x64.



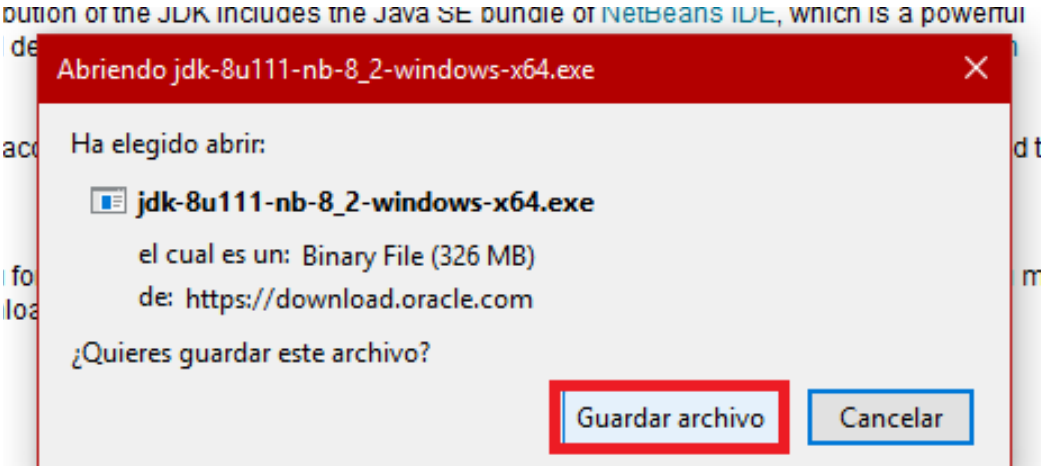
En caso de que no sepas cual Windows usas, si x64 o x86, no te preocupes lo averiguaremos en este momento, teclea la tecla CTRL mas la tecla E al mismo tiempo.
Inmediatamente nos abrirá el Explorador de Archivos
Damos clic derecho al icono de Mi PC y después clic izquierdo a Propiedades.



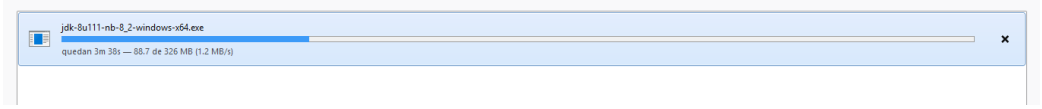
Nos abrirá las Propiedades del sistema, inmediatamente revisaremos el Tipo de Sistema.
allí veremos si estamos usando un sistema de 64bits o de 32bits.



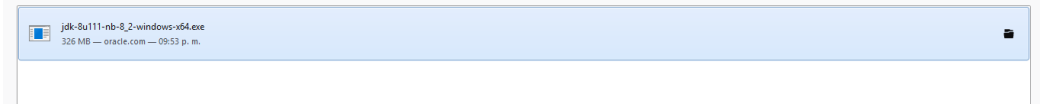
Una vez identificado el sistema que tenemos, procederemos a hacer la descarga el IDE, hacemos clic en el enlace de descarga y se nos abrirá una ventana para guardar el archivo ejecutable, procederemos a dar en Guardar.



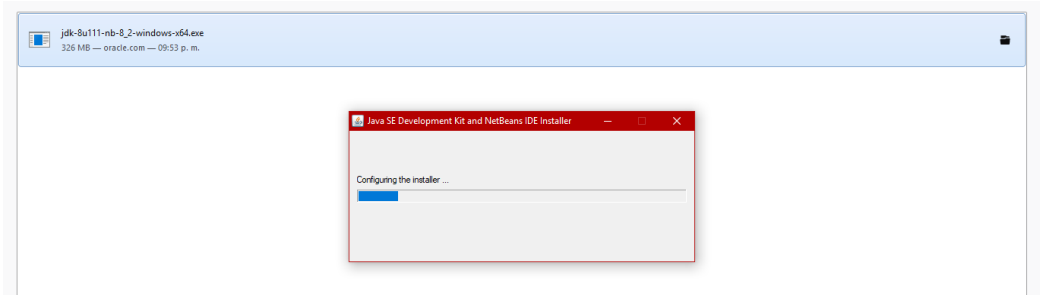
Empezara la descargar,esperamos a que termine de descargarse.



Cuando termine la descarga procederemos a abrir el archivo ejecutable haciendo doble clic.

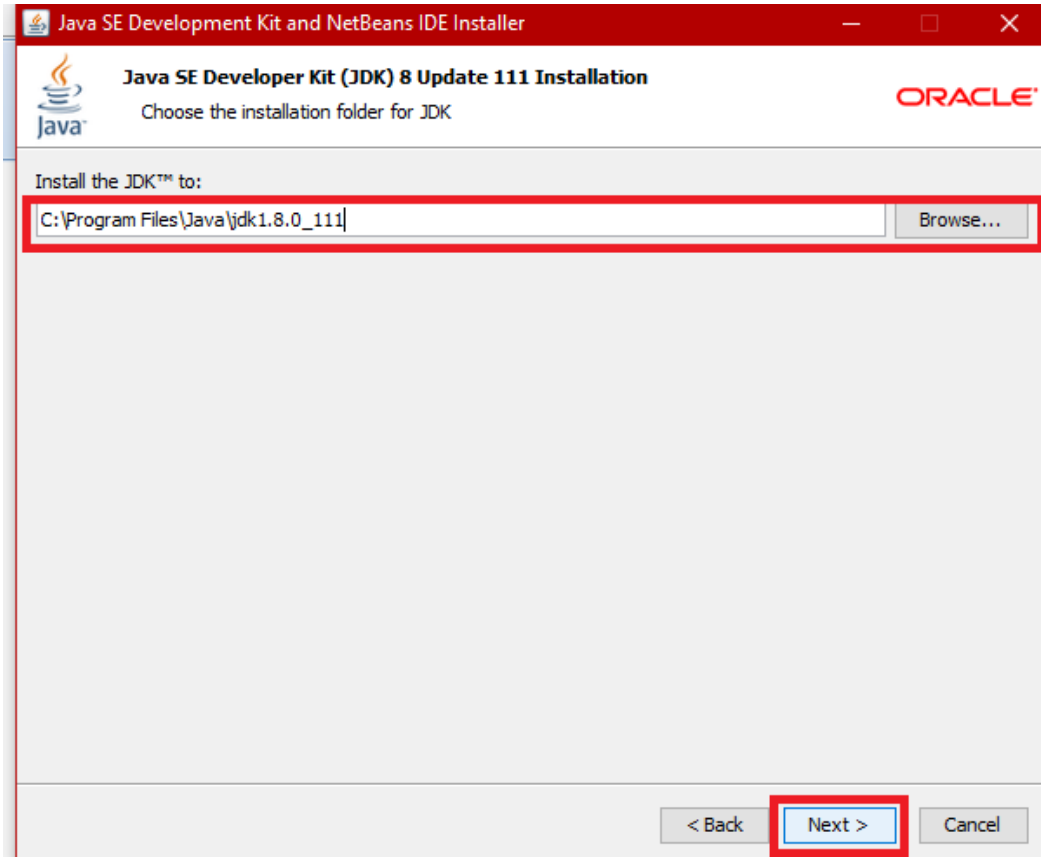


Que a su vez iniciaría el proceso de instalación.

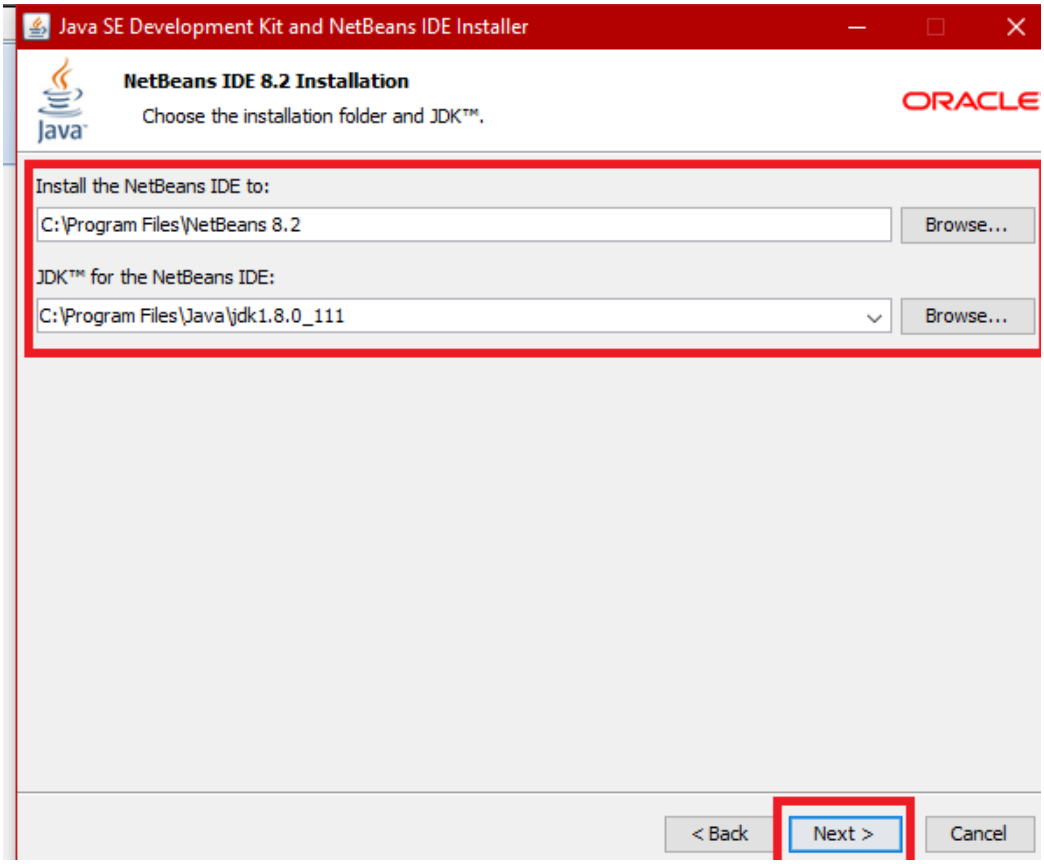


Nos abrirá la ventana para preceder a la instalación del IDE.
y damos clic en el botón de siguiente.

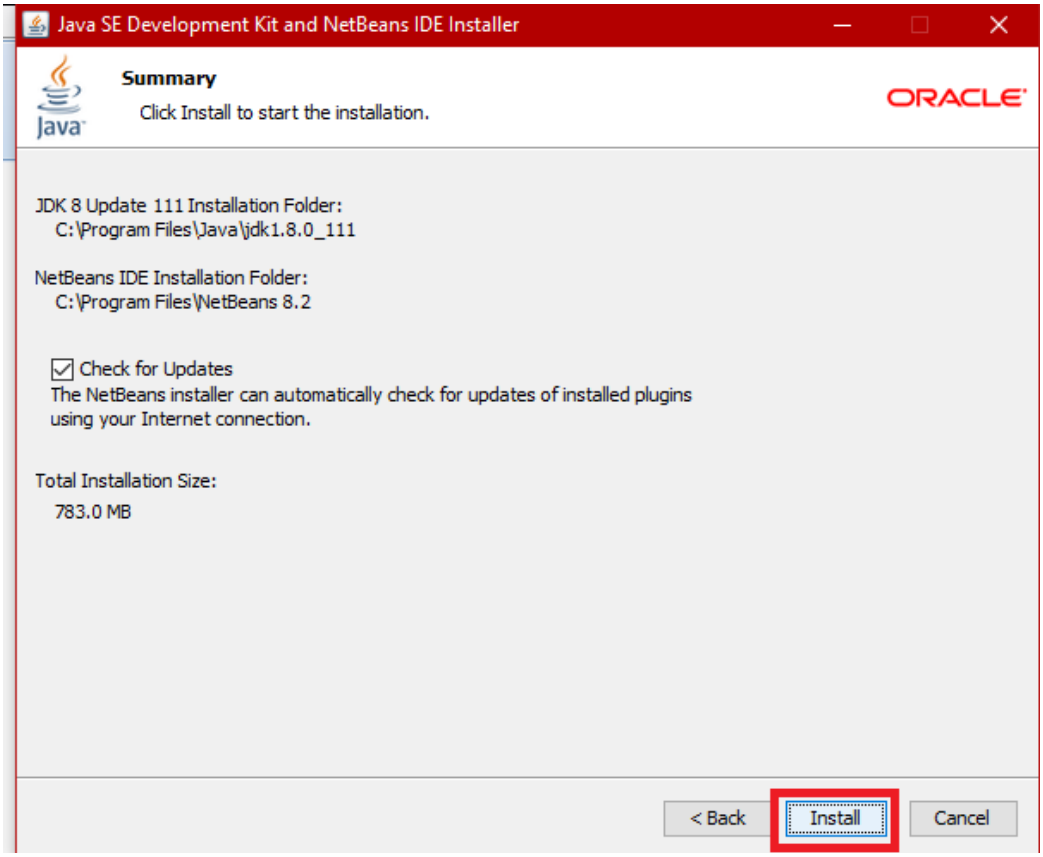
Allí podremos elegir donde queremos que se instale los recursos de Java (JDK) para poder hacer uso de ellos, en este caso se permitirá que se instale en la ruta predeterminada que es el disco C, y daremos clic en siguiente.



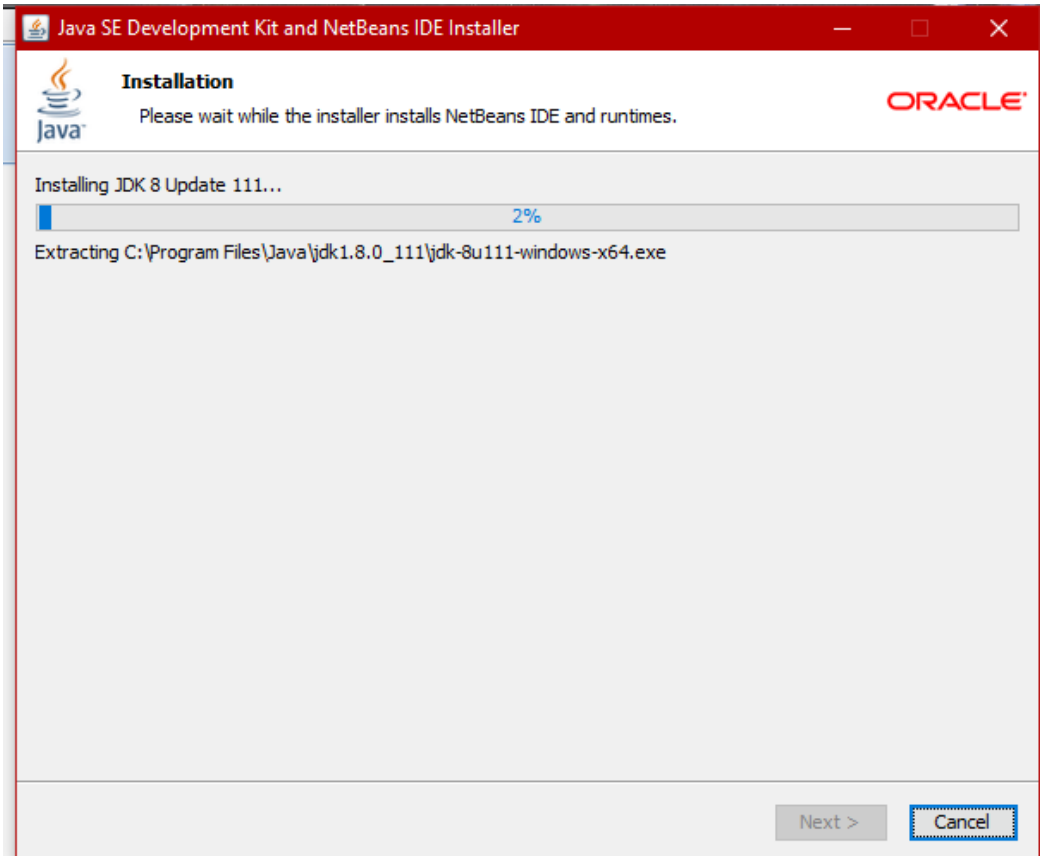
Pues nos pedirá la ruta de instalación para el IDE NetBeans y el JDK para NetBeans, de igual manera dejare la ruta predeterminada en el disco C, y damos clic en siguiente.



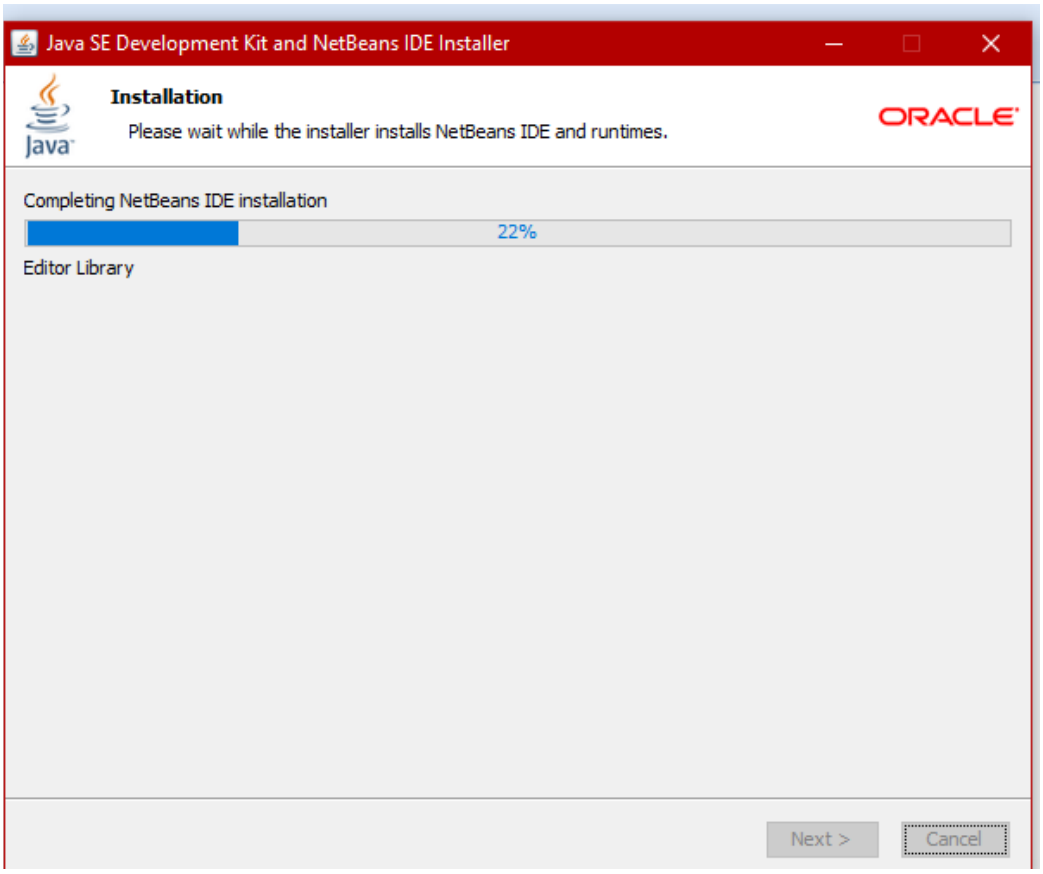
Ya para finalizar nos pediría que si queremos revisar si hay actualizaciones, y también para empezar la instalación. Damos clic en Instalar.



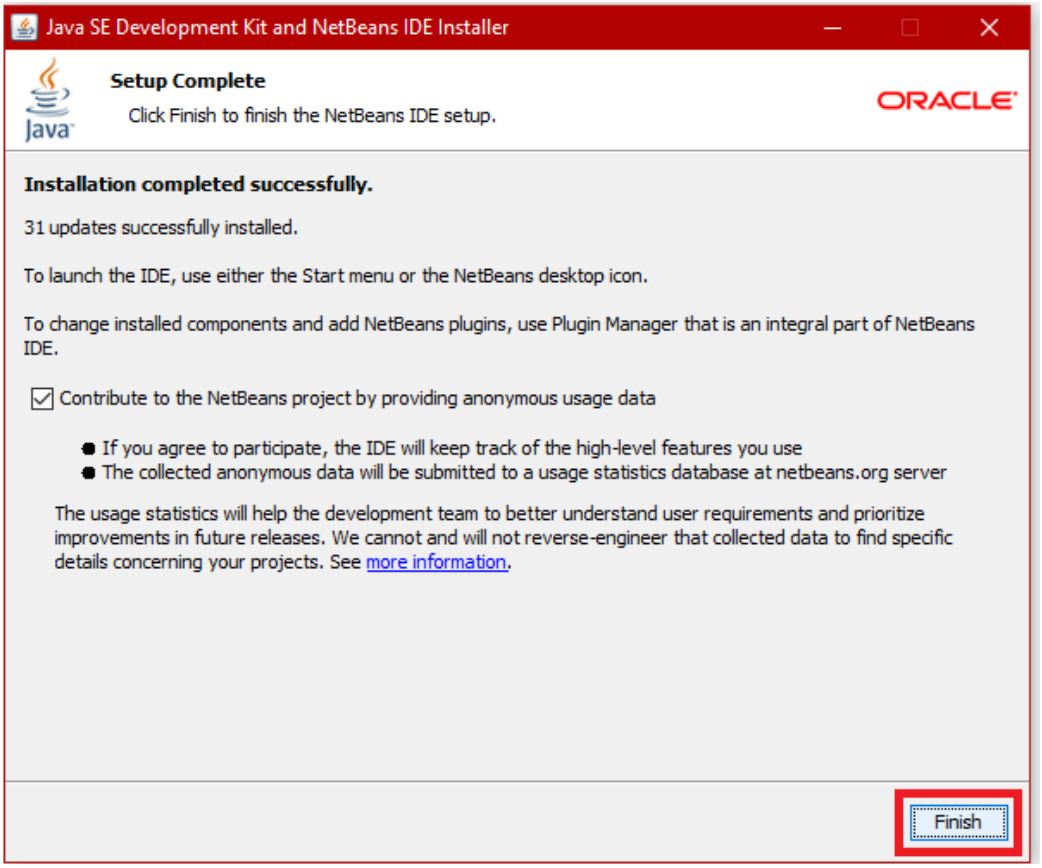
Empezará el proceso de instalación.



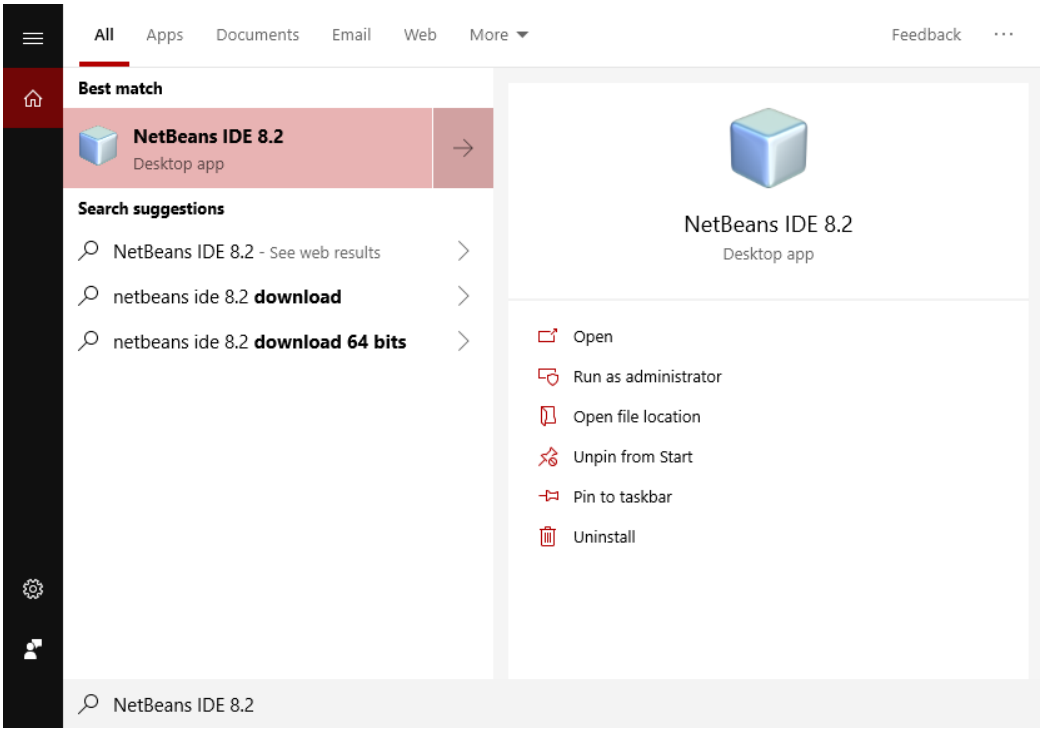
Esperamos a que acabe la instalación.



Una vez terminado el proceso de instalación damos clic en finalizar.



Listo si buscamos en nuestros programas veremos que ya tenemos instalado y listo para usar nuestro IDE NetBeans.

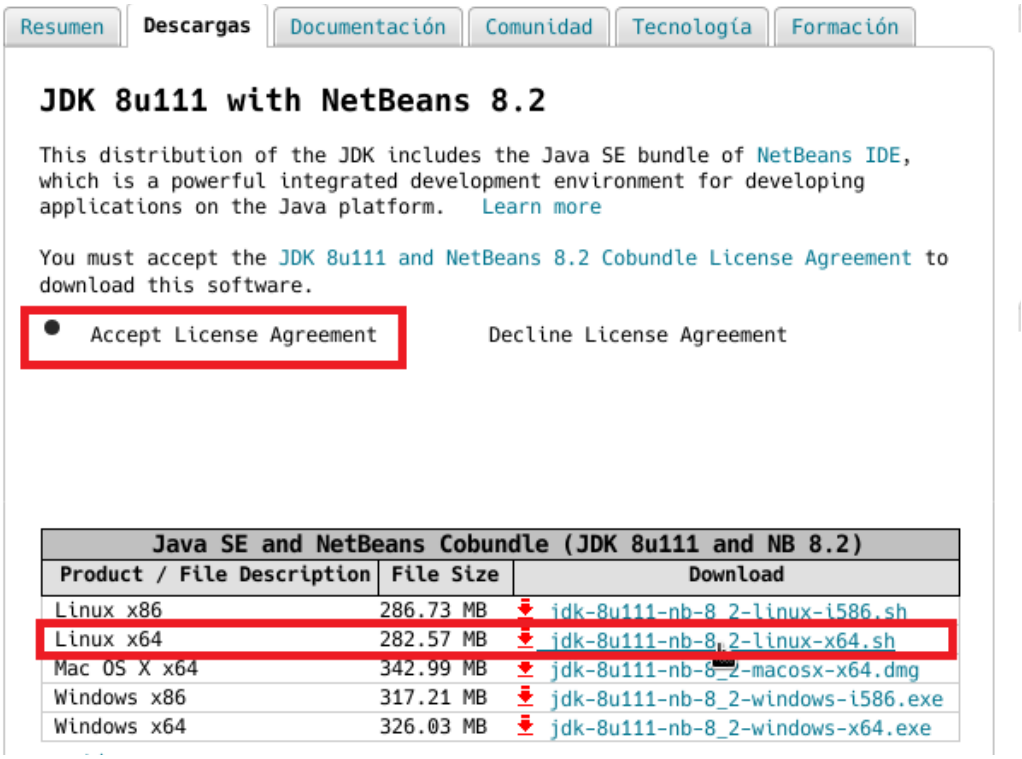


Instalación de Netbeans (Linux)

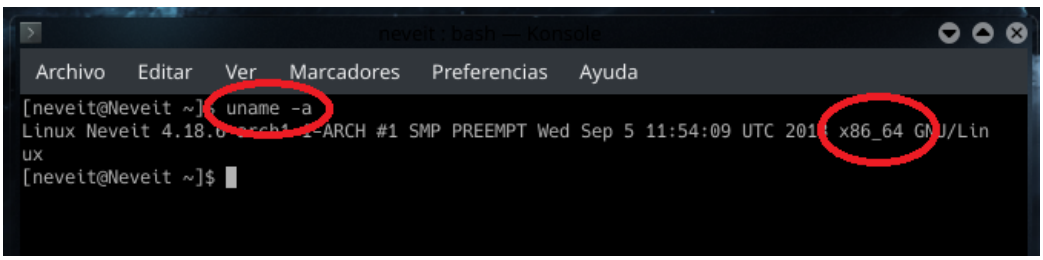
Entramos a nuestro navegador y ponemos el siguiente enlace.

https://www.oracle.com/technetwork/es/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html

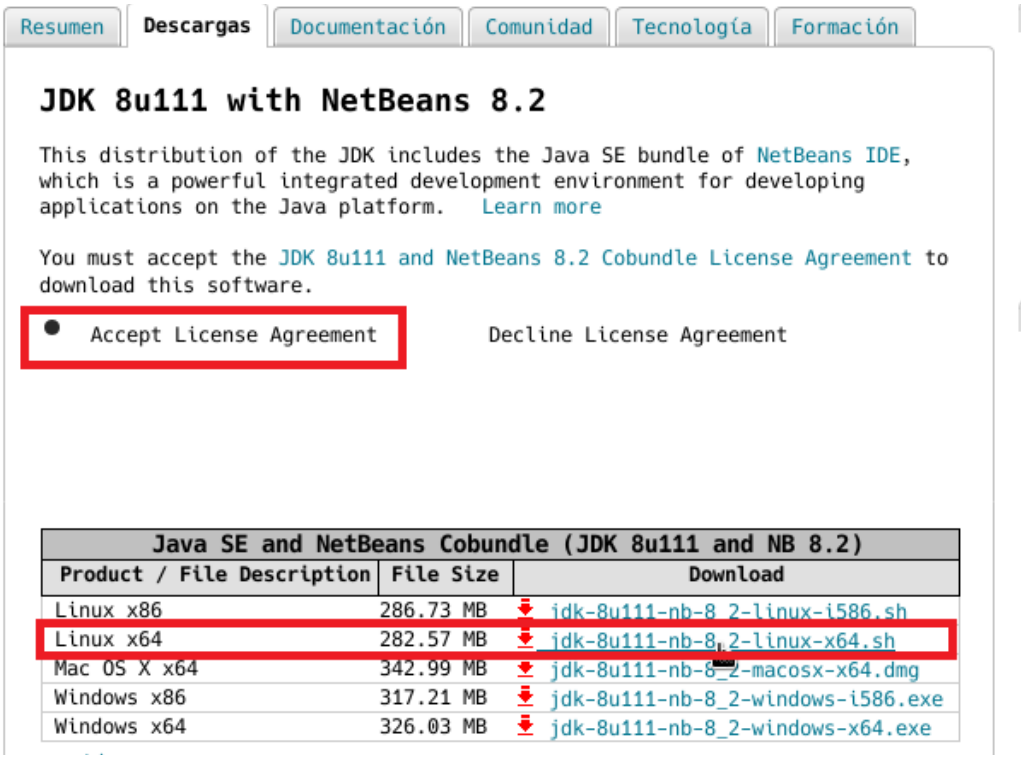
Inmediatamente nos direccionará a la página de Oracle en la parte de descargas.



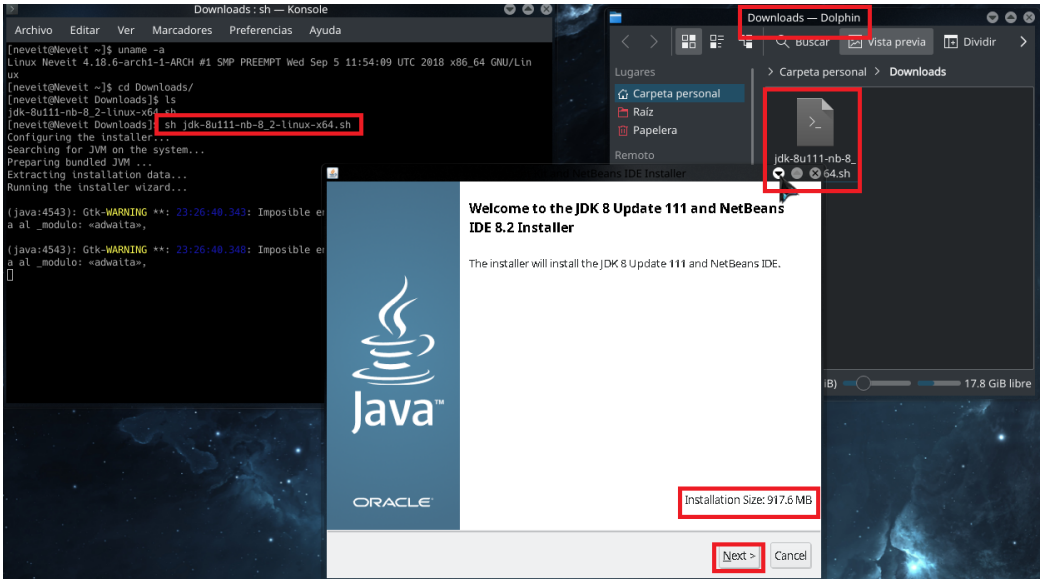
Para saber la arquitectura de nuestro equipo y así descargar el adecuado usamos el siguiente comando: *(uname -a)* en la terminal y damos Enter. Veremos que dice x86_64 en mi caso, entonces usare x64



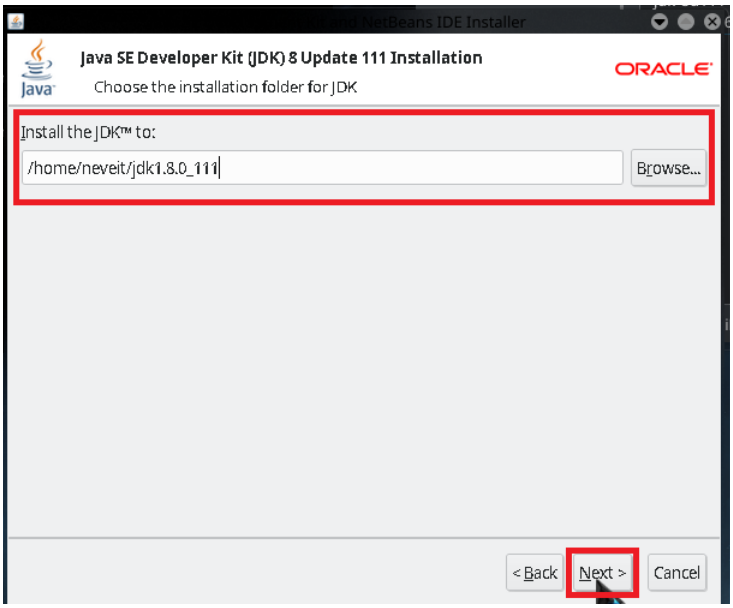
Una vez que ya sepamos que arquitectura tenemos daremos clic en el enlace para descargar el IDE.



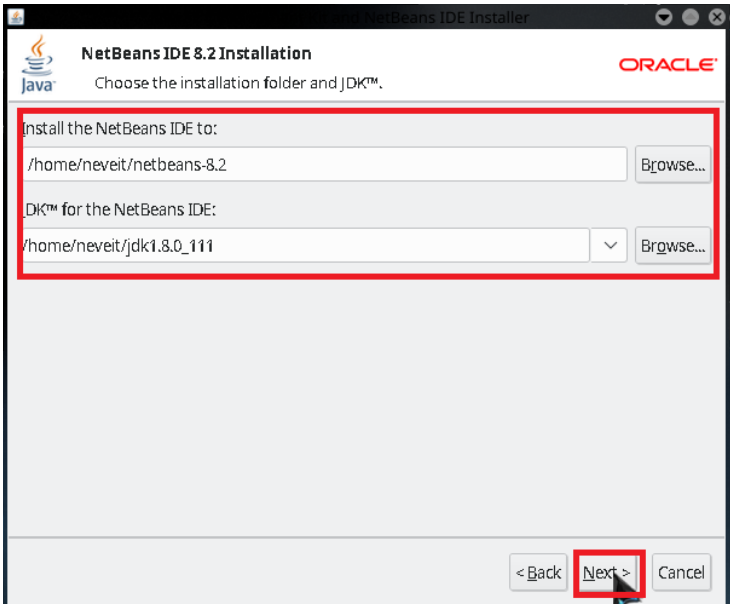
Cuando acabe de descargarse el archivo nos iremos a la carpeta donde se descargo, haciendo uso de la terminal.
Veremos que es un archivo de terminación *.sh* así que para instalarlo, se usara el comando *sh* seguido del nombre del archivo, como se ve en la imagen. Al dar Enter nos saldrá el proceso de instalación e inmediatamente la ventana para instalar NetBeans, y el espacio que ocupara.



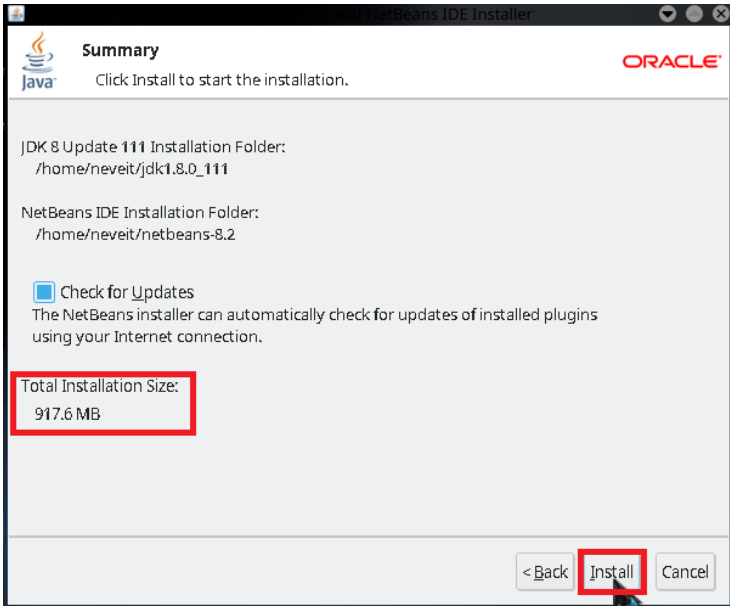
Allí podremos elegir donde queremos que se instale los recursos de Java (JDK) para poder hacer uso de ellos, en este caso dejare que se instale la ruta predeterminada, y daremos clic en siguiente.



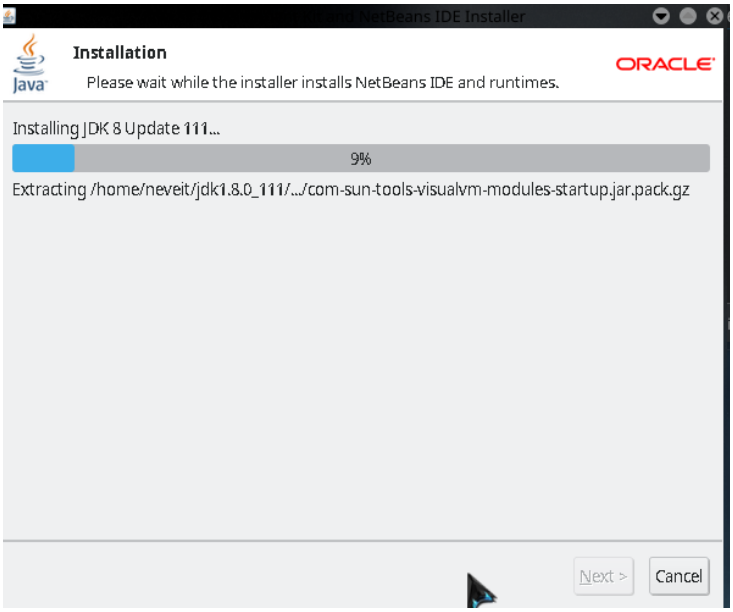
Pues nos pedirá la ruta de instalación para el IDE NetBeans y el JDK para NetBeans, de igual manera se dejara la ruta predeterminada, y damos clic en siguiente.



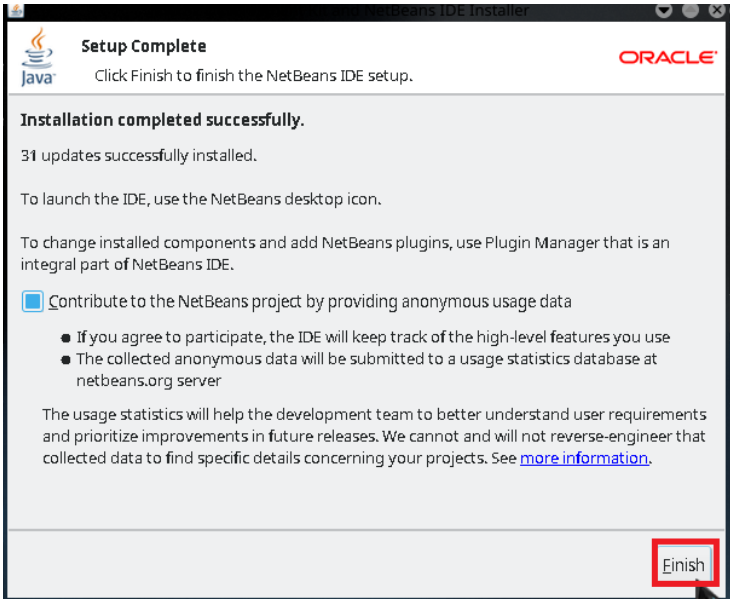
Por último nos preguntara sí queremos revisar si hay actualizaciones y el espacio que va ocupar y si deseamos empezar el proceso de instalación.



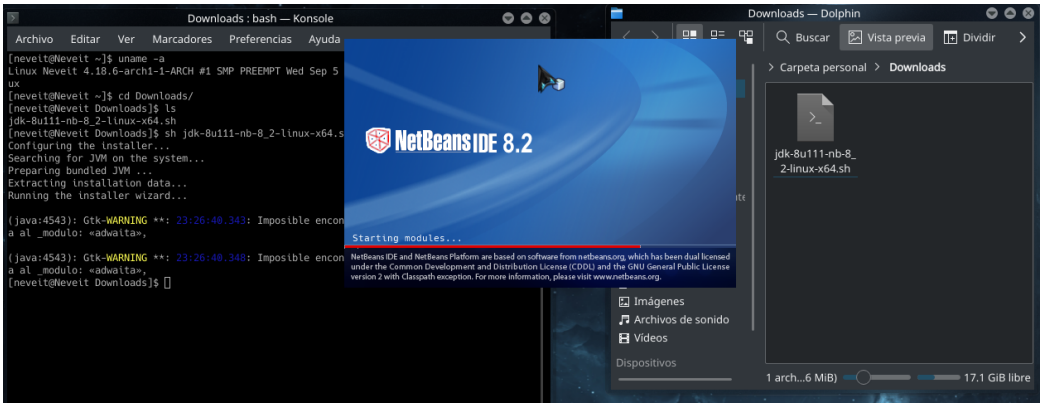
Inmediatamente empezara el proceso de instalación.



Al terminar nos dirá que la instalación fue un éxito.



Ya para terminar comprobamos que realmente tenemos el IDE instalado.



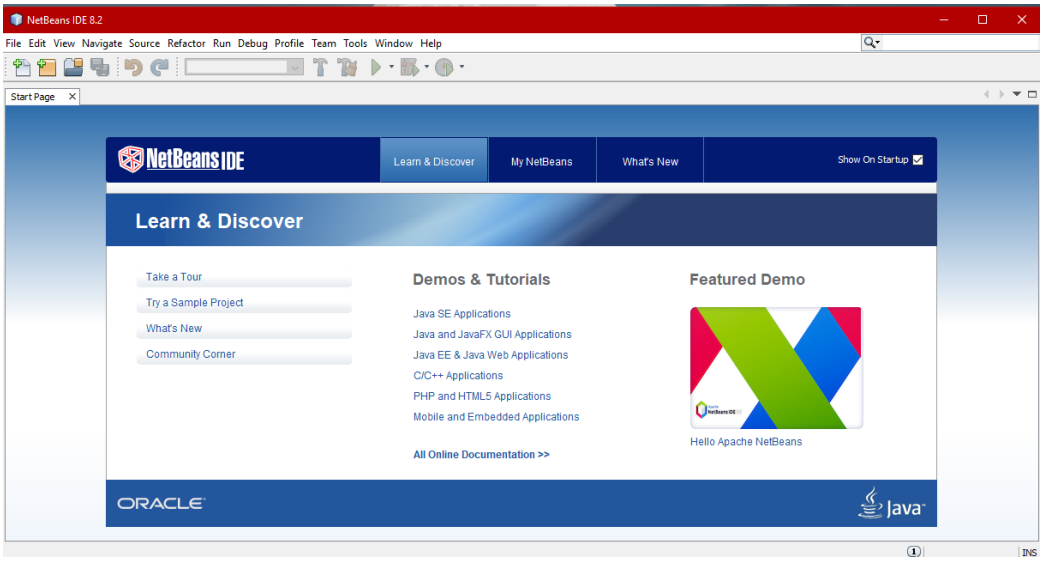
Observando esta pantalla se muestra que ya esta cargando el IDE, y hemos concluido su instalación. Ahora daremos un breve ejemplo para comprobar que ya se puede programar y una pequeña explicación sobre que proceso lleva la computadora para convertir nuestras palabras en instrucciones que pueda entender y usar.

0.3. Compilar y Correr el código

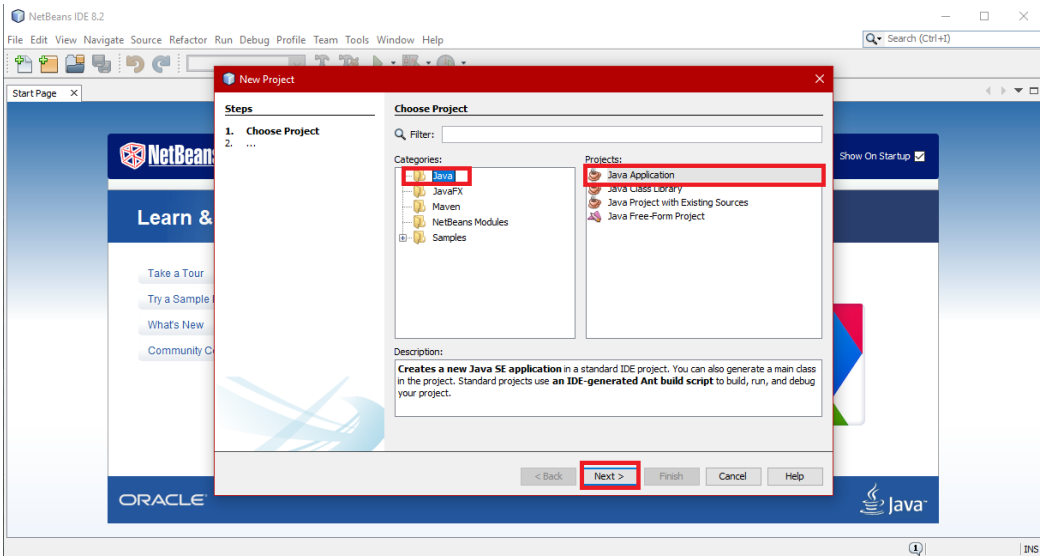
Para comprobar que podemos empezar a programar sin problemas, se recomienda copiar y pegar el siguiente código en el entorno de programación que se haya seleccionado:

```
1 public class HolaMundo
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hola Mundo!");
6     }
7 }
```

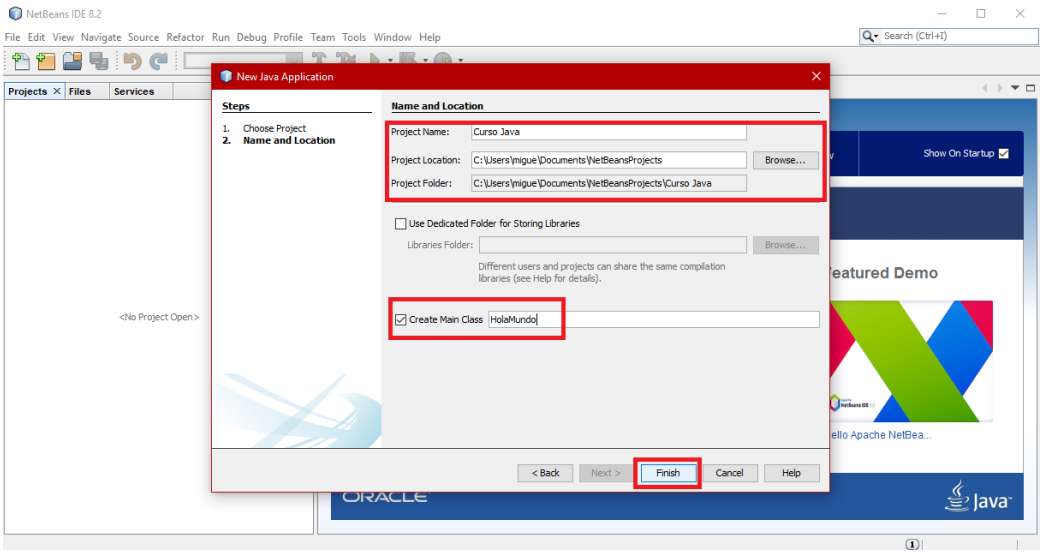
En este caso veremos como hacerlo en NetBeans.
Abrimos nuestro IDE, y en la barra de herramientas daremos clic en File y después en New Project.



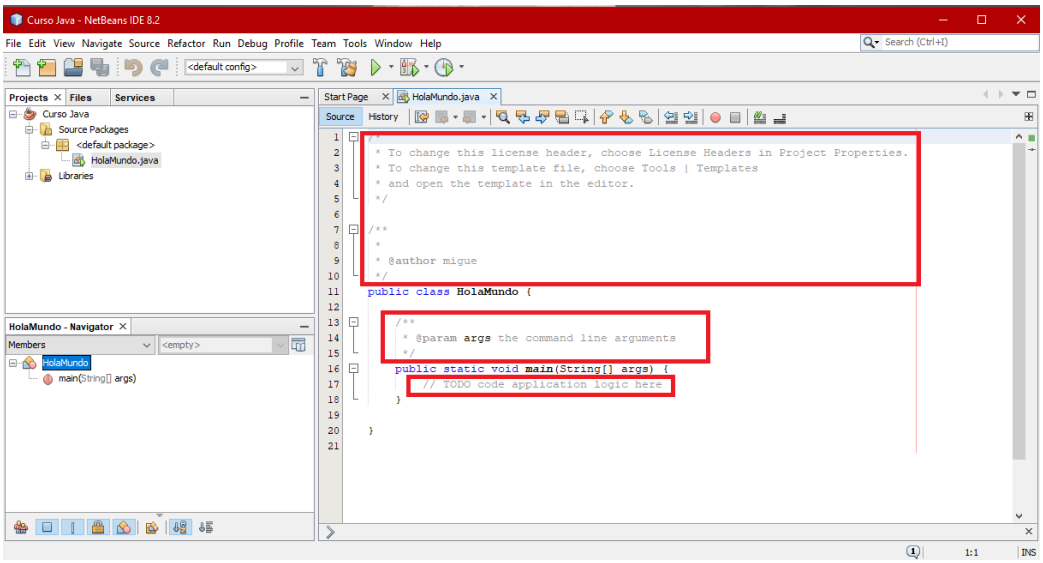
En Categoría elegimos Java, En Project daremos en Java Application. después clic en *next* (o siguiente).



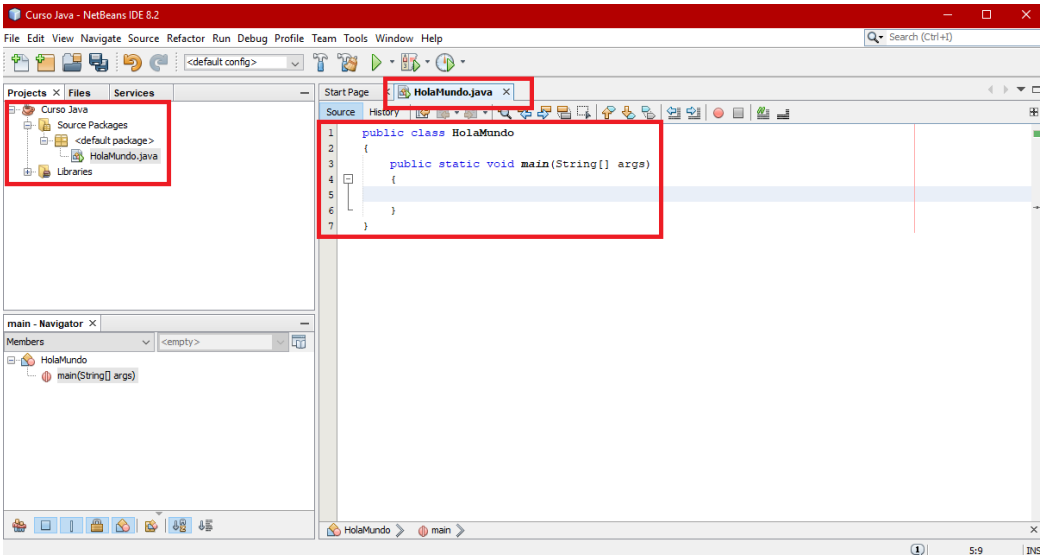
Nos pedirá el nombre del Proyecto así como la localización donde se guardara, y el nombre de la Clase.
La Primera letra debe ser mayúscula (mas adelante se explica el porque) una vez ya listo daremos clic en *next*.



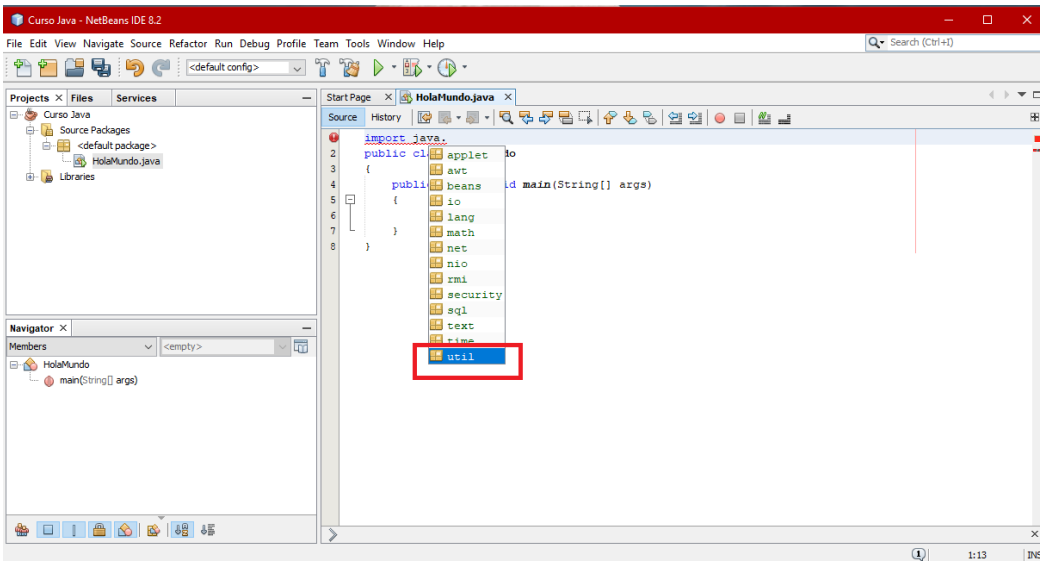
Todo lo que esta en color gris son comentarios, por el momento borraremos eso.



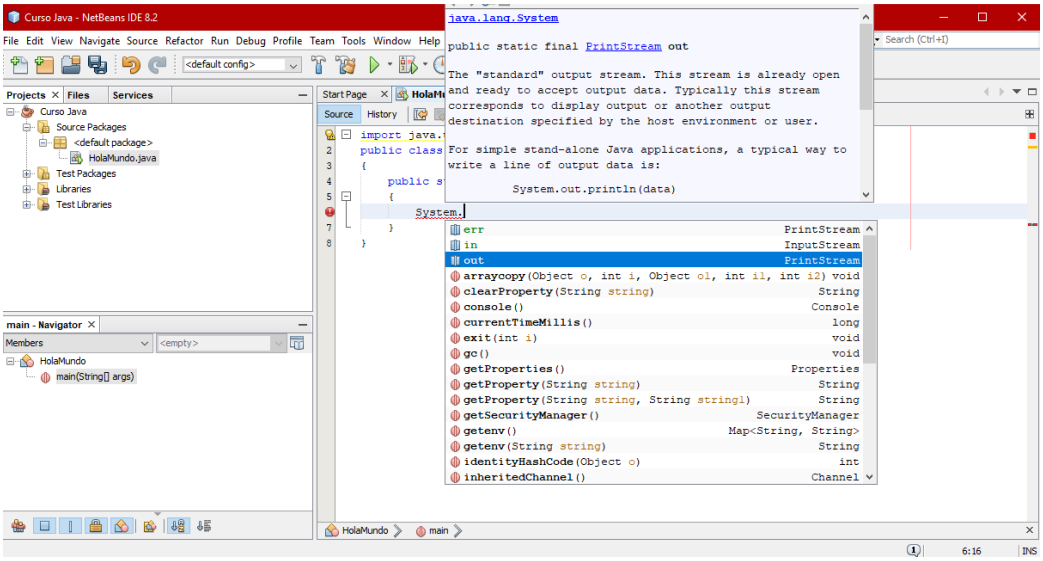
Nótese que del lado derecho dice Projects, es donde estarán nuestros programas, arriba donde se encuentra el código dice el nombre de la clase, y por ultimo una vez borrado los comentarios tenemos ya listo nuestra plantilla para nuestro primer programa en Java.



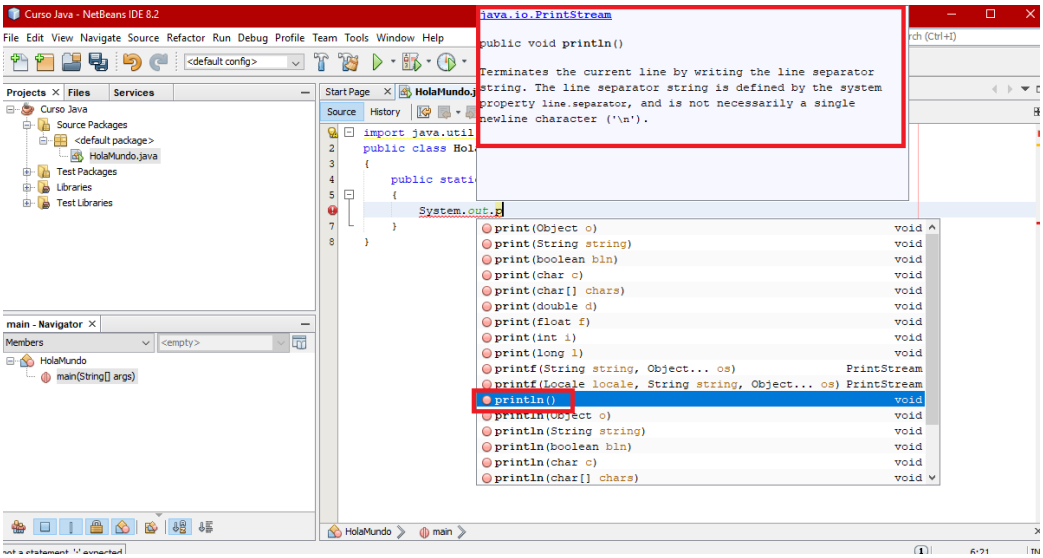
Ahora para hacer nuestro primer programa necesitaremos las paqueterías para que nuestro programa identifique los comandos que usaremos.



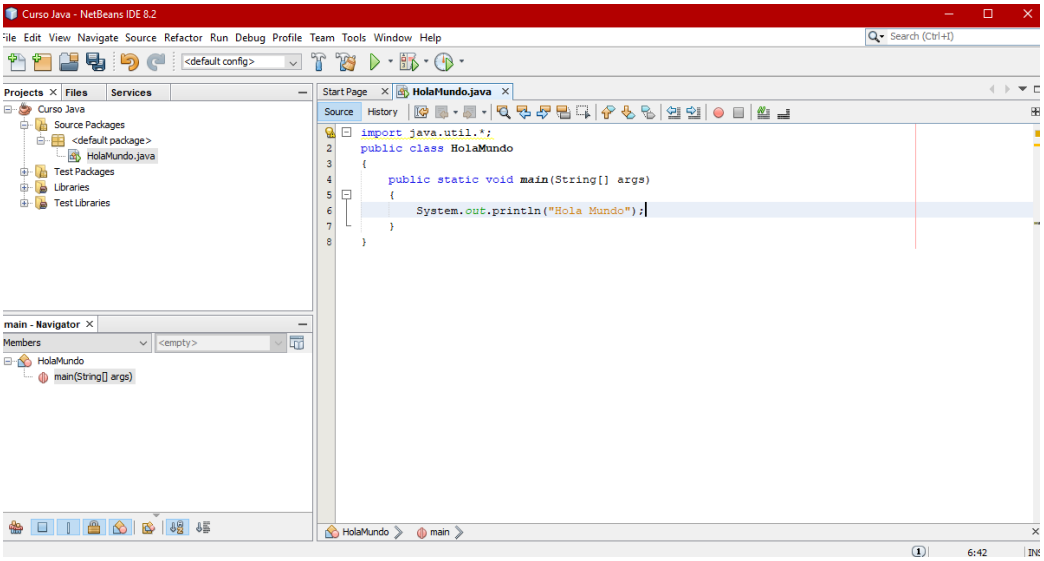
Como vemos al escribir algunos comandos que tienen varios tipos de funciones nos despliega todas las opciones disponibles así como su funcionamiento.



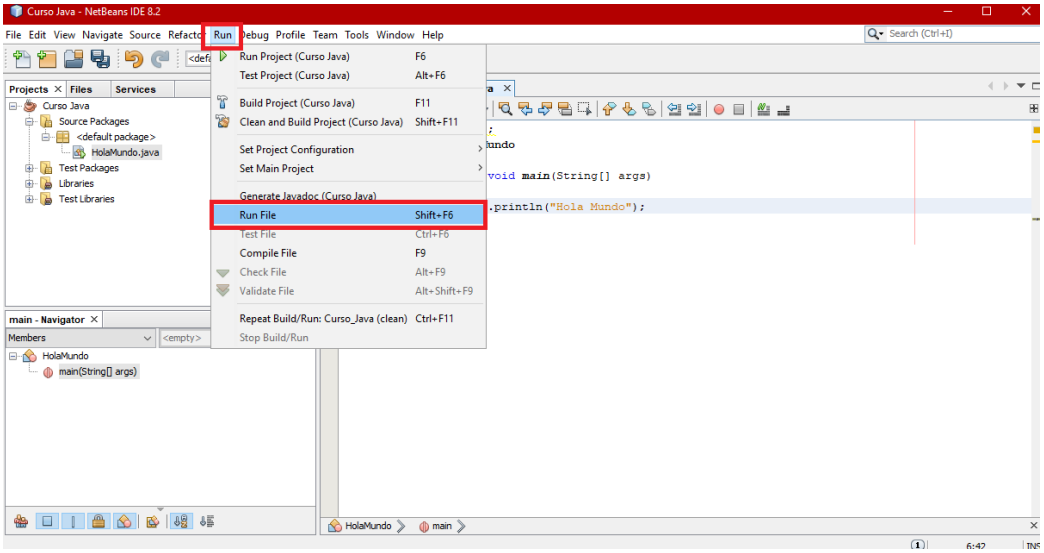
En este caso ocuparemos el comando *System.out.println("Hola Mundo");*, ya que al final le agrega el salto de linea.



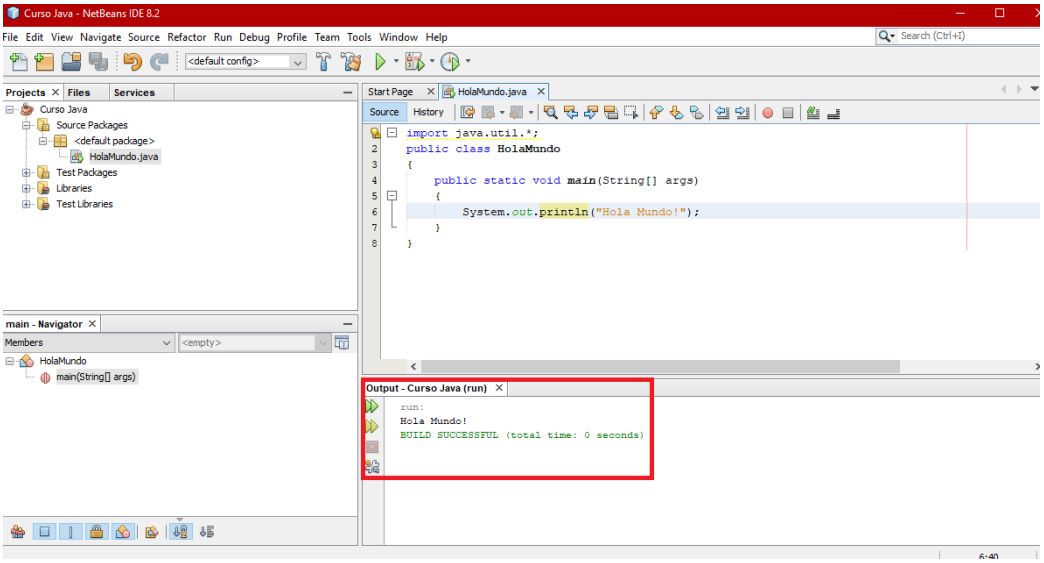
Y listo nuestro primer código en Java ya esta listo, te dejamos el código abajo para que lo puedas ver, copiar y pegar



Para correr el código nos desplazamos sobre la barra de herramientas sobre la pestaña de Run, y de allí damos clic en Run File.



En la parte de abajo nos saldrá la salida de nuestro programa en este caso, como es de costumbre nuestro Hola Mundo!



Debería de escribirse en la salida: *Hola Mundo*. La posición de la salida o este mensaje varían según lo que se seleccionó para programar. Independientemente de si se escribió conservando la jerarquía (espacios, tabulaciones y enters) o no.

Es importante que se haya escrito tal cuál el código, la excepción son el mensaje, la jerarquía y nombres como el que esta al lado de *class*: **Hola_mundo**. Esto es porque la máquina (o computadora) necesita instrucciones precisas, al menos en java, por lo que una mayúscula o espacios cambian por completo el sentido.

Nótese el siguiente ejemplo:

Código 1: hola mundo

```
/* ********************* TEXTO SIN ACENTOS **********************/
// Recuerda documentar tus codigos

public class hola_mundo
{
   public static void main(String []args)
   {
   System.out.println("Hola Mundo");
   }
}
```

Este ultimo ejemplo al intentarse compilar va a generar errores, para em-

pezar el nombre "*hola_mundo*" no es valido, aunque el programa lo acepta, esto es porque debe empezar con mayúscula al tratarse de un objeto o clase (class). El compilador se puede quejar por el **Void** ya que al llevar mayúscula no es algo que Java conozca, en si, no es lo mismo que con minúscula *void*. Y finalmente *Main* debe empezar con minúscula, de lo contrario, no existe la parte principal del programa, que como su nombre en inglés lo indica, debe ser "*main*" exactamente.
Entonces se debe tener cuidado al escribir el código, pero con la práctica se puede ir familiarizando con los errores que puedan existir, también se recomienda no confundir puntos y comas, así como números, esto es, se recomienda ser *muy* literal a la hora de escribir, ya que así es la computadora.

También hay que notar que hace falta en este código `import java.util.*;`, lo cual es una librería, hablaremos de ellas mas adelante en la sección 3 Palabras reservadas, pero por el momento se recomienda colocar al inicio de cada código, si se usa algún IDE de los ya mencionados, ya que esta librería contiene datos básicos que podemos llegar a usar, pero como la mayoría de los códigos aqui presentados incluyen cosas estándar, se puede omitir.

Capítulo 1

Bases

Una vez que ya sabemos como empezar a usar el lenguaje Java, continuaremos con aprender a usarlo, entendiendo sus componentes básicos así como la forma en que pensar para dirigir nuestros pensamientos e instrucciones hacia el lenguaje.

1.1. Programación

¿Qué es? Actualmente se asocia con la creación de las instrucciones y materiales que usara un dispositivo para crear y correr trabajos.
Cada vez gana más popularidad la programación, por sus pocos requerimientos para generar *lo que se pueda imaginar*, desde videojuegos y ocio, hasta cálculos científicos y económicos, entre otras aplicaciones.

1.2. Paradigmas de programación

En si los paradigmas son la forma de pensar y/o resolver los problemas que requieren de programación. Unos lenguajes pueden tomar distintos paradigmas a la vez, mientras que otros se restringen a solo uno.
De los más populares se encuentran:

- Paradigma Estructurado.
- **Paradigma Orientado a Objetos.**
- Paradigma Orientado a Eventos.
- Paradigma Funcional.

1.3. Paradigma Orientado a Objetos (POO).

Este paradigma es de los mas populares, se basa en la abstracción de entidades o cosas reales hacia clases, las cuales generan objetos o instancias con las propiedades especificas del objeto, estas propiedades se llaman atributos y pueden representar edad, tamaño, entre otras valores. Las acciones que pueden realizar (o realizarse sobre los mismos objetos) se les conoce como métodos.
De estas principales características surgen otros conceptos como el encapsulamiento, el cual permite que ciertas partes de los objetos no puedan usarse o modificarse por algo ajeno al objeto, así como la herencia que, como dice el nombre, permite heredar partes del objeto (o más bien su abstracción como clase) a otras para que las usen de otra forma o agreguen más componentes.

Es común que este paradigma sea combinado con el estructurado, el cual sigue una serie de pasos e instrucciones en secuencia, esto es que sigue una estructura, lo cual permite facilitar el acomodo del programa y su completitud. Estos dos paradigmas son utilizados por java, y se usaran en este manual al ser los mas intuitivos y fáciles de seguir, aunque se debe recordar que los lenguajes no se restringen a un solo paradigma y pueden usar inclusive más o menos.

1.4. Empezando a programar en java

Como se vio en la sección 0.3 Compilar y Correr el código, puede usarse cualquier medio para programar, siempre que tenga la misma versión (o mayor en muchos lenguajes, incluyendo java), para propósitos de introducción y permitirle al lector programar desde donde este (inclusive dispositivos móviles), el código mostrado en este manual ha sido compilado y ejecutado en el sitio: https://www.compilejava.net/.
Para empezar se mostraran las partes principales de un programa en java con el ejemplo clásico de "Hola Mundo":

Código 1.1: Hola mundo

```
1 /* ********************* TEXTO SIN ACENTOS *********************/
2 // Recuerda documentar tus codigos
3
4
5 /* Este es un comentario que se contiene
6    dentro de los simbolos delimitadores,
7    sin importar cuantas lineas se tengan.
```

```
8       Los comentarios son ignorados por el
9       compilador, pero sirven para hacer
10      notas sobre el codigo.
11      */
12  //Este es un comentario de linea, funciona solo en la misma linea
13
14  public class MiClase{ //La esencia del POO son las clases
15
16      //Java siempre necesita la siguiente funcion para correr
17      public static void main(String []args){ /*Entre () se tienen
            los datos que toma la funcion, conocidos como argumentos*/
18
19          System.out.println("Hola Mundo"); /*Esta funcion
                escribe(imprime) lo que esta entre parentesis, siempre
                que cumpla la sintaxis"*/
20
21       } //Los corchetes encierran el contexto, no pueden ir solos.
22  } //Los corchetes van desde dentro hasta afuera. {3{2{1}2}3}
```

Como se menciona en el código, los comentarios (Texto en verde) son ignorados por el compilador, lo cual es útil para que los humanos podamos dejar información sobre que hacer partes del código. Sin embargo, los acentos o caracteres fuera del abecedario simple pueden aparecer extraños o modificarse, por lo que se recomienda no usarlos.
Se considera buena práctica comentar el código que uno realiza, así puede saber para que era cada parte si se llega a olvidar posteriormente o se quiere revisar.

Se deja como ejercicio al lector compilar y ejecutar el programa eliminando los comentarios, para que pueda observar que produce el mismo resultado que si se dejan. **Nótese** que lo que no esta de color verde (no es un comentario) lleva mayúsculas y minúsculas, así como puntos y orden, si se reemplazan, omiten o eliminan/agregan cosas, entonces el programa marcara errores. Véase la sección 0.3 Compilar y Correr el código para mas información sobre algunos errores comunes.

La función *System.out.println(String texto)* debe cumplir con que su argumento sea un tipo de dato *String* (que por ejemplo llamamos *texto*) el cual es un arreglo/conjunto de caracteres.
Al colocar algo entre " " estamos diciendo que todo lo de adentro es un arreglo de caracteres (incluyendo los espacios), podemos juntarle (o concatenarle)

al *String* otros datos con el signo + que también puedan representarse así como los números, por ejemplo *System.out.println(" Un texto y el numero " + 3.4 + " aquí.")* imprimirá en pantalla: *Un texto y el numero 3.4 aquí.*. Nótese que los espacios entre "" se conservaron en la frase final.

El argumento String []args del programa anterior es un dato que veremos mas adelante en la sección 5.3 Memoria.

1.5. Tipos de datos básicos

Para almacenar y usar datos, debemos declarar de que tipo son, para que la computadora conozca como operarlos, a continuación se presentan los mas generales y comunes en la mayoría de los lenguajes de programación:

- **bit**: unidad de almacenamiento mínima, puede valer 1 o 0;
- **byte**: unidad compuesta por 8 bits;
- **char** (carácter): puede valer 1 a 2 bytes, representa símbolos o letras;
- **int** (entero): 2 a 4 bytes, puede representar números enteros.
- **float** (punto flotante): 4 bytes, representa números reales.
- **double** (doble punto flotante): 8 bytes, representa números reales con mayor precisión.

Dado que los tamaños de cada dato pueden variar a veces por la arquitectura de la computadora y mas que nada por el lenguaje de programación, a continuación se muestra un ejemplo de código[1] que imprime los tamaños de cada dato básico en el lenguaje Java que se usa en este manual para familiarizar al lector:

Código 1.2: Tipo de Datos

```
1 /* ********************* TEXTO SIN ACENTOS **********************/
2 // Recuerda documentar tus codigos
3
4 public class Datos
5 {
6     public static void main(String[] args)
7     {//aqui se declara un dato int y
```

[1]Código compilado y ejecutado en https://www.compilejava.net/

```
8         int div=Byte.SIZE; // se le asigna un valor con =
9         System.out.println("Tamanio de Byte: " + div + " bits.\n"
10          + "    char:   " + (Character.SIZE/div) + " Bytes.\n"
11          + "    int:    " + (Integer.SIZE/div) + " Bytes.\n"
12          + "    float:  " + (Float.SIZE/div) + " Bytes.\n"
13          + "    double: " + (Double.SIZE/div) + " Bytes.\n");
14    }
15 }
```

```
Successfully compiled /tmp/java_s7BIaf/Mainthings.java <--
main method
```

```
Tamanio de byte: 8 bits.
        char: 2 bytes.
        int: 4 bytes.
        float: 4 bytes.
        double: 8 bytes.
```

Observa como cambia la representación de cada valor al ser impreso según su tipo de dato.

Código 1.3: Datos impresos

```
1  /* ********************* TEXTO SIN ACENTOS *********************/
2  // Recuerda documentar tus codigos
3
4  public class DatosImpresos
5  {
6    public static void main(String[] args)
7    {
8      char a=42;
9      int b=42;
10     float c= (float) (41 + 0.9999998);
11     double d=41 +0.9999998;
12     System.out.println("char a = " + a);
13     System.out.println("int b = " + b);
14     System.out.println("float c = " + c);
15     System.out.println("double d = " + d);
16   }
17 }
```

En el código anterior la parte de *(float)* indica a la computadora que queremos convertir un tipo de dato a un tipo *float*, esta parte junto con los respectivos paréntesis que encierran a la operación sirven para lograr asignar el valor deseado a c, ya que al ser float de un tamaño menor a double, debemos indicarle **explícitamente** a la computadora que queremos hacer la conversión del tipo de dato aunque **perdamos** dígitos.

También obsérvese como float recorta dígitos que en algunas situaciones nos podrían ser útiles, así que es importante saber con que propósito se usara la aplicación, para saber si vale la pena el peso extra que conlleva un double, o un float es suficiente.
Experimenta un poco cambiando el valor de a para observar que caracteres puede llegar a escribir.

Capítulo 2

Operadores

Para asociar valores unos con otros, como por ejemplo sumas, relaciones, lógica, etc... , requerimos hacer operaciones tal como en matemáticas, es de esta manera como se definen signos, conocidos como **operadores**, en java para hacer estas operaciones, a continuación se presenta una tabla con el orden que siguen estos operadores, siendo operadores de la misma linea o precedencia, evaluados de izquierda a derecha, excepto los últimos de asignación que son evaluados de derecha a izquierda:

Orden	Operador	Descripción
1	++ −−	Operadores posfijos de decremento e incremento
2	++ −− + − ∼ !	Operadores prefijos unarios
3	* / %	Operadores binarios multiplicativos y modulo
4	+ −	Operadores binarios de adición
5	<< >> >>>	Operadores binarios de shift o desplazamiento en bits
6	< > <= >= *instanceof*	Operadores relacionales
7	== ! =	Operadores de equidad
8	&	Operador bit a bit AND (y)
9	ˆ	Operador bit a bit OR exclusivo (xor/o exclusivo)
10	|	Operador bit a bit OR (o)
11	&&	Operador AND (y) logico
12	||	Operador OR (o) logico
13	? :	Operador ternario
14	= += -= *= /= %= &= ˆ= |= <<= >>= >>>=	Operadores de asignación

Tabla 2.1: Operadores [1]

Si bien se observan muchos operadores, en realidad necesitamos solo unos cuantos al iniciar, de los cuales ya hemos visto algunos como + y =. Para tener una buena idea de como se pueden usar véanse los siguientes temas sobre operadores.

2.1. Operadores aritméticos

Tal como su nombre lo indica son aquellos que conocemos por operaciones matemáticas, el único que nos puede parecer extraño es el % el cual, como

[1] Debido a que pueden existir algunas diferencias entre fuentes, se uso la tabla de la compañía Oracle [**?**].

se menciona en la tabla 2.1 es el operador "modulo", en si divide un número a entre otro b y regresa el residuo de esta división. Veremos como funcionan en el siguiente ejemplo en el cual usaremos el operador = que se menciona a detalle en 2.7 Operadores de asignación, por el momento podemos decir que funciona para "asignar" o dar un valor a una variable. Observe el siguiente ejemplo:

Código 2.1: Operadores aritméticos

```
1  public class OperadoresArit
2      {
3        public static void main(String[] args)
4        {
5          int a= 7, b=3; // se asigna un valor con =
6          System.out.println(a + " + " + b +" = " + (a + b) );
               //operador binario suma
7          System.out.println(a + " - " + b +" = " + (a - b) );
               //operador binario resta
8          System.out.println(a + " * " + b +" = " + (a * b) );
               //operador binario multiplicacion
9          System.out.println(a + " / " + b +" = " + (a / b) );
               //operador binario division
10         System.out.println(a + " % " + b +" = " + (a % b) );
               //operador binario modulo
11         /*notese que 3 divide exactamente 2 veces al 6 pero deja un
               residuo de 1*/
12       }
13     }
```

Como se menciono anteriormente, las operaciones se realizan de izquierda a derecha, ahora **experimenta** cambiando de lugar la variable a con la b para ver como afecta el orden a las operaciones.
Estos operadores también funcionan con otros tipos de datos básicos que vimos, por ejemplo **experimenta** cambiando el int por un double y observa las salidas, también cambia los valores de a y b a negativos (agrega un - antes del numero y después del signo =).

2.2. Operadores unarios

Como su nombre lo indica, son aquellos que aplican sobre una sola variable. De los operadores de incremento y decremento, es importante marcar la diferencia entre los posfijos y prefijos como se muestra en la siguiente lista:

- $++$: Operador de incremento, aumenta el valor de la variable en 1.
  - *Postfijo*: primero evalúa la variable y luego incrementa su valor.
  - *Prefijo*: primero incrementa su valor y luego evalúa la variable.
- $--$: Operador de decremento, reduce el valor de la variable en 1.
  - *Postfijo*: primero evalúa la variable y luego reduce su valor.
  - *Prefijo*: primero reduce su valor y luego evalúa la variable.
- **-** : Da el valor negativo. Si era negativo será positivo.
- $+$: Da el valor aritmético. Su uso es muy raro pero puede llegar a servir como se mostrará en un ejemplo mas adelante.
- **!** : Niega el valor de una variable, se usa con "booleanos" como se muestra en el ejemplo 2.2.
- $\sim$: Niega bit a bit los valores de una variable, este sera explicado a detalle en 2.5 Operadores bit a bit.

Código 2.2: Operadores unarios

```
1  public class OperadoresUnar
2  {
3    public static void main(String[] args)
4    {
5      int a= -1, b= 0, c=0;
6      boolean bool= true; //dato de verdadero y falso
7
8      System.out.println("Operadores postfijos");
9      System.out.println("a++ = " + (a++) );
10     System.out.println("a = " + a );
11     System.out.println("a-- = " + (a--) );
12     System.out.println("a = " + a );
13     System.out.println("\nOperadores prefijos");
14     // el \n anterior es una secuencia de escape
15     System.out.println("b = " + b );
16     System.out.println("++b = " + (++b) );
17     System.out.println("b = " + b );
18     System.out.println("--b = " + (--b) );
19     System.out.println("b = " + b );
20     //Observa bien el siguiente enunciado
21     c = a++ + ++b;
```

```
22      System.out.println("\nc = " + c );
23
24      System.out.println("\nOperador unario !");
25      System.out.println("!bool = " + !bool);
26    }
27  }
```

La secuencia de escape "\n" y otras más se explican en la sección 6.1 Secuencias de escape. El tipo de dato booleano "bool" se explica en la siguiente bisección. Similarmente al ejemplo anterior, **experimenta** cambiando de lugar la variable *a* con la *b* para ver como afecta el orden a las operaciones que le dan un valor a *c*.
Experimenta cambiando el int por un double y observa las salidas, también cambia los valores de a y b.

2.3. Operadores relacionales y los booleanos

Los operadores relacionales son aquellos que analizan una relación y resuelven el si esta es verdadera o falsa.
Mencionando esta condición de verdad o falsa, también se presenta un nuevo tipo de dato: el booleano. Cuyo nombre viene del matemático *George Boole* quien creo un álgebra basada completamente en 1 y 0, o bien pues verdadero o falso, entre otros posibles nombre.
El tamaño en bytes de un dato de tipo booleano depende mucho de la maquina virtual que usa java, la cual también puede verse afectada por cosas como arquitectura de computadora entre otros detalles. Lo que se mantiene cierto es que solo puede tomar 1 valor a la vez, el cual se le puede asignar con las palabras reservadas *true* y *false*. Se hablara más de estas palabras en la siguiente sección 3 Palabras reservadas.
Conociendo esto se explican las características de cada operador: *instanceof*.

- $<$: verifica que el valor de la izquierda es menor que el de la derecha.
- $>$: revisa que el valor de la izquierda es mayor que el de la derecha.
- $==$: analiza si el valor de la izquierda es igual al de la derecha. Es importante **no confundir** este operador con el $=$ de **asignación**.
- $!=$: checa que el valor de la izquierda es diferente al de la derecha.
- $<=$: verifica que el valor de la izquierda es menor o igual al de la derecha.

- >= : revisa que el valor de la izquierda es mayor o igual al de la derecha.
- **instanceof** : revisa si el objeto de la izquierda es una instancia o puede representarse como una clase de la derecha.

Observa el siguiente código para tener una idea mejor de los booleanos.

Código 2.3: Operadores relacionales

```
1  public class OperadoresRela
2  {
3    public static void main(String[] args)
4    {
5      int a= 7, b=3;
6      String ejemplo ="a";
7      System.out.println(" a < b = " + (a < b) );
8      System.out.println(" a > b = " + (a > b) );
9      System.out.println(" a == b = " + (a == b) );
10     System.out.println(" a != b = " + (a != b) );
11     System.out.println(" a <= b = " + (a <= b) );
12     System.out.println(" a >= b = " + (a >= b) );
13
14     System.out.println( ejemplo instanceof String);
15   }
16 }
```

Como se observa, los resultados devueltos son de tipo booleano.
Ahora **experimenta** con los valores de a y b para ver como se analizan de forma diferente.

2.4. Operadores lógicos

Estos operadores son aquellos que se encargan de la lógica y las evaluaciones, vimos brevemente en 2.2 Operadores unarios al operador negación "!". Observa las tablas de a continuación para ver como se evalúan según los operandos, donde V es Verdadero y F es Falso.

a	b	||
F	F	F
V	F	V
F	V	V
V	V	V

Tabla 2.2: Valores con ||

a	b	&&
F	F	F
V	F	F
F	V	F
V	V	V

Tabla 2.3: Valores de &&

a	!a
F	V
V	F

Tabla 2.4: Valores con !

Ejecuta el siguiente código para ver el comportamiento de estos operadores.

Código 2.4: Operadores logicos

```
1  public class OperadoresLogi
2  {
3    public static void main(String[] args)
4    {
5      boolean a= true, b=false;
6      System.out.println(" a && b = " + (a && b) );
7      System.out.println(" a || b = " + (a || b) );
8      //ahora usando el operador de negacion
9      System.out.println(" !b = " + (!b) );
10     System.out.println(" a && !b = " + (a && !b) );
11     System.out.println(" !a || b = " + (!a || b) );
12   }
13 }
```

Ahora **experimenta** cambiando los valores de a y b, *recuerda que un boolean solo toma true ó false.*

2.5. Operadores bit a bit

Estos operadores pueden parecer un poco mas complicados, pero su funcionamiento es similar a la de los operadores lógicos, solo que se usa 1 para representar verdad y 0 para falso, en base en esto se representan los numeros en base binaria, y se operan.
Cuando mencionamos los tamaños de bytes en 1.5 Tipos de datos básicos de cada dato, también se mencionó que se componían de bits, estos se suelen analizar de derecha a izquierda, se cuentan empezando desde 0, y si tienen valor de 1 se eleva 2 al valor de su posición y se suma al total que vale.
Por ejemplo: 1101, de derecha a izquierda tenemos un 1 en la posición 0, un segundo 1 en la posición 2, y un ultimo 1 en la posición 3, entonces el valor de 1101 sera $2^0+0+2^2+2^3 = 1+4+8 = 13$. Conociendo esto se enlista como

funciona cada operador, tomando en cuenta que como su nombre lo marca, operan cada **bit** de a por otro **bit** de b en la misma posición, el número en binario que obtienen al final es el que regresan.

- **&** : hace la misma operación que el operador && pero con cada bit.
- | : hace la misma operación que el operador || pero con cada bit.
- ^ : regresa 1 *solo si* los 2 bits que esta analizando son diferentes.
- ∼ : aplica la operación de ! en cada bit de la variable..

Hasta el momento hemos visto que se da el valor tal cual a la variable, pero también hay otras formas de asignarles valores, ahora veremos que se puede separar un valor con guiones bajos _ así como representarlos en forma binaria poniendo "0b" al inicio del número a representar en esa base.
Corre el siguiente programa para ver lo anterior en acción.

Código 2.5: Operadores bit a bit

```
1  public class OperadoresBits
2  {
3    public static void main(String[] args)
4    {
5      char a=0b0001_0011_1001;
6      char b=0b01011010;
7      System.out.println(" a = " + (+a)
8          + " = " +Integer.toBinaryString(a) );
9      System.out.println(" b = " + (+b)
10         + " = " +Integer.toBinaryString(b) );
11     System.out.println(" a & b = " + (a & b)
12         + " = " +Integer.toBinaryString(a & b));
13     System.out.println(" a ^ b = " + (a ^ b)
14         + " = " +Integer.toBinaryString(a ^ b));
15     System.out.println(" a | b = " + (a | b)
16         + " = " +Integer.toBinaryString(a | b));
17     System.out.println(" ~a = " + (~a)
18         + " = " +Integer.toBinaryString(~a));
19   }
20 }
```

Observa que el operador ∼ regreso un valor negativo, esto es por la forma en que se representan los valores negativos en binario, donde al tener una cantidad de bits limitados, se optó por dividir la cantidad de números representables a la mitad más uno para los negativos.

Como muestra la representación binaria gracias al método *toBinaryString()* de *Integer*, no cuesta más que invertir **todos** los bits que contiene el tipo de dato, para obtener el valor original en negativo menos 1 extra.
Nota como el operador unario + no es totalmente inútil, si en este ejemplo eliminas el + al lado de la a o b veras que el programa escribe su representación como carácter, en lugar de número.
Experimenta cambiando los valores de a y b para ver como cambian los resultados, también intenta hacer la representación a mano de los números en binario y las operaciones para tener una mejor idea de como funcionan.

2.6. Operadores de desplazamiento de bits o shift

Estos operadores actúan sobre la representación en binario del tipo de dato que usemos, siendo cada uno un poco diferente:

1. << : Mueve los bits hacia la izquierda agregando 0s por la derecha.
2. >> : Mueve los bits hacia la derecha agregando 0s o 1s, por la izquierda, si el valor original es positivo o negativo respectivamente.
3. >>> : Mueve los bits hacia la derecha agregando 0s por la izquierda, ignora si el valor era negativo y regresa su representación en positivo.

Ve el siguiente ejemplo:

Código 2.6: Operadores de desplazamiento en bits

```
1  public class OperadoresDespl
2  {
3    public static void main(String[] args)
4    {
5      char a=0b0001_0011_1001;
6      int b=-10;
7      System.out.println(" a = " + (+a)
8          + " = " +Integer.toBinaryString(a) );
9      System.out.println(" b = " + (+b)
10         + " = " +Integer.toBinaryString(b) );
11     System.out.println("\nOperador <<");
12     System.out.println(" a << 2 = " + (a << 2)
13         + " = " +Integer.toBinaryString(a << 2));
14     System.out.println(" b << 2 = " + (b << 2)
15         + " = " +Integer.toBinaryString(b << 2));
```

```
16      System.out.println("\nOperador >>");
17      System.out.println(" a >> 2 = " + (a >> 2)
18          + " = " +Integer.toBinaryString(a >> 2));
19      System.out.println(" b >> 2 = " + (b >> 2)
20          + " = " +Integer.toBinaryString(b >> 2));
21      System.out.println("\nOperador >>>");
22      System.out.println(" a >>> 2 = " + (a >>> 2)
23          + " = " +Integer.toBinaryString(a >>> 2));
24      System.out.println(" b >>> 2 = " + (b >>> 2)
25          + " = " +Integer.toBinaryString(b >>> 2));
26    }
27  }
```

Similarmente a los operadores anteriores **experimenta** con los valores para observar las distintas representaciones.

2.7. Operadores de asignación

Como su nombre lo indica, son aquellos que asignan un valor a otro. Hasta el momento hemos estado utilizando =, pero también podemos mezclar este junto la mayoría de los operadores vistos anteriormente para ahorrar pasos a la hora de escribir. Todos siguen el mismo formato donde "a *operador*= b" es lo mismo que "a = a *operador* b".
Hablando del operador = que tanto hemos usado, este hace tal cual lo que menciona, donde hace que la variable de la izquierda tenga lo mismo que lo de la derecha. Es importante asegurarse de que es compatible el valor que queremos asignar, inclusive usando una conversión explicita en la forma de "(*tipo_de_dato*)", ya que si asignamos un valor no valido, nuestro programa puede fallar.
Observa el siguiente ejemplo de operadores:

Código 2.7: Operadores de asignación

```
1   public class OperadoresAsig
2   {
3     public static void main(String[] args)
4     {
5       int a=1, b=-10;
6       a += 1;
7       System.out.println(" a += 1 = " + (a));
8       a -= b;
9       System.out.println(" a -= b = " + (a));
```

```
10      a *= b;
11      System.out.println(" a *= b = " + (a));
12      a /= b;
13      System.out.println(" a /= b = " + (a));
14      a %= b;
15      System.out.println(" a %= b = " + (a));
16      a &= b;
17      System.out.println(" a &= b = " + (a));
18      a ^= b;
19      System.out.println(" a ^= b = " + (a));
20      a |= b;
21      System.out.println(" a |= b = " + (a));
22    }
23  }
```

Experimenta con los valores de a y b. Se deja como ejercicio agregar los últimos tres operadores de asignación, de los cuales se debe notar que pasa al pedir que se muevan los bits en una cantidad negativa.

2.8. Operador ternario

Este operador aunque puede parecer algo confuso, es muy útil para asignar valores al crearlos, además de ser más rápido que usar su equivalente en *if* y *else* que se vera más adelante.
Funciona con la siguiente estructura: *condición* ? a : b. Lo cual se puede leer como: si *condición* es cierta, entonces regresa el valor de a, si no, el valor de b. Analiza el siguiente ejemplo:

Código 2.8: Operador ternario

```
1   public class OperadorTern
2   {
3     public static void main(String[] args)
4     {
5       int a=1, b=-10;
6       System.out.println(" El valor mayor entre a y b es: " + (a>=b ?
            a : b) );
7     }
8   }
```

Dado que este operador tiene la precedencia casi más baja, se debe usar paréntesis para asegurar el orden en que se realizan las operaciones.
Como los ejercicios anteriores, *experimenta* con los valores de a y b para ver

como cambia el resultado.

Capítulo 3

Palabras reservadas

Hemos entendido hasta ahora algunos componentes, pero ¿Qué pasa con algunas palabras que hemos estado viendo como "public"?, estas se conocen como *palabras reservadas* del lenguaje Java, y como el nombre lo indica, tienen un propósito ya definido por el lenguaje.
Algunos que hemos utilizado son los nombres de los tipos de datos: *int, char, double, float, boolean*. También los valores *true* y *false*.
Como se nota en el código, estas palabras se marcan de un color distintivo, en este manual se usa el color azul.
Cada palabra puede tener un significado y uso distinto, a continuación se muestran algunos de los más comunes, excluyendo los mencionados anteriormente que ya hemos visto:

- **package**: Aunque no se ha mencionado en los códigos de ejemplo, este es necesario para indicar al programa completo el contexto en donde se encuentra, esto es, cada programa diferente que hagamos y este relacionado dentro del mismo programa se compile todo junto. Es común usarlo en programas compuestos por varios archivos, así como en algunos IDEs que lo llegan a pedir como NetBeans.

- **import**: importar, sirve para traer variables, clases, funciones entre otros elementos que se encuentran librerías, estas son en las que se basa Java para generalizar su utilidad y se verán algunas de las mas populares a mayor detalle en el manual intermedio de Java. Las librerías y sus secciones se suelen separar por puntos, siendo la palabra más a la izquierda la que contiene a las de la derecha, si se quiere traer los contenidos de **todas** las secciones de una división se coloca * al lado del punto.

- **public**: declara la variable o método (función) como publica, esto es

que cualquier parte del programa puede acceder a ella, se asocia con el concepto de encapsulamiento que veremos en 5.2 Usando POO en java.

- **private**: declara la variable o método como privada, así solo la clase u objeto que lo contiene puede usarlo, se asocia con el concepto de encapsulamiento. Cuando una variable o función no tiene public, private o protected, se toma private por default dento del mismo paquete o "package".
- **protected**: protegido, similar a private, pero los hijos de la clase pueden acceder a esta parte, se asocia con el concepto de encapsulamiento y herencia.
- **static**: estático, establece que solo puede existir una declaración y definición de esta variable o función en el programa.
- **void**: vació, sirve para declarar que no se espera un valor, una función que lo tiene antes indica que es un proceso, ya que realiza acciones pero no regresa un valor al terminar.
- **return**: se coloca antes del valor a regresar, el cual debe ser un tipo de dato similar al que se declaro antes del nombre de la función. Marca la salida de la función.
- **break**: termina antes la secuencia de control en la que se encuentra, se explica mas sobre estas en 4 Secuencias de control básicas.
- **continue**: se brinca a la siguiente iteración de la secuencia de control actual.
- **new**: se utiliza para crear nuevos objetos de una clase.
- **try**: intenta realizar lo que le siga entre paréntesis, este y las siguientes 2 palabras son usadas para funcion o partes de código que pueden causar errores.
- **catch** : atrapa, sirve para declarar que hacer cuando en el código anterior dentro de try tira un error.
- **throw**: aventar, regresa un tipo de error manejable cuando sucede una condición.

Capítulo 4

Secuencias de control básicas

Como se ha ido viendo, programar es hacer instrucciones para la computadora, pero ¿como hacemos que entienda casos o repita acciones? Para esto sirven las secuencias de control. Estas nos permiten, como su nombre lo indica, darle un sentido y seriación a nuestro programa de forma menos lineal. A continuación se presentan las principales de Java que también son comunes en otros lenguajes.

- **if,else** (si,si no): secuencia básica de verdadero o falso, se puede expandir a un "switch" cuando se tendrían muchos if y else, este switch toma la condición a comparar y corre los casos bajo los que la condición sea igual a la respuesta.
- **for** (para): se suele usar para realizar operaciones desde un valor hasta otro, por ejemplo desde 0 hasta 5.
- **while** (mientras): hace algo mientras sea verdad el análisis o la condición dada.
- **do while** (haz mientras): similar a while pero ejecuta las instrucciones al menos 1 vez.

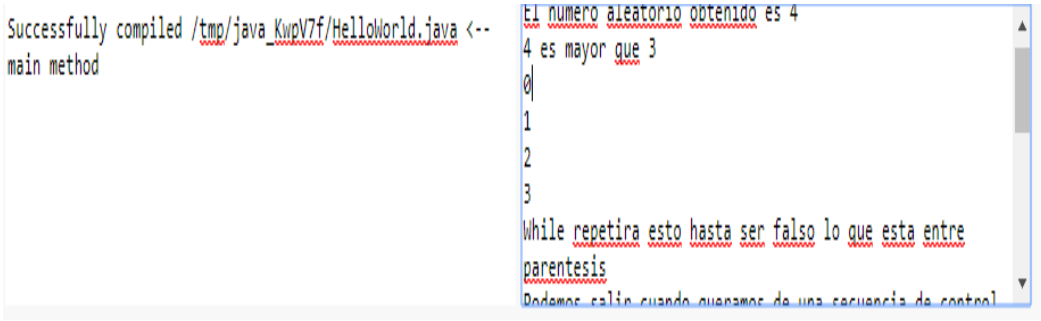
A continuación se presenta un ejemplo donde se implementan estas secuencias de control:

Código 4.1: Secuencias de control

```
1 import java.util.Random; /*Esta es una libreria, la cual contiene
     utilidades para hacer cosas con numeros pseudo-aleatorios */
2
3 public class SecContEje
4 {
```

```
5     public static void main(String[] args)
6     {
7        Random rand = new Random(552); //5523 da 2, 552 da 4
8          int i, var = rand.nextInt(4) +2;
9        System.out.println("El numero aleatorio obtenido es " + var);
10
11      if(var>3)
12         System.out.println(var +" es mayor que 3");
13        else if (var==3) //esta combinacion funciona como un if
14         System.out.println(var +" es igual a 3");
15      else
16           System.out.println(var +" es menor que 3");
17
18      for(i=0;i<var;i++)
19        {
20         System.out.println(i);
21        }
22
23      while(var>3)
24        {
25         System.out.println("While repetira esto hasta ser falso lo
               que esta entre parentesis. ");
26            System.out.println("Podemos salir cuando queramos de una
                  secuencia de control, usando break");
27            break;
28        }
29
30      i=0; //Tenemos que regresar a 0 el entero i.
31      do //Los bucles/loops atoran el programa al dejarlo encerrado
32        { //haciendo las mismas operaciones
33         System.out.println("Las secuencias de control pueden
               causar bucles si no se cuidan");
34            if(i++ >3)
35               break;
36
37        }while(var>3);
38    } //es recomendable revisar que no se tengan loops
39 }
```

Se invita al lector a cambiar el número 552 por el comentado para observar como cambia el código. También se pueden omitir unas y otras secuencias de control así como modificarlas para ganar un mejor entendimiento de las mismas.



Como se menciona en los comentarios del código, los bucles pueden causar problemas inesperados, este caso es sencillo pero existen otros en donde ya no sabemos que pasa con la variable, por ejemplo cuando el usuario debe ingresar el valor, y no tomamos en cuenta algo de lo que llega a escribir.

Capítulo 5

Pensando en código

Hasta el momento ya se han mostrado los componentes fundamentales del lenguaje con los que podemos hacer operaciones y acciones básicas. Ahora se mostraran formas de expandir nuestro conocimiento con las ideas de programación en general, aprovechando el paradigma destacable de Java para generalizar componentes que necesitemos y con los cuales se pueda experimentar aun más fuera de este manual.

5.1. Pseudocódigo

En este manual hemos estado estudiando nada mas el lenguaje de Java, pero que pasa cuando queremos darle indicaciones o algoritmos a una persona que no maneja Java o el mismo lenguaje de programación que nosotros. Para esto se tiene el pseudocódigo como una herramienta que permite concentrarse enteramente en el procedimiento, dejando de lado aquellos detalles que sean más específicos de cada lenguaje.
El pseudocódigo considera una estructura similar al código pero usando un lenguaje más parecido a un instructivo, así el código y/o algoritmo es legible por cualquier persona sin importar que lenguaje de programación use.

Ejemplo:

```
funcionUno(tipo Algo,tipo2 otro)
  tipo extra <- Algo
  if extra = Algo
    then imprime(otro)
return Algo
```

Como se puede observar, el pseudocódigo contempla unos elementos que simplifican la sintaxis y permiten entender mejor las partes y operaciones del

algoritmo.

- Usa flechas para mostrar asignación de un valor a otro, siendo la punta de la flecha el dato que recibe un valor.
- Puede omitir delimitadores del texto, por ejemplo { } o ';', entre otros, pero requiere conservar una estructura y espaciado para dar a entender el contexto y el orden de las operaciones.
- Suele usar espacios o tabulaciones para acomodar el texto y mostrar una jerarquía.
- También puede omitir el tipo de dato, especificándolo solo por el contexto.

Entre otras características que pueden ser especificas del ambiente donde se use.

5.2. Usando POO en java

Como se mencionó anteriormente, java usa el Paradigma Orientado a Objetos para llevar la estructura y lógica del programa. Ahora veamos los elementos que se mencionaron anteriormente en la sección 1.3 Paradigma Orientado a Objetos (POO) y como se ven usados en Java.
Para empezar se enlistan las 4 propiedades importantes del POO:

- **Encapsulamiento**: como se mencionó, consiste en restringir partes del objeto a otros elementos (private, public).
- **Abstracción**: Es la generalización de los elementos a encerrar en *clases* (class).
- **Herencia**: Para no volver a repetir código, se pueden usar las clases ya creadas, esto se puede hacer al *heredarles* a otras clases. Los constructores y destructores se usan pero no se heredan. Los elementos declarados como *private* tampoco se heredan. Los elementos public si.
- **Polimorfismo**: Esto es el dar un uso distinto a una misma función, se puede lograr al *sobreescribir* las funciones heredadas por ejemplo.

Ejemplo de abstraer un objeto que da un mensaje y el encapsulamiento:

Código 5.1: Uso de Objetos en Java

```
1  public class ObjetosJava{
2      public static void main(String[] args){
3         OtroObjeto miObjeto = new OtroObjeto("Hola Mundo!");
4         System.out.print(miObjeto);
5      }
6  }
7
8
9  public class OtroObjeto{
10     private String mensaje;
11     public OtroObjeto(String input){ //Constructor del objeto
12        mensaje = "Objeto Dice: " + input;
13     }
14     /*El siguiente metodo es necesario para poder
15     ser impreso a travez de System.out.print(Objeto)*/
16     public String toString(){
17        return mensaje;
18     }
19 }
```

```
Successfully compiled /tmp/java_QsYnyP/HelloWorld.java <--
main method
Successfully compiled /tmp/java_QsYnyP/OtroObjeto.java
```

```
Objeto dice: Hola Mundo!
```

Ahora veamos un ejemplo con herencia y polimorfismo:

Código 5.2: Herencia y Polimorfismo en Java

```
1  public class Principal{
2      public static void main(String[] args){
3         Padre uno = new Padre("Juan");
4         Hijo dos = new Hijo("Juan");
5         uno.es();
6         dos.es();
7      }
8  }
9
10 public class Padre{
11     private String nombre;
12       public Padre(String input){ //Constructor del objeto
```

```
13          nombre = input;
14      }
15      public void es(){
16          System.out.println("El papa se llama: " + nombre);
17      }
18      public String getNom(){
19          return nombre;
20      }
21  }
22
23  public class Hijo extends Padre{ //extends indica que hereda
24      public Hijo(String input){ //Constructor del objeto
25          super(input + "ito");
26      }
27      public void es(){ //mismo nombre de funcion, texto diferente
28          System.out.println("El hijo se llama: " + getNom());
29      }
30  }
```

Como se menciona en el comentario, la función *es()* lleva exactamente el mismo nombre, parámetros y tipo (public void), sin embargo, lo que hace es distinto que la función padre. Y al correr el código el hijo usara la función de *es()* que define. Se deja como ejercicio borrar la función en el hijo y correr el código para observar como desaparece el polimorfismo.

5.3. Memoria

A veces conocemos la cantidad de información a guardar, en otras ocasiones no, pero al menos es importante entender la naturaleza de la misma para poder administrarla.
Para esto existe:

- La memoria estática: La cual es común cuando conocemos la cantidad de información y su máximo a utilizar. Suele usarse en forma de "arreglos" (arrays), las cadenas de caracteres, y los conjuntos de números son los más populares. Pueden hacerse arreglos con corchetes [] después del tipo de variable, colocando un número para dar la cantidad máxima a usar, o dejando los corchetes pero definiendo en su momento todos los elementos que irán.

- La memoria dinámica: Esta se usa cuando desconocemos la cantidad, sirve para pedir en el momento que lo requerimos espacio de almacena-

miento, tiene mucho uso gracias a lo que su nombre indica. Un ejemplo que hemos estado usando es el *String*, este es un arreglo de caracteres dinámico, esto es, le podemos quitar y/o agregar sin preocuparnos de haber definido una cantidad máxima de letras.

Una mezcla de estos dos tipos de memoria es el *String[] args* que se ha estado usando en las funciones *main* de cada programa. Donde se esta diciendo que se toma un arreglo de Strings, o un arreglo de arreglos de caracteres, es útil definir nombres y usos para evitar confusiones, como se puede notar en esta ultima forma de llamar a *String[] args*.

Capítulo 6

Sección Yeyecoa

Lo aprendido hasta ahora han sido elementos básicos y necesarios para empezar a programar y encaminarse en Java. Sin embargo hay algunos temas que pueden ser de utilidad, como a continuación se presentan algunos. Si ya te sientes listo para hacer retos avanza a 6.2 Ahora Yeyecoa.

6.1. Secuencias de escape

Algunos caracteres que tienen una diagonal ("\") antes tienen un significado para el compilador, a estos se les llaman "secuencias de escape", y representan cosas distintas, como se muestra en la siguiente tabla:

Secuencia de escape	Descripción
\t	Inserta una tabulación o tab en el texto en este punto.
\b	Inserta un backspace en la posición donde esta el texto.
\n	Inserta una nueva linea o "enter".
\r	Inserta un "retorno de carro", solía ser para regresar el marcador del texto al inicio, pero en tiempos modernos a cambiado un poco su funcionalidad, siendo inútil o similar a un \n en algunos compiladores o entornos.
\f	Inserta un "formfeed", sirve para limpiar la pantalla en consola.
\'	Inserta una comilla simple ' en donde se escribe la secuencia.
\"	Inserta una comilla doble " en donde se escribe la secuencia.
\\	Inserta un \en el lugar donde se escribe la secuencia.

Tabla 6.1: Secuencias de escape [1]

Es de notar que estos toman significado cuando se trata de procesar texto, principalmente se han mostrado en ejemplos con la función "*System.out.print( )*" o "*System.out.println( )*" ,donde la primera solo escribe el texto que se le manda, mientras que la segunda anexa un "\n" al final del texto ingresado entre paréntesis.

6.2. Ahora Yeyecoa

Para practicar lo aprendido, realiza las siguientes actividades, si te sientes atorado en alguna revisa la respuesta, aunque no es necesario que sean idénticos los códigos para tener los mismos resultados, *inclusive puede hacer código mucho mejor*! Los ejercicios **yeyecoa** son algo especiales ya que requieren pensar más en la solución, pero te permitirán perfeccionar tu conocimiento.

Práctica 1: Busca en linea los valores en ASCII de los caracteres que ne-

[1] Se usa la tabla de la compañía Oracle [**?**].

cesites, para que así asignándolos a varios datos tipo char puedas imprimir "Hola Mundo!"

Respuesta 1:

Yeyecoa 2: Usando secuencias de control, imprime una pirámide hecha con * de al menos 5 niveles.

Respuesta 2:

Práctica 3: Usando secuencias de control, calcula el factorial de un número, considera el factorial de 0 y 1 como que vale 1. *El factorial de un número es la multiplicación del mismo por el número anterior a este, así sucesivamente*

hasta llegar a 1.

Respuesta 3:

Práctica 4: Usando secuencias de control. Escribe en pantalla que letra denomina la calificación de un alumno, siendo que más de 90 puntos es "A", entre 79 y 91 es "B", entre 69 y 80 es "C", entre 59 y 70 es "D", menos de 60 es F. Para el valor de la calificación escribe una variable

Respuesta 4:

Yeyecoa 5: Calcula el Máximo Común Divisor entre 2 números. *Pista: usa el operador % y compáralo a 0 para revisar si un número es divisible, aprovecha también la secuencia de control for y el operador lógico* &&.
El Máximo Común Divisor (MCD) es el número mas grande que puede dividir a 2 números sin dejar residuo.

Respuesta 5:

Código 6.5: Respuesta del ejercicio 5

```
1  public class EjercicioCinc
2  {
3      public static void main(String[] args)
4      {
5          int a=24, b=54;
6          int menor = (a<b ? a : b);
7          int mcd=1;
8
9          for(int i=2; i<= menor; i++)
10             if(a%i == 0 && b%i == 0)
11                 mcd =i;
12
13         System.out.print(mcd);
14     }
15 }
```

No te detengas con estos ejercicios, busca ejemplos o casos de la vida real que se te podrían facilitar si emplearas lo aprendido de programación para realizar las operaciones y/o cálculos que veas necesarios. Si te llamaron la atención estos ejercicios revisa la sección 7.1 Por diversión para obtener sugerencias de donde poder practicar más.

Capítulo 7

Expandiendo el conocimiento

El manual ha llegado a su fin, pero el camino de la programación apenas comienza. Puedes aprovechar lo aprendido de distintas formas que te llamen la atención, y en caso de querer conocer que más estudiar se presentan unas recomendaciones para encaminarse en la profundización sobre este lenguaje.

7.1. Por diversión

La programación es tanto una habilidad como un arte así como lo son el dibujar, el cantar, etc. Por lo cuál requiere de práctica y dedicación para desarrollarla. Para esto se pueden buscar mas recursos y fuentes de los cuales aprender y ponerse a prueba, por ejemplo:

- Aprender y/o practicar el inglés, ya que muchos recursos, y problemas se encuentran así, sin mencionar que las cosas básicas también están escritas en inglés.
- Resolver problemas que requieran programación, para esto existen sitios y aplicaciones como: Dcoder (App para android), projectEuler (https://projecteuler.net/archives), codechef (https://www.codechef.com/), entre otros.

7.2. Por profesión

Para perseguir la programación como carrera o profesión son convenientes:

- Cursos en distintos lenguajes, en la práctica siempre es útil conocer mas de un solo lenguaje de programación.

- Convertir ideas en creaciones, muchos problemas y aprendizajes se obtienen de intentar manifestar las ideas en realidades, además de que así se han generado muchas carreras, para esto es recomendable aprender diseño de software.

- Libros y guías , la editorial O'Reilly es popular por tener libros completos sobre la mayoría de lenguajes, incluyendo Java.

7.3. Sugerencias

Existen muchos medios y fuentes de los cuales aprender el lenguaje Java, solo que la mayoría están en inglés, lengua que también se sugiere aprender para poder aprovechar muchos de los recursos disponibles, algunos que se recomiendan para tener mejor entendimiento y habilidades de Java se mencionan a continuación:

- **Sams Teach Yourself Java in 24 Hours**: si bien es un libro muy corto (en inglés), es bueno para dar una mirada mas rápida y amplia a los usos de Java, desde código hasta su uso en aplicaciones y juegos.

- **Tutoriales de Java:** La empresa Oracle tiene tutoriales muy completos sobre su lenguaje, se encuentran en inglés disponibles en su sitio: https://docs.oracle.com/javase/tutorial/. En la pagina también se encuentra un enlace a Amazon de los tutoriales en forma de libro.

- **Android Studio** es un entorno de programación en Java específicamente enfocado a programar en Android, existen cursos completamente dedicados a esto.

Motivación final

Estimado lector,

Felicitaciones por haber concluido este manual introductorio a la programación con Java. Programar en cualquier lenguaje es un paso valiente y grande para introducirse a la programación en general, de ahí en adelante dependerá mucho de los gustos particulares para desarrollarse, ya sean estos gustos por lo que se desea realizar, o por el lenguaje y sus herramientas.

Al final, la pasión y dedicación son los que traerán los resultados de aplicar los conocimientos y habilidades hacia las actividades, trabajos e ideas que se elaboren. Es por esto que se invita al lector ha buscar metas que realizar con los conocimientos aprendidos, así podrá ir obteniendo mayor experiencia.

También se destaca que el programar y su aprendizaje es un proceso laboreoso y que puede contener dificultades, pero todo esto es parte del proceso de aprendizaje, a lo cual se recomienda no rendirse al encontrarse con problemas u obstáculos, y animarse a preguntar o buscar en linea por soluciones que tal vez no le pares can obvias al inicio, pero posiblemente alguien mas ya haya tenido la duda o el error al que usted se enfrente.

Sin mas que agregar se le desea buena suerte.
-Grupo de Computo Cuántico y Científico de la Facultad de Estudios Superiores Acatlan .

Este documento fue creado gracias al PAPIME PE104919.