



I. OBJETIVOS.

- Que el estudiante conozca más sobre Java y el IDE NetBeans.
- Que el estudiante se familiarice con las restricciones de IDE NetBeans.

II. INTRODUCCIÓN.

¿Qué es NetBeans?

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios (¡y subiendo!) en todo el mundo. SunMicroSystems fundó el proyecto de código abierto NetBeans en junio 2000 y continúa siendo el patrocinador principal de los proyectos.

A día de hoy hay disponibles dos productos: el NetBeans IDE (Entorno de Desarrollo Integrado) y el NetBeans Plataforma.

El NetBeans IDE: Es un entorno de desarrollo - una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java - pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans. El IDE NetBeans es un producto libre y gratuito sin restricciones de uso.

También disponible está el NetBeans Plataforma: Una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio grandes. Empresas independientes asociadas, especializadas en desarrollo de software, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones.

Ambos productos son de código abierto y gratuito para el uso tanto comercial y como no comercial. El código fuente está disponible para su reutilización de acuerdo con la CommonDevelopment and DistributionLicense (CDDL).

¿Qué es netbeans.org?

NetBeans.org es el portal de la comunidad de código abierto de NetBeans dedicado a construir un IDE de primera clase. netbeans.org permite a usuarios de más de 160 países de todo el mundo estar en contacto con los recursos y los programadores de NetBeans. Es posible descargar desde aquí las últimas versiones de NetBeans, acceder a la documentación de ayuda en línea, profundizar su conocimiento personal de Java, estar al corriente de las últimas noticias, unirse a una lista de distribución, contribuir código, conocer las personas implicadas en el proyecto, conocer gente, y mucho más.

Historia:

NetBeans comenzó como un proyecto estudiantil en Republica Checa (originalmente llamado **Xelfi**), en 1996 bajo la tutoría de la Facultad de Matemáticas y Física en la Universidad de Charles en Praga. La meta era escribir un entorno de desarrollo integrado (IDE) para Java parecida a la de Delphi. Xelfi fue el primer entorno de desarrollo integrado escrito en Java, con su primer pre-reléase en 1997.

Xelfi fue un proyecto divertido para trabajar, ya que las IDEs escritas en Java eran un territorio desconocido en esa época. El proyecto atrajo suficiente interés, por lo que los estudiantes, después de graduarse, decidieron que lo podían convertir en un proyecto comercial. Prestando espacios web de amigos y familiares, formaron una compañía alrededor de esto. Casi todos ellos siguen trabajando en NetBeans.

Tiempo después, ellos fueron contactados por **RomanStanek**, un empresario que ya había estado relacionado con varias iniciativas en la Republica Checa. Él estaba buscando una buena idea en que invertir, y encontró en Xelfi una buena oportunidad. Ellos se reunieron, y el negocio surgió.

El plan original era desarrollar unos componentes JavaBeans para redes. **JardaTulach**, quien diseñó la arquitectura básica de la IDE, surgió con la idea de llamarlo NetBeans, con el fin de describir lo que ellos harían. Cuando las especificaciones de los Enterprise JavaBeans salieron, ellos decidieron trabajar con este estándar, ya que no tenía sentido competir con él, sin embargo el nombre de NetBeans se quedó.

En la primavera de 1999, **NetBeans DeveloperX2** fue lanzado, soportando Swing. Las mejoras de rendimiento que llegaron con el JDK 1.3, lanzado en otoño de 1999, hicieron a NetBeans una

alternativa realmente viable para el desarrollo de herramientas. En el verano de 1999, el equipo trabajó duro para rediseñar a DeveloperX2 en un NetBeans más modular, lo que lo convirtió en la base de NetBeans hoy en día.

Algo más paso en el verano de 1999. Sun Microsystems quería una mejor herramienta de desarrollo de Java, y comenzó a estar interesado en NetBeans. En otoño de 1999, con la nueva generación de NetBeans en Beta, el acuerdo fue realizado.

Sun adquirió otra compañía de herramientas al mismo tiempo, **Forté**, y decidió renombrar NetBeans a **Fortéfor Java**. El nombre de NetBeans desapareció de vista por un tiempo.

Seis meses después, se tomó la decisión de hacer a NetBeans en **Open Source**. Mientras que Sun había contribuido considerablemente con líneas de código en varios proyectos de código abierto a través de los años, NetBeans se convirtió en el primer proyecto de código abierto patrocinado por ellos. En Junio del 2000 NetBeans.org fue lanzado.

Instalar NetBeans:

La instalación de NetBeans es muy sencilla. En la mayoría de los casos, basta con descomprimir el archivo ZIP descargado en una unidad de disco con espacio libre suficiente. Para ejecutar NetBeans necesitará cumplir los requisitos técnicos detallados a continuación.

Componente	Características
Una plataforma compatible: NetBeans IDE 8 o Superior	Windows 2000/XP/Vista/7 u 8; Linux (x86/x64); Solaris (x86/x64); Solaris (sparc); Mac OS X; OS independent.
Espacio de disco suficiente:	Dependiendo de la versión que desea instalar la que menos utiliza es de 22 MB y la completa de 140 MB como requerimiento MINIMO.
Memoria RAM Suficiente:	Debería bastar 512 MB
Java JDK 9	jdk-9u4-windows-i586-p.exe de capacidad de 72 MB.

Fundamento de un entorno típico en Java.

La siguiente explicación define los pasos típicos para crear y ejecutar un programa en Java, utilizando los entornos de desarrollo de Java. Estos pasos se aplican en el dibujo mostrado más abajo y se explican en este texto.

Por lo general, los programas en Java pasan a través de cinco fases para poder ejecutarse como se muestra en la figura 1.1. Estas fases son: Edición, Compilación, Carga, Verificación y Ejecución.

La fase 1: Consiste en editar un archivo. Esto se logra mediante un programa de edición (el bloq de notas de Windows, el vim de Linux, o el que más te guste). El programador escribe un programa en Java utilizando el editor, y realiza las correcciones, si es necesario. Cuando un programador determina que el programa está listo, guarda el programa en un archivo con la extensión **.java**.

La fase 2: El programador proporciona el comando **javac** para compilar el programa. El compilador de Java traduce el programa a código de bytes: las instrucciones que el intérprete de Java (o máquina virtual) puede entender. Por ejemplo, para compilar un programa llamado **Hola.java**, escriba:

javac Hola.java

En la ventana de comandos de su sistema. Si el programa se compila correctamente, el compilador genera un archivo **Hola.class**. Este archivo contiene códigos de bytes que se interpretan en la fase de ejecución.

La fase 3: Se conoce como carga. El programa debe colocarse en memoria antes de ejecutarse. De esto se encarga el cargador de clases, que toma los archivos **.class** que contienen los códigos de bytes y los transfiere a la memoria principal. Después, el archivo **.class** puede cargarse desde un disco en un sistema o a través de una red.

A medida que se cargan las clases, el verificador de códigos de bytes se encarga de verificar sus códigos de bytes.

La fase 4: Este proceso de verificación asegura que los códigos de bytes de las clases sean válidos y que no violen las restricciones de seguridad de Java. Java implementa una estrecha seguridad, ya que los programas en Java que llegan a través de la red no deben ser capaces de dañar sus archivos o su sistema (como podrían hacerlo un virus de computadora). Tome en cuenta que la verificación del código de bytes también se lleva a cabo en aplicaciones que descargan clases a través de una red.

Finalmente en **la fase 5**, el intérprete, bajo el control del sistema operativo, interpreta el programa un código de bytes a la vez, realizando en esta forma las acciones especificadas por el programa.

Existen dos tipos de programas para los cuales ocurre este proceso: **las aplicaciones y los applets**.

Ver figura 1.1

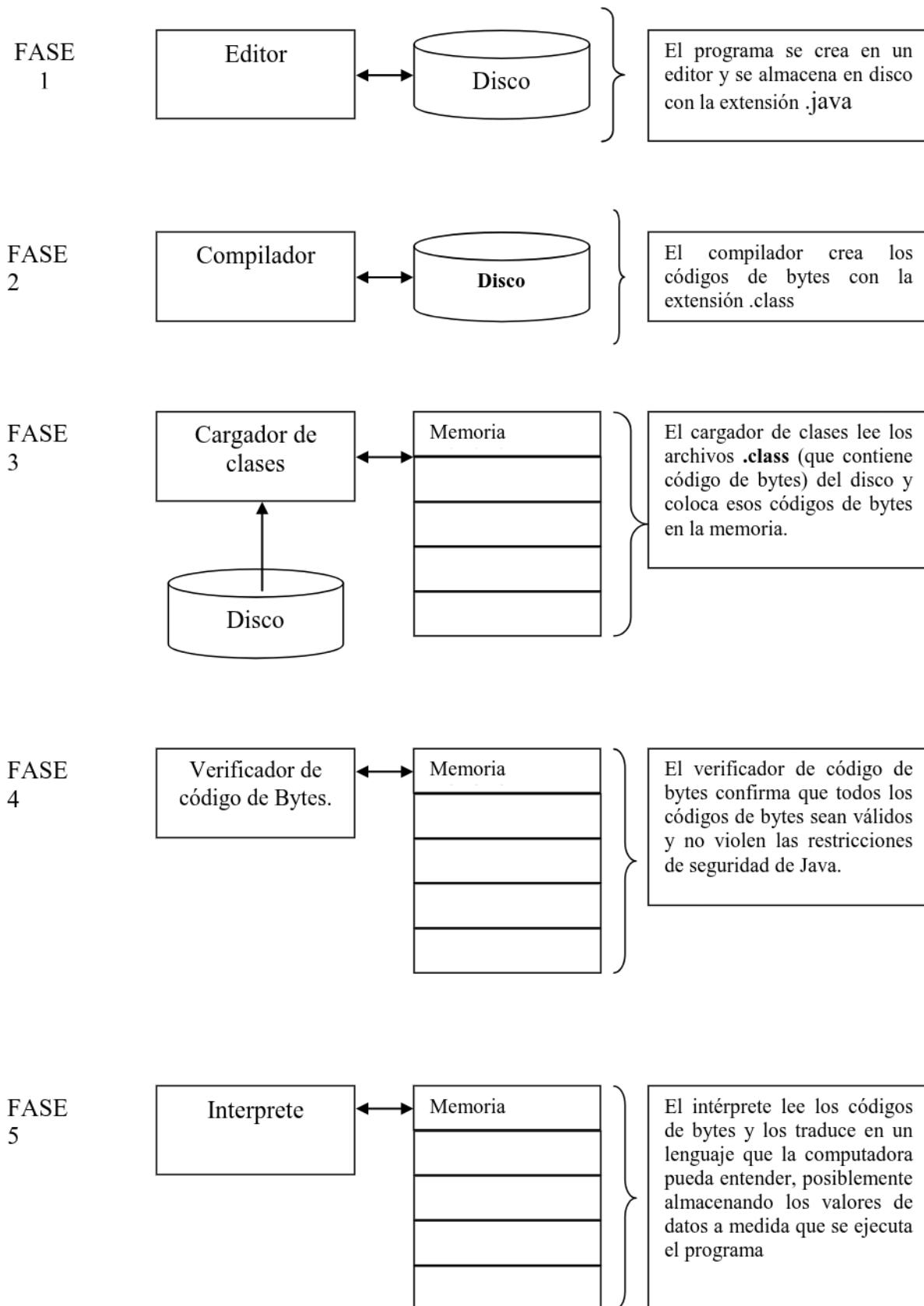


Fig. 1.1 Entorno típico en Java.

Una aplicación es un programa (como por ejemplo un programa de procesamiento de palabras, de hojas de cálculo, de dibujo o de correo electrónico) que generalmente se guarda y ejecuta desde el equipo local de usuario.

Un applet es un pequeño programa que se guarda en un equipo remoto, al cual se conectan los usuarios a través de un navegador Web. El equipo remoto se conoce como un servidor Web. Los applets se cargan desde un equipo remoto en el navegador, se ejecutan en éste y se descartan cuando termina la ejecución. Para ejecutar un applet de nuevo, el usuario debe dirigir un navegador a la ubicación apropiada en Internet y volver a cargar el programa en el navegador.

Las aplicaciones se cargan en memoria y se ejecutan, utilizando el intérprete de Java mediante el comando `java`. Al ejecutar una aplicación de Java llamada, por ejemplo, `Hola.class`, el comando

java Hola

Invoca al intérprete para la aplicación `Hola`, y hace que el cargador de clases cargue la información utilizada en el programa `Hola`. **[Nota: El intérprete de Java se conoce también como la Máquina Virtual, o la JVM (Java Virtual Machine)].**

Los navegadores Web como el **FireFox o Internet Explorer** se utilizan para ver los documentos en Internet, que generalmente son archivos HTML. Un documento HTML puede contener un applet de Java. Cuando el navegador ve la referencia (las etiquetas) a un applet en un documento HTML, inicia el cargador de clases de Java para cargar el applet. Todos los navegadores que cuentan con soporte para Java tienen un intérprete de Java incluido (es decir, una JVM). Una vez que se carga el applet, el intérprete de Java lo ejecuta. Los applets también pueden ejecutarse desde una línea de comandos, mediante el comando **appletviewer** que se incluye con el J2SDK. Al igual que un navegador, el **appletviewer** requiere un documento HTML para poder invocar al applet. Por ejemplo, si el archivo `Hola.html` hace referencia al applet `Hola`, el comando **appletviewer** se utiliza de la siguiente manera:

`appletviewer Hola.html`

Este comando ocasiona que el cargador de clases cargue la información utilizada en el applet `Hola`. El **appletviewer** es un navegador muy básico; solo sabe cómo interpretar referencias a los applets e ignora el resto de código HTML del documento.

Nota: el compilador de Java, `javac`, no es un compilador tradicional en cuanto a que no convierte un programa de Java, de código fuente a código máquina nativo para una plataforma

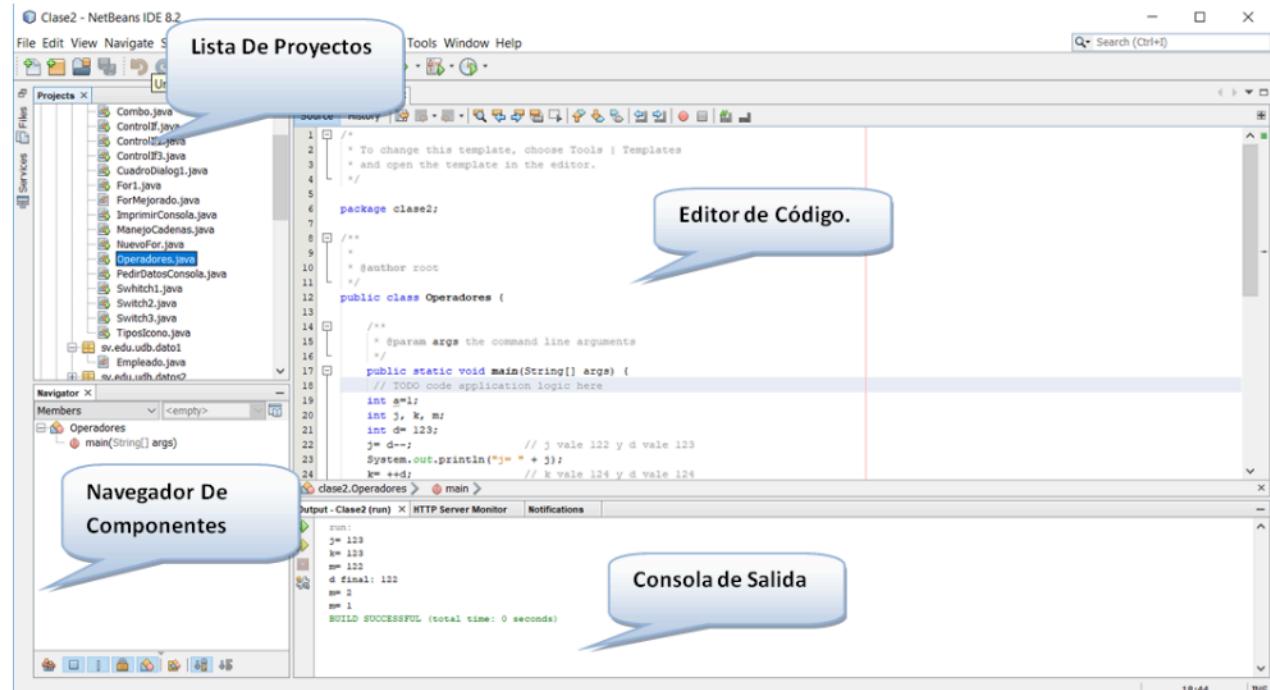
computacional específica. En vez de ello, el compilador de Java traduce el código fuente en código de bytes; el lenguaje de la Máquina Virtual de Java (JVM). La JVM es un programa que simula la operación de una computadora y ejecuta su propio lenguaje máquina (es decir, código de bytes de Java).

III. Procedimiento.

IDE de trabajo NetBeans IDE 8 o Superior

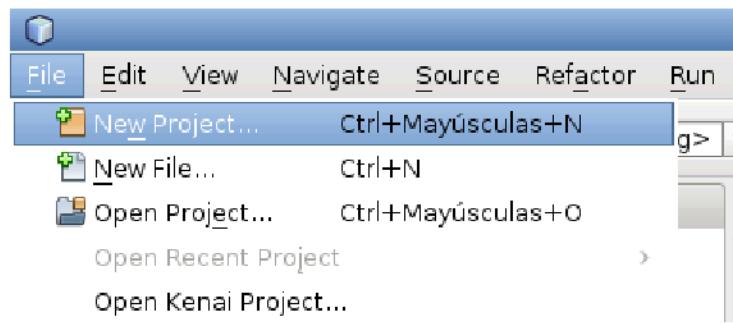


Pantalla principal al Iniciar NetBeans IDE 8.2

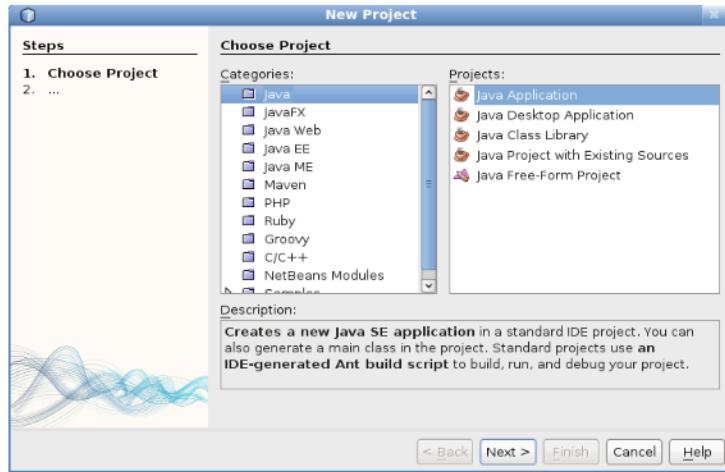


Abriendo un Nuevo Proyecto:

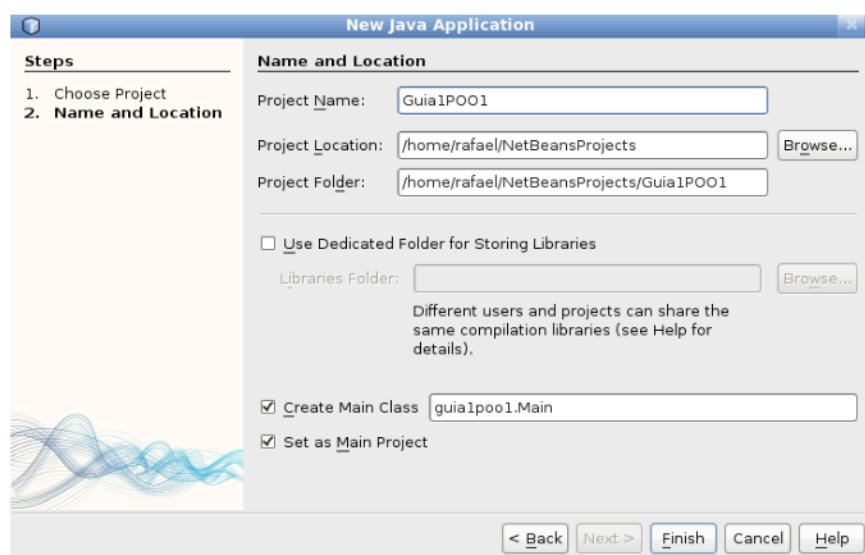
- En el IDE, seleccione File> New Project, tal como se muestra en la siguiente figura.



- En el asistente de New Project, ampliar la categoría de Java y seleccione Java Application, como se muestra en la siguiente figura. A continuación, haga clic en Next.



- En el Nombre y ubicación de la página del asistente, haga lo siguiente (tal y como se muestra en la siguiente figura):
 - * En el campo Nombre del proyecto (Project Name), es Guia1.
 - * En el campo Crear la clase principal (CreateMainClass), es guia1.Main.



- Haga clic en Finalizar (Finish).

El proyecto es creado y abierto en el IDE. Debe ver los siguientes componentes:

- La ventana de Proyectos, que contiene una vista en árbol de los componentes del proyecto, incluyendo archivos de origen, las bibliotecas que depende de su código, y así sucesivamente.
- La Fuente Editor ventana con un archivo llamado Main.java abierto.
- La ventana del Navegador, que usted puede utilizar para navegar rápidamente entre los elementos seleccionados dentro de la clase.

The screenshot shows the Eclipse IDE interface. On the left, the 'Projects' view displays a tree of Java projects. One project, 'Guia1', is expanded, showing its source packages ('guia1' containing 'Guia1.java' and 'LecturaParametros.java'). Other projects listed include 'Guia13POOII', 'Guia1POO2', 'Guia2Poo2', 'Guia3LP3', 'Guia6POO1', 'Guia7DFWV1', 'Guia7POO', 'Guia8POO1', 'Guia9POO1', 'GuiaDFW5', 'hopacapp', 'InventarioLibros_MVC', and 'Ireport5.6'. On the right, the 'Guia1.java' source editor is open. The code is as follows:

```
/*
 * To change this license header, choose License Headers in
 * Project Properties. To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package guial;

/**
 *
 * @author User
 */
public class Guia1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

Ejercicio 1:

A continuación agregara la siguiente línea de código.

```
// guia1.java
```

Esta línea la agregaremos antes de la declaración de la clase pública.

The screenshot shows the 'Guia1.java' source editor again. A new line of code, 'System.out.println("Hola Mundo");', has been added at the end of the main method, just before the closing brace. The code now looks like this:

```
/*
 * To change this license header, choose License Headers in
 * Project Properties. To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package guial;

/**
 *
 * @author rafael.torres
 */
public class Guia1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println("Hola Mundo");
    }
}
```

```
System.out.println("Hola Mundo");
```

Esta línea la agregaremos dentro del **main** de nuestra clase.

Explicación del Código:

- // guia1.java

Esta porción de código indica un comentario de línea al estilo de c++. Los programadores insertan comentarios para documentar los programas y mejorar la legibilidad de estos. Los comentarios también ayudan a otras personas a leer y comprender un programa. El compilador de Java ignora estos comentarios, de manera que la computadora no hace nada cuando el programa se ejecuta. Los programadores usan líneas en blanco y espacios para facilitar la lectura de los programas.

- public class Main {

Comienza la declaración de una clase, para la clase Main. Todo programa en Java consiste de, cuando menos, una declaración de clase que usted, el programador, debe definir. Estas clases se conocen como clases definidas por el programador o clases definidas por el usuario. La palabra clave **class** introduce una declaración de clase en Java, la cual debe ir seguida inmediatamente por el nombre de la clase.

Por convención, todos los nombres de clases en Java comienzan con una letra mayúscula, y la primera letra de cada palabra en el nombre de la clase debe ir en mayúscula (por ejemplo, **EjemploDeNombreDeClase**). El nombre de la clase se conoce como identificador, y está compuesto por una serie de caracteres que pueden ser letras, dígitos, guiones bajos (_) y signos de moneda (\$); sin embargo, no puede comenzar con un dígito ni tener espacios.

Buena práctica de programación: Cuando lea un programa en Java, busque identificadores en los que la primera letra del mismo esté en mayúscula. Dichos identificadores generalmente representan clases.

Observación de Ingeniería de Software: Evite usar identificadores que contengan signos de moneda (\$). El compilador comúnmente utiliza estos signos para crear nombres de identificadores.

Error común de programación: Java es sensible a mayúsculas y minúsculas. El no utilizar la combinación apropiada de letras minúsculas y mayúsculas para un identificador, generalmente produce un error de compilación.

Cuando usted guarda su declaración de clase **public** en un archivo, el nombre de éste debe ser el nombre de la clase, seguido de la extensión “**.java**”. Para nuestra aplicación, el nombre del

archivo es **Main.java**. Todas las extensiones de clases en Java se guardan en archivos que terminan con la extensión “**.java**”.

Error común de programación: Tanto en términos de ortografía como en ahorro de tiempo, es un error que una clase **public** tenga un nombre de archivo que no sea idéntico al nombre de la clase (más la extensión **.java**). Por lo tanto, es también un error que un archivo contenga dos o más clases **public**.

Error común de programación: Si un archivo contiene la declaración de una clase, es un error que no finalice con la extensión **.java**. Si se omite esa extensión, el compilador de java no podrá compilar la declaración de la clase.

- public static void main (String args[])

Es el punto de inicio de toda aplicación Java. Los paréntesis después de **main** indican que éste es un bloque de construcción del programa, al cual se le llama método. Las declaraciones de clases en Java generalmente contienen uno o más métodos. En una aplicación en Java, sólo uno de esos métodos debe llamarse **main** y debe definirse como se muestra anteriormente; de no ser así, el intérprete java no ejecutará la aplicación. Los métodos pueden realizar tareas y devolver información una vez que las hayan concluido. La palabra clave **void** indica que este método realizará una tarea (en el caso del ejemplo anterior, mostrará una línea de texto), pero no devolverá ningún tipo de información cuando complete su tarea. Más adelante veremos que muchos métodos devuelven información cuando finalizan sus tareas.

La llave izquierda “{“, comienza el cuerpo de la declaración del método. Su correspondiente llave derecha “}” debe terminar el cuerpo de la declaración del método.

- System.out.println("BienvenidoaNetBeans IDE 7");

Indica a la computadora que realice una acción es decir, que imprima la cadena de caracteres contenida entre los caracteres de comillas dobles. A una cadena también se le denomina cadena de caracteres, mensaje o literal de cadena. Genéricamente, nos referimos a los caracteres entre comillas dobles como cadenas. El compilador no ignora los caracteres de espacio en blanco dentro de las cadenas.

Es un error de sintaxis que una cadena no aparezca entre caracteres de comillas dobles, en una línea de programa.

System.out se conoce como el objeto de salida estándar. **System.out** permite a las aplicaciones en Java mostrar conjuntos de caracteres en la ventana de comandos, desde la cual se ejecuta la aplicación en Java.

El método **System.out.println** muestra (o imprime) una línea de texto en la ventana de comandos. La cadena dentro del paréntesis es el argumento para el método. El método **System.out.println** realiza su tarea, mostrando su argumento en la ventana de comandos. Cuando **System.out.println** completa su tarea, posiciona el cursor de salida al principio de la siguiente línea en la ventana de comandos.

Algunos programadores se les dificulta, cuando leen o escriben un programa, relacionar las llaves izquierda y derecha ({,}) que delimitan el cuerpo de la declaración de una clase o de un método. Por esta razón, incluyen un comentario de fin de línea después de una llave derecha de cierre (}) que termina la declaración de un método y que termina la declaración de una clase.

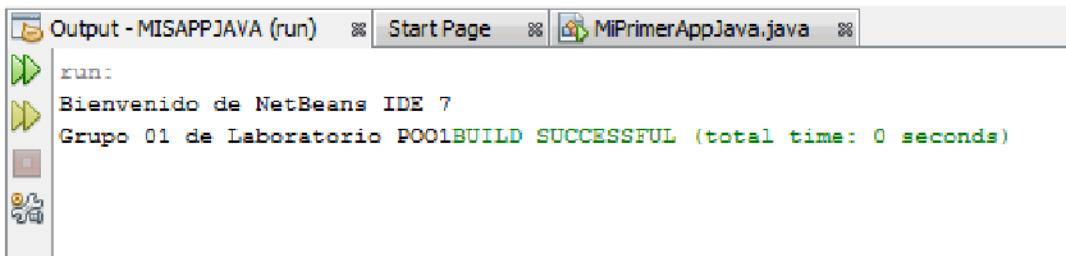
Cómo mostrar una sola línea de texto con varias instrucciones

- System.out.print("Grupo 01 de Laboratorio ");
- System.out.println("Bienvenido a NetBeans IDE 7");

Estas nuevas líneas del método **main** muestran unas líneas de texto en la ventana de comandos. La primera instrucción utiliza el método **print** de **System.out** para mostrar una cadena. A diferencia de **println**, después de mostrar su argumento, **print** no posiciona el cursor de salida al inicio de la siguiente línea en la ventana de comandos; el siguiente carácter que muestra el programa en la ventana de comandos aparecerá inmediatamente después del último carácter que muestre **print**. Cada instrucción **print** o **println** continúa mostrando caracteres a partir de donde la última instrucción **print** o **println** dejó de mostrar caracteres.

```
public class Main {  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        // TODO code application logic here  
        //System.out.print("Grupo 01 de Laboratorio ");  
        System.out.println("Bienvenido a NetBeans IDE 7");  
        System.out.print("Grupo 01 de Laboratorio POO1");  
    }  
}
```

Salida en la ventana de comandos

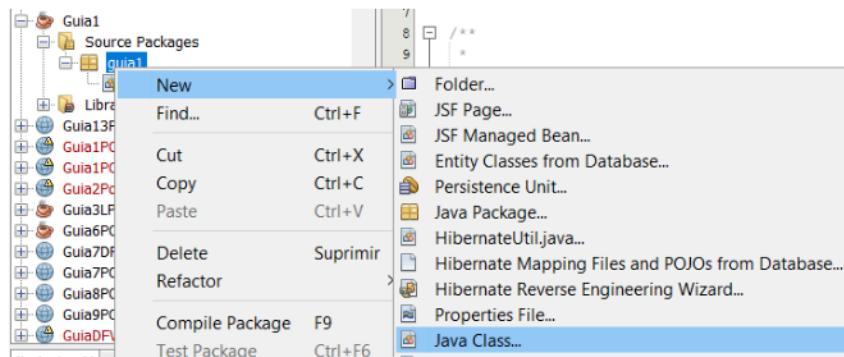


The screenshot shows the NetBeans IDE interface with the 'Output' tab selected. The log output is as follows:

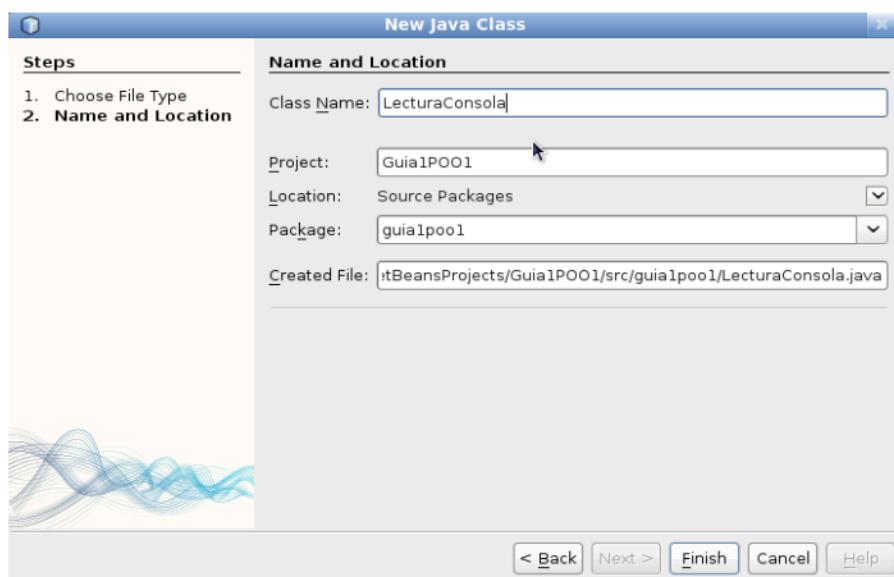
```
run:
Bienvenido de NetBeans IDE 7
Grupo 01 de Laboratorio POOLBUILD SUCCESSFUL (total time: 0 seconds)
```

Cómo leer datos por Consola

- Crear una nueva clase dando click derecho sobre el paquete **guia1**,



- elegimos **New** y luego la opción “**Java Class...**”, aparecerá una ventana como la siguiente.

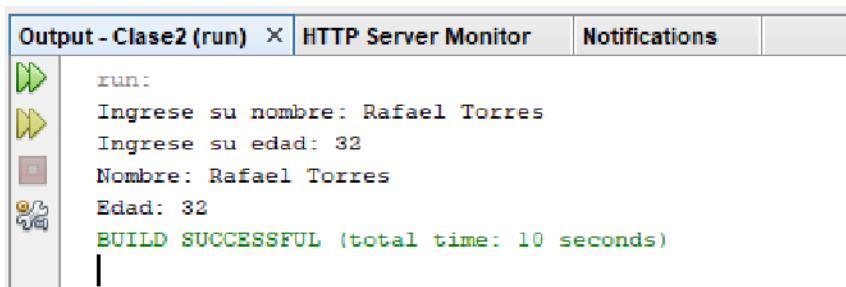


- En el campo “**ClassName**” ingresar **LecturaConsola**.

- Ingresar el siguiente código.

```
import java.util.*;  
  
public class LecturaConsola {  
    public static void main(String[] args){  
        Scanner reader = new Scanner(System.in);  
        String nombre = "";  
        int edad;  
  
        System.out.print("Ingrese su nombre: ");  
        nombre=reader.nextLine();  
        System.out.print("Ingrese su edad: ");  
        edad=reader.nextInt();  
  
        System.out.println("Nombre: "+nombre);  
        System.out.println("Edad: "+edad);  
    }  
}
```

- Salida en la ventana de comandos



```
Output - Clase2 (run) × HTTP Server Monitor Notifications  
run:  
Ingresé su nombre: Rafael Torres  
Ingresé su edad: 32  
Nombre: Rafael Torres  
Edad: 32  
BUILD SUCCESSFUL (total time: 10 seconds)
```

Explicación de la clase Scanner

La utilización de la clase Scanner es muy sencilla. Lo primero que tenemos que hacer es declarar un objeto Scanner instanciándolo contra la consola, es decir, contra el objeto **System.in**.

Scanner reader = new Scanner(System.in);

Ahora, para leer lo que el usuario está introduciendo por la consola deberemos de utilizar el método **nextLine**. Este nos devolverá los caracteres que encuentre en la consola hasta encontrarse un retorno de carro. El valor se lo asignaremos a una variable String.

nombre=reader.nextLine();

Ejercicio 2:

¿Cómo mostrar texto en un cuadro de diálogo?

El primer ejercicio, muestra la salida de un programa en la ventana de comandos. Y ahora, en el segundo ejemplo vamos a mostrar la salida de un programa en un **cuadro de dialogo**.

Generalmente, los cuadros de diálogo son ventanas en las que los programas muestran mensajes importantes a los usuarios del programa. La clase **JOptionPane** de Java proporciona cuadros de diálogo previamente empaquetados, los cuales permiten a los programadores mostrar ventanas que contengan mensajes para los usuarios.

Uno de los puntos fuertes de Java es su extenso conjunto de clases predefinidas que los programadores pueden usar, en vez de tener que “reinventar la rueda”. A lo largo del ciclo emplearemos muchas de esas clases. Las numerosas clases predefinidas de Java se agrupan en categorías de clases relacionadas, conocidas como paquetes. Un paquete es una colección de clases con nombre; éstos se conocen colectivamente como la biblioteca de clases de Java, o la interfaz de programación de aplicaciones de Java [API (Por sus siglas en inglés) de Java]. Los paquetes del API de Java se dividen en básicos y opcionales. Los nombres de la mayoría de los paquetes del API de Java comienzan, ya sea con “Java” (paquetes básicos) o “Javax” (paquetesopcionales). Aunque algunos nombres de paquetes en el API de Java comienzan con org.). Muchos de ellos se incluyen como parte del Kit de desarrollo de software para Java 2. A medida que Java continua evolucionando, la mayoría de los nuevos paquetes se desarrollan como paquetesopcionales. Estos a menudo pueden descargarse de java.sun.com y se utilizan para mejorar las capacidades de Java. En este ejemplo utilizamos la clase predefinida de Java **JOptionPane**, la cual se encuentra en el paquete **javax.swing**.

- **Paso 1**

Agregue una nueva clase al proyecto, con el nombre de **CuadroDialogo**. Para hacer esto haga clic derecho en el paquete **guia1poo1** y seleccione la opción **New** y después **Java Class...**

```
import javax.swing.JOptionPane;
public class CuadroDialogo {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        JOptionPane.showMessageDialog(null, "Bienvenido a NetBeans IDE 7");
        System.exit(0);
    }
}
```

Ejecute el código (Clic derecho en el nombre de la clase y escoge la opción **Run File**) y vera como salida el siguiente cuadro de dialogo:



Explicación del Código

La línea

- Import javax.swing.JOptionPane;

Es una declaración **import**. Los programadores utilizan declaraciones **import** para identificar las clases predefinidas que se utilizan en un programa en Java. El compilador trata de cerciorarse de que usted utilice correctamente las clases del API de Java. Las declaraciones **import** ayudan al compilador a localizar las clases que usted desea utilizar. Por cada nueva clase que utilizamos del API de Java, debemos indicar el paquete en el que se encuentra esa clase. Esta información sobre el paquete es importante, ya que le ayuda a localizar las descripciones de cada paquete y cada clase en la documentación del API de Java.

Nota: todas las declaraciones **import** deben aparecer antes de la declaración de la clase. Colocar una declaración **import** dentro del cuerpo de la declaración de una clase, o después de la declaración de una clase, es un error de sintaxis.

Olvidar incluir una declaración **import** para una clase utilizada en su programa, generalmente produce un error de compilación con el mensaje “cannot resolve symbol” (no se puede resolver el símbolo). Cuando esto ocurra, compruebe que haya proporcionado las declaraciones import correctas, y que los nombres en las declaraciones import estén escritos apropiadamente.

La línea indica al compilador que nuestro programa utiliza la clase **JOptionPane** del paquete **javax.swing**. Este paquete contiene muchas clases que ayudan a los programadores en Java a crear Interfaces Gráficas de usuario (GUI, por sus siglas en inglés) para las aplicaciones.

La línea

- JOptionPane.showMessageDialog(null, "Bienvenido a NetBeans IDE 7");

Llaman al método **showMessageDialog** de la clase **JOptionPane** para mostrar un cuadro de diálogo que contiene un mensaje. Este método requiere dos argumentos. Cuando un método requiere varios argumentos, éstos se separan con comas (,). El primer argumento ayuda a la aplicación en Java a determinar en donde se va a colocar el cuadro de dialogo y el segundo argumento es la cadena a mostrar en el cuadro de dialogo. Cuando el primer argumento es **null**, el cuadro de dialogo aparece en el centro de la pantalla.

La Línea

- **System.exit(0);**

Utiliza el método estático exit de la clase System para terminar la aplicación. Esto se requiere para terminar cualquier aplicación que muestre una interfaz gráfica de usuario. Observe una vez más la sintaxis utilizada para llamar el método: el nombre de la clase (System), un punto (.) y el nombre del método (exit)

Ejercicio 3:

¿Suma de Enteros?

Paso 1.

Agregue una nueva clase al proyecto, con el nombre de **SumaEnteros**

Paso 2.

Agregue el siguiente código a la clase **SumaEnteros**

Explicación del Código

La línea

- **Import javax.swing.JOptionPane;// Paquete de Java**

Indica que el programa utiliza la clase **JOptionPane** del paquete **javax.swing**.

```

Import javax.swing.JOptionPane;// Paquete de Java
public class SumaEnteros {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        String primernumero;
        Stringsegundonumero;

        int numero1;
        int numero2;
        int suma;

        primernumero = JOptionPane.showInputDialog("Digite el primer numero");
        segundonumero = JOptionPane.showInputDialog("Digite el segundo numero");

        numero1 = Integer.parseInt(primernumero);
        numero2 = Integer.parseInt(segundonumero);

        suma = numero1 + numero2;

        JOptionPane.showMessageDialog(null, "La suma es:" +
        suma,"Resultado",JOptionPane.PLAIN_MESSAGE);

        System.exit(0);
    }
}

```

La línea

- public class SumaEnteros {

Comienza la declaración de la clase **SumaEnteros**. El nombre de archivo para esta clase public debe ser **SumaEnteros.java**.

Las líneas

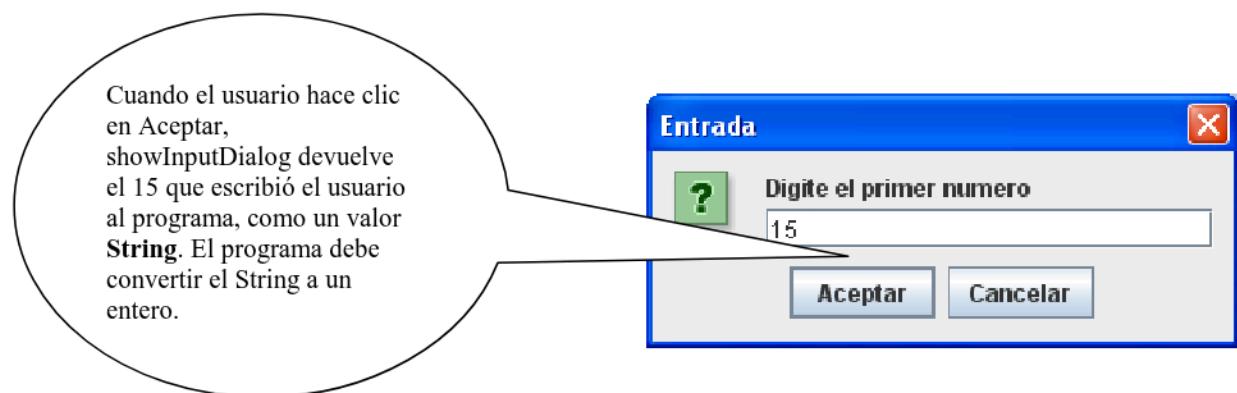
- String primernumero; //primera cadena introducida por usuario
- String segundonumero; //segunda cadena introducida por usuario

Son instrucciones para declarar variables, las cuales especifican los nombres y tipos de las variables. Estas variables declaradas son de tipo **String** (pueden almacenar secuencias de caracteres). La clase **String** se define en el paquete **java.lang**.

La línea

- primernumero = JOptionPane.showInputDialog("Digite el primer numero");

Utiliza el método **showInputDialog** de **JOptionPane** para mostrar el cuadro de dialogo el argumento para **.showInputDialog** indica lo que el usuario debe escribir en el campo de texto. Este mensaje se llama indicador, ya que hace que el usuario realice una acción específica.



Nota: en este programa, si el usuario escribe un valor no entero o hace clic en el botón **Cancelar** en el cuadro de dialogo, se producirá un error lógico en tiempo de ejecución y el programa no funcionará correctamente. Más adelante hablaremos sobre el **Manejo de excepciones**, para hacer que sus programas sean más robustos, al permitirles manejar dichos errores. Esto también se conoce como hacer que sus programas sea tolerantes.

Las líneas:

- `numero1 = Integer.parseInt(primernumero); //conversión del primer números`
- `numero2 = Integer.parseInt(segundonumero); //conversión del segundo números`

Convierte los dos valores **String** que introduce el usuario en valores **int**. El método estático **parseInt** de la clase **Integer** convierte su argumento **String** en número entero. La clase **Integer** se encuentra en el paquete **java.lang**.

La línea

- `JOptionPane.showMessageDialog(null, "La suma es:" +
suma,"Resultado",JOptionPane.PLAIN_MESSAGE);`

Utiliza el método estático **showMessageDialog** de la clase **JOptionPane** para mostrar el resultado de la suma. Esta nueva versión de método **showMessageDialog** de **JOptionPane** requiere 4 argumentos.

- El primer argumento **null** indica que el cuadro de dialogo de mensaje aparecerá en el centro de la pantalla.
- El segundo argumento es el mensaje a mostrar.
- El tercer argumento “**Resultados**” representa la cadena que debe aparecer en la **barra de título** del cuadro de dialogo.

- El cuarto argumento, **JOptionPane.PLAIN_MESSAGE**, es el tipo de cuadro de dialogo, un valor que indica el tipo de cuadro de dialogo de mensaje que se va a mostrar. Este tipo de cuadro de dialogo no muestra un icono a la izquierda del mensaje.

El argumento de la línea anterior ("La suma es: " + **suma**) utiliza el operador + para concatenar en el mensaje de salida la cadena **La suma es:** con el valor de otro tipo (la variable **suma**). La expresión se evalúa de la siguiente manera:

Java determina que los dos operadores del operador + (la cadena “la suma es” y el entero suma) son de distintos tipos, y que uno de ellos es **String**.

Java convierte automáticamente **suma** a **String**.

Java adjunta la representación **String** de suma al final de “**la suma es**”.

El método **showMessageDialog** muestra la cadena resultante en el cuadro de dialogo.

Ejercicio 4:

Paso 1

Agregue una nueva clase al proyecto, con el nombre de **Comparación**.

Paso 2

Agregue el siguiente código a la clase **Comparación**.

```
import javax.swing.JOptionPane;
public class Comparacion {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        String primernumero;
        Stringsegundonumero;
        String resultado;

        int numero1;
        int numero2;

        primernumero = JOptionPane.showInputDialog("Escriba el primer entero:");
        segundonumero = JOptionPane.showInputDialog("Escriba el Segundo Entero:");

        numero1= Integer.parseInt(primernumero);
        numero2 = Integer.parseInt(segundonumero);

        resultado= "";
    }
}
```

```

if (numero1==numero2) {
    resultado= resultado + numero1 + " == " + numero2;
}
if (numero1 != numero2) {
    resultado= resultado + numero1 + " != " + numero2;
}
if (numero1 < numero2) {
    resultado= resultado + "\n" + numero1 + " < " + numero2;
}
if (numero1 > numero2) {
    resultado= resultado + "\n" + numero1 + " > " + numero2;
}
if (numero1 <= numero2) {
    resultado= resultado + "\n" + numero1 + " <=" + numero2;
}
if (numero1 > numero2) {
    resultado= resultado + "\n" + numero1 + " >=" + numero2;
}

JOptionPane.showConfirmDialog(null, resultado,"Resultado de la
comparación",JOptionPane.INFORMATION_MESSAGE);
System.exit(0);
}
}

```

Explicación del código

La línea

- resultados = “”;

Asigna a resultados la cadena vacía. Todas las variables que se declaran en un método (como main) deben inicializarse (recibir un valor) antes de que puedan usarse en una expresión. Aún no sabemos cuál será el valor final de la cadena resultado, por lo que vamos a asignar a esta variable la cadena vacía como valor inicial temporal.

Ejercicio 5:

Paso 1

Agregar una nueva clase al proyecto llamada “**LecturaParametros**”

Paso 2

Insertar el siguiente código.

```
1  /*
2   * To change this license header, choose License Headers in
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package guial;
7
8  /**
9   *
10  * @author rafael
11  */
12 public class LecturaParametros {
13     public static void main(String[] args) {
14         System.out.print("Parametro 1: " + args[0]);
15         System.out.print("Parametro 2: " + args[1]);
16     }
17 }
```

Paso 3.

Ejecutar la clase y verificar si da el siguiente error, analizarlo e intentar darle una explicación a su docente y solicitarle como se deberá de hacer para ejecutar correctamente esta clase.

```
Output X HTTP Server Monitor Notifications
Debugger Console X Guia1 (run) X
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at guial.LecturaParametros.main(LecturaParametros.java:14)
C:\Users\User\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

V. EJERCICIOS COMPLEMENTARIOS.

- Modifique el ejercicio 3, para que no permita que el usuario ingrese números negativos. Si acaso los ingresará que se le notifique por medio de una ventana que no se pueden ingresar números negativos.
- Crear un programa en java que solicite la información básica de acuerdo a los siguiente:
 - > Nombres
 - > Apellidos
 - > Edad
 - > Carrera

La información solicitada previamente deberá de ser impresa en consola.

- Tomando en cuenta las instrucciones del ejercicio anterior, crear un programa que ahora utilice las cajas de texto de JoptionPane, tanto para captura de datos como para impresión de información. Para la impresión de información deberá realizarla en una sola caja de texto y dando un salto de línea entre cada dato solicitado.
- Validar el Ejercicio 5, para que no genere un error al ejecutarla al no enviarle parámetros, para este caso no deberá de ocupar excepciones.

VI. REFERENCIA BIBLIOGRAFICA.

- http://www.netbeans.org/index_es.html
Última fecha de visita: 13/01/2019
- https://netbeans.org/project_downloads/usersguide/shortcuts-80.pdf
Última fecha de visita: 13/01/2019
- Libros en la biblioteca de la UDB
 - **Aprendiendo Java 2 en 21 Días**
Lemay, Laura
 - **Cómo Programar en Java**
Deitel, Harvey M.