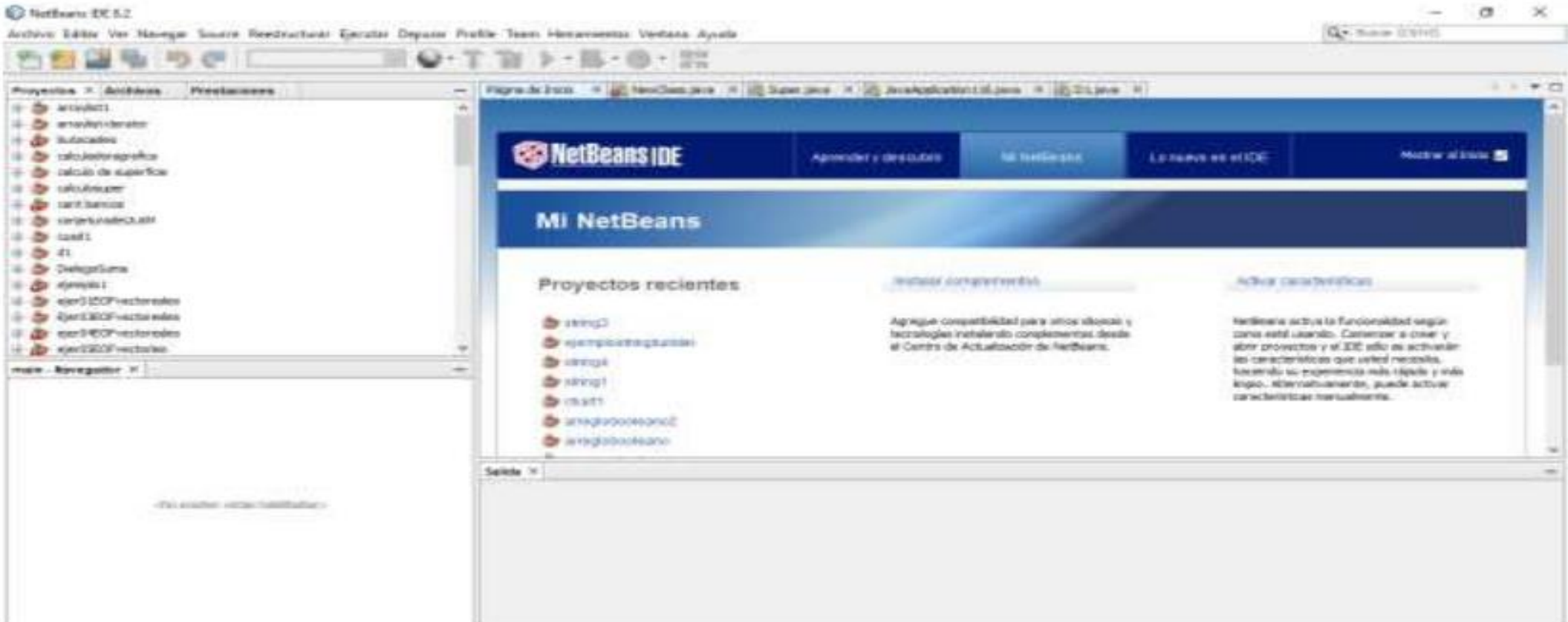
Clase 20

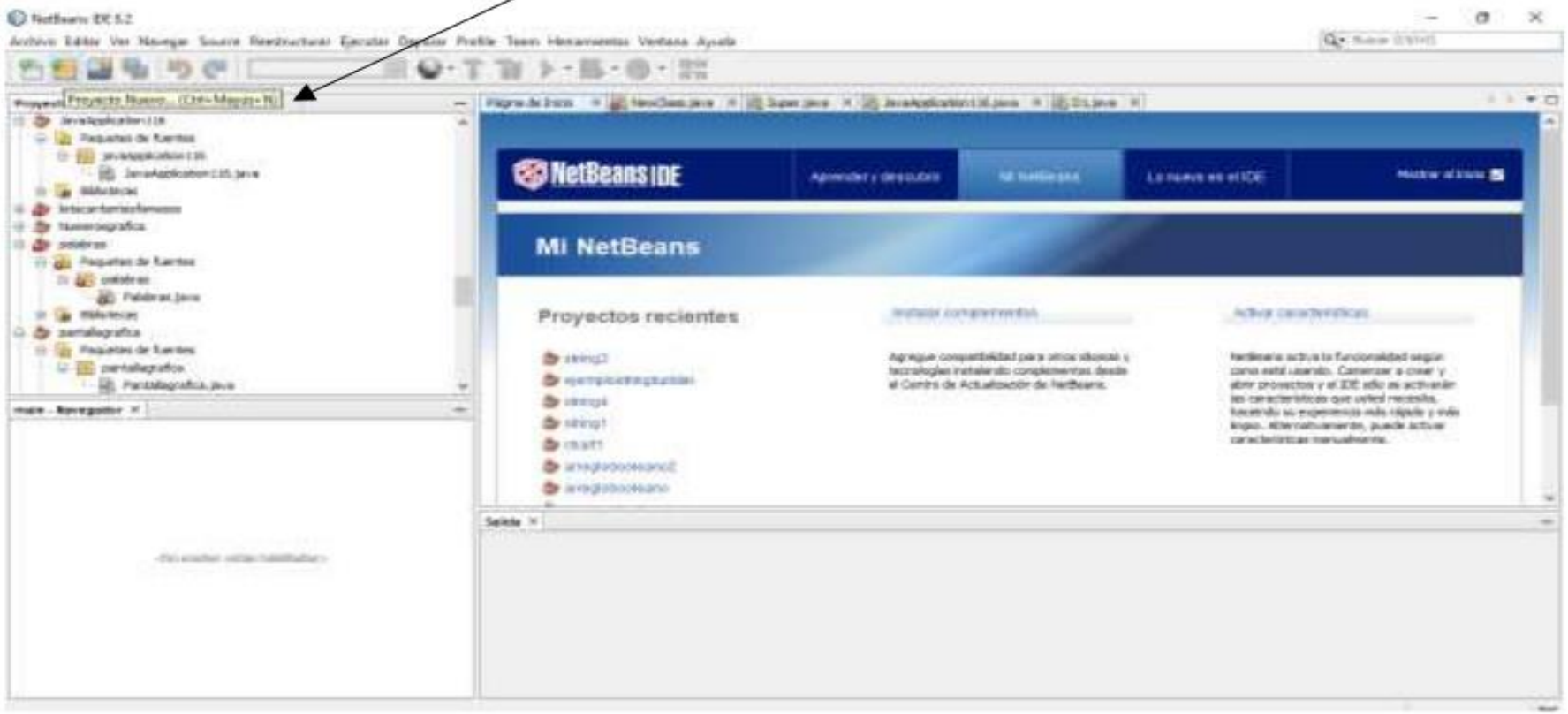
Java-POO Parte 1

Instalar NetBeans , Manejo de Clases , Herencia , Abstracción

Ingreso al Programa :

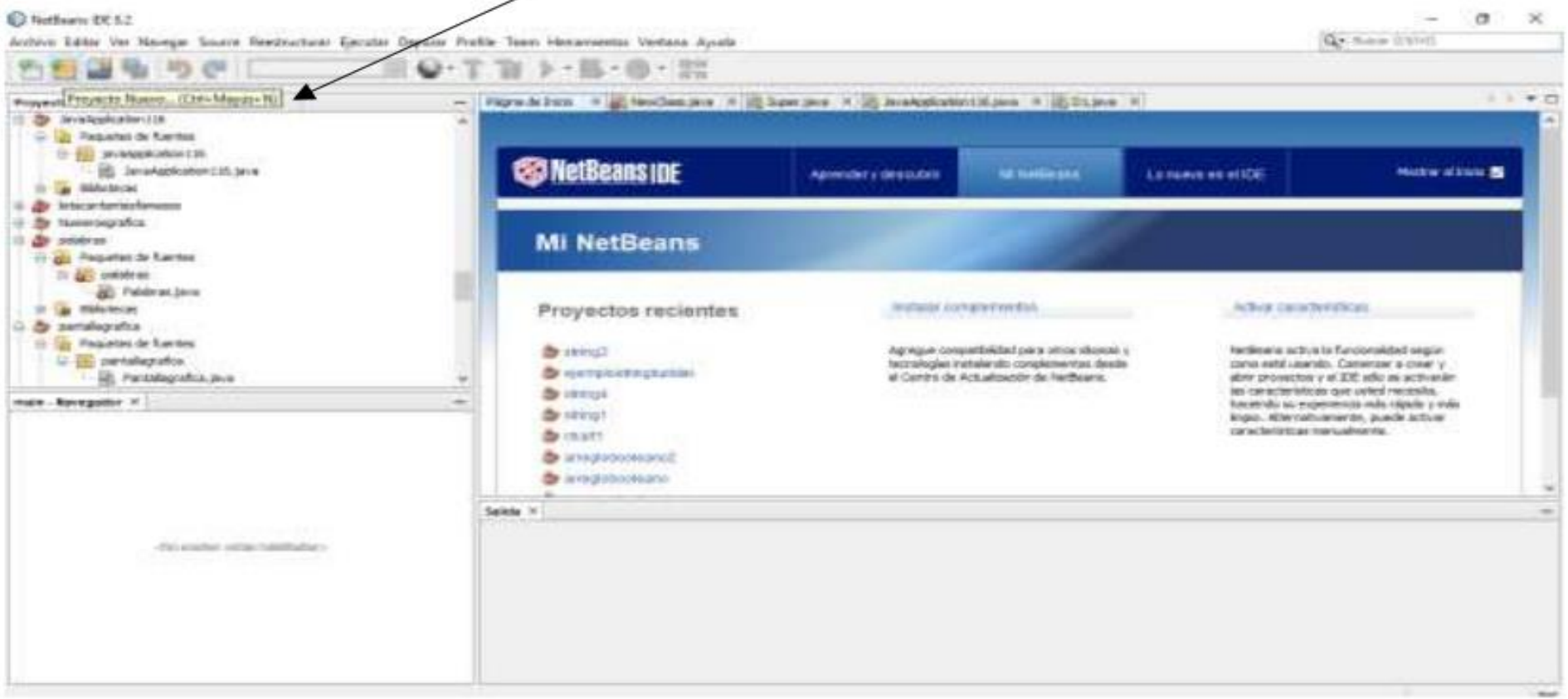


Seleccionamos Proyecto Nuevo

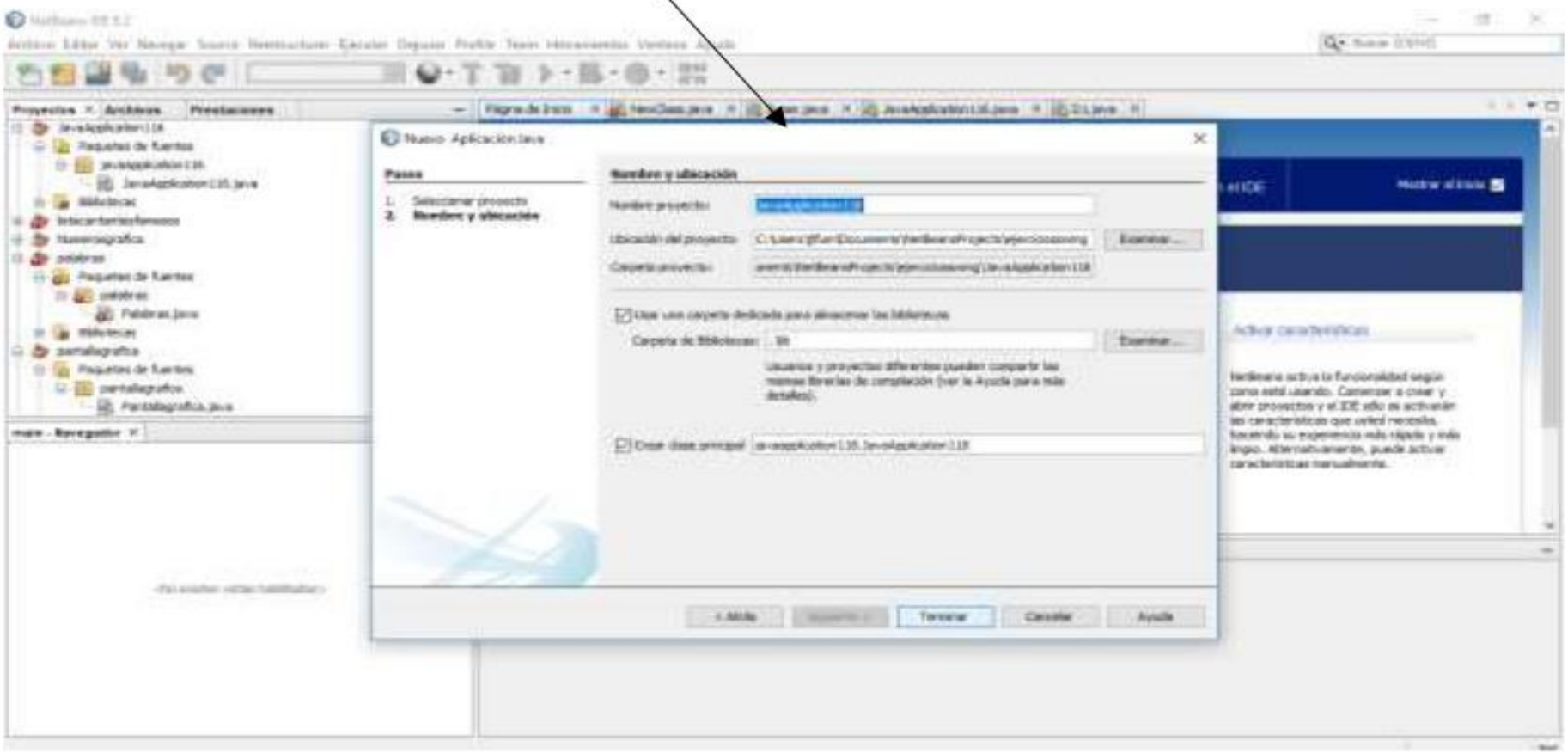


Vamos Buenos Aires

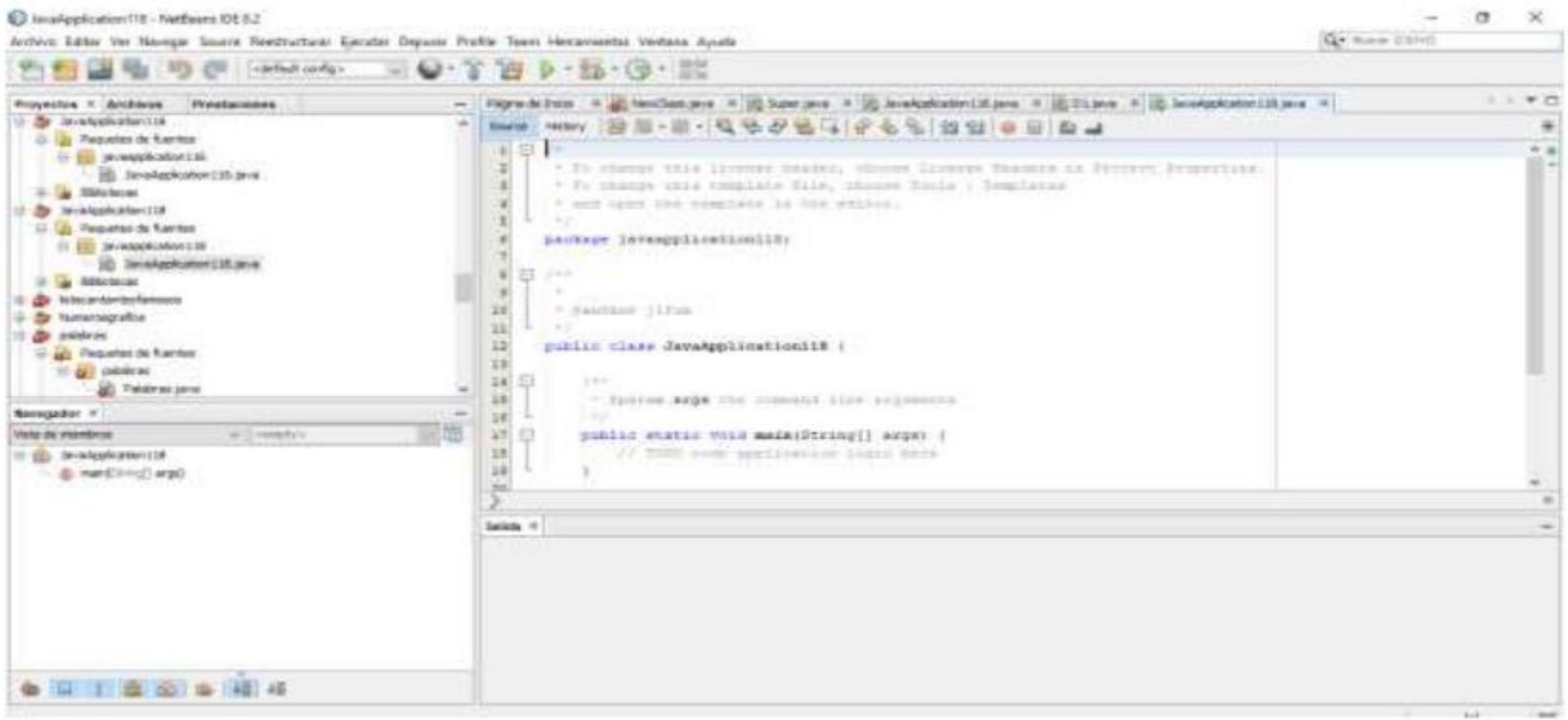
Seleccionamos Proyecto Nuevo



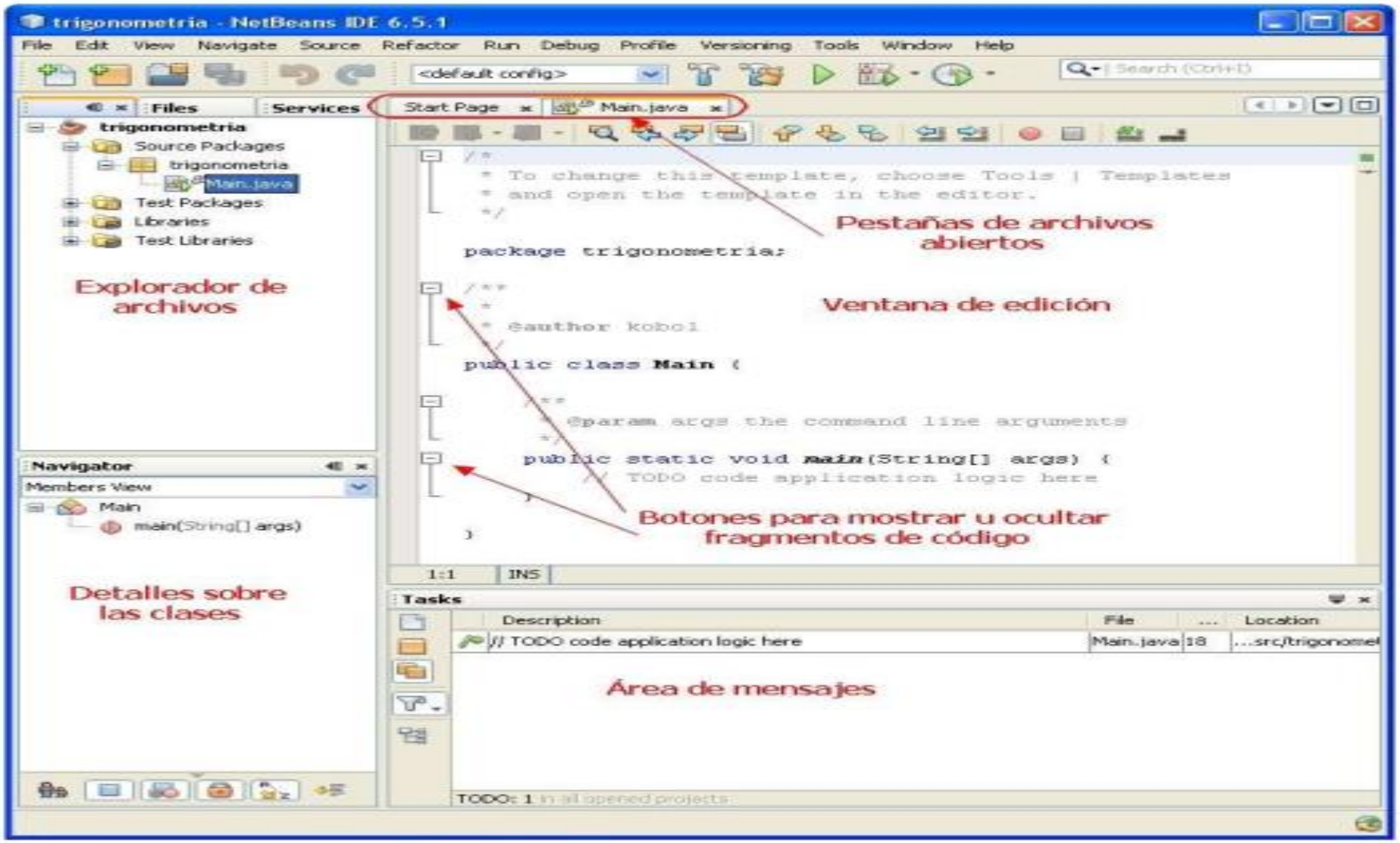
Colocamos el nombre a nuestro proyecto y luego Terminar. Nos indica la carpeta donde guarda los trabajos **NetbeansProjects**



Y ya nos mostrará la pantalla lista para empezar a realizar nuestro programa .



Partes de la pantalla de trabajo.



En la imagen superior vemos la pantalla dividida en 4 partes, *el Explorador de Archivos* como en Windows pero acá nos muestra todos los proyectos que tenemos creados y cada una de la partes que lo componen como por ejemplo las librerías o Package.

El sector de detalle de las clases , aca se ve la clase que estamos trabajando , en la imagen dice **MAIN** pues es el nombre que tiene puesto el proyecto.

Las pestañas de archivos abiertos habrá una por cada proyecto que abramos . cómo las solapas de las páginas web en el navegador.

Ventana de edición , aquí vamos a colocar nuestros códigos de JAVA .

Área de mensajes , se ven los mensajes de errores y demás y cuando ejecutemos el programa , se verán los resultados del algoritmo creado en ese sector también.

¿Qué es el PACKAGE?

Un **package** es una agrupación de clases afines. Recuerda el concepto de librería existente en otros lenguajes o sistemas. Una clase puede definirse como perteneciente a un **package** y puede usar otras clases definidas en ese o en otros **packages**. Los **packages** delimitan el espacio de nombres (space name).

En **Java** y en varios lenguajes de programación más, existe el concepto de **librerías**. Una **librería en Java** se puede entender como un conjunto de clases, que poseen una serie de métodos y atributos. Lo realmente interesante de estas **librerías** para **Java** es que facilitan muchas operaciones.

La clase **Scanner** de **Java** por ejemplo forma parte de la librería de **Java.Util** y provee métodos para leer valores de entrada de varios tipos Los valores de entrada pueden venir de varias fuentes, incluyendo valores que se entren por el teclado o datos almacenados en un archivo.

El **package** llevará el mismo nombre de la clase que creamos y que sale del nombre que ponemos a nuestro proyecto , por ejemplo COCHE , nuestro PACKAGE se verá asi en JAVA

Package coche; (en minúscula) recordemos que va **;** para distinguirla de la **Public Class Coche** (C mayúscula)

¿Qué es el Public Class?

Las clases en **Java** (**Java Class**) son **plantillas** para la creación de objetos, en lo que se conoce como programación orientada a objetos, la cual es una de los principales paradigmas de desarrollo de software en la actualidad. Por ejemplo Un **clase** llamada **Medios de Locomoción** , y dentro de esta clase la **subclase** **Medios de Locomoción Terrestres** , si nos imaginamos esto podríamos decir nombrar **objetos** como **Coches , camiones, micros, trenes , etc** estos tienen cosas en común por ejemplo , **ruedas , usar combustibles**

, frenos , luces , bocinas , etc ...todas esas cosas en común decimos que son **atributos** y a su vez esos **atributos** van a accionase en forma distinta en cada uno de estos objetos que mencionamos

Por ejemplo la forma de abrirse la puerta de un coche y la puerta de un tren, en cuanto a su movimiento de apertura.....

Por lo tanto aquí pondremos el nombre de ese **Objeto o Clase por ejemplo Public Class Coche (Pública pues la usan todos)** que en realidad es el nombre que le pusimos **a nuestro proyecto**

¿Qué es el PUBLIC STATIC VOID?

En **Java** un programa es un conjunto de definiciones de clases que están dispuestas en uno o más archivos. Dentro de la clase **Coche** por ejemplo se define el método main (Principal) :

public static **void** main (String args[]) { ... } ... Todos lo que escribamos entre estas llaves se va a ejecutar como programa.

La palabra **void** indica que el método main no retorna ningún valor.

Ejemplo de un programa codificado en JAVA, el proyecto se llama **Ejercicio23EOFleo y vemos que va en cada parte que hemos venido explicando.**

```
/*
 * 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 ejercicio23eofleo;  (el nombre del proyecto en minúscula)
import java.util.Scanner;  (Importamos de la librería Java.Util , el Scanner)
```

```
/** (La barra y los asteriscos delimitan espacios para colocar comentarios por ejemplo que hace el programa)
 *
 * @author jlfum
 */
public class Ejercicio23EOFleo {        (nombre de la clase igual a la del proyecto y del Package pero con la primera letra en mayúcula)


    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {  desde esta llave hasta la última llave de cierre va toda nuestra codificación)
                Scanner teclado=new Scanner(System.in);
                double a;
            a=0;
            System.out.println("---- CALCULADOR DE AÑO BICIESTO ----");
                System.out.println("Introduce un año");
            a=teclado.nextInt();

                if (a%4==0 && a%100!=0) {
                        System.out.println("El año "+a+" es bisiesto");
                } else {
                        if ((a%400==0 && a%100==0)) {
                                System.out.println("El año "+a+" ES bisiesto");
                        } else {
                                System.out.println("El año "+a+" NO es bisiesto");
```

```
                    }
                }
            }
        }
```

Por cada llave de apertura debe haber una de cierre , si alguna vez usamos las funciones condicionales de Excel sucede los mismo pero con los paréntesis.

Crear una clase

Definición de clases

La definición de una clase especifica cómo serán los objetos de dicha clase, esto es, de que variables y de que métodos constarán.

La siguiente es la definición más simple de una clase:

```
class nombreClase /* Declaración de la clase */
{

/* Aquí va la definición de variables y métodos */

}
```

Como se puede observar, la definición de una clase consta de dos partes fundamentales:

Mensajes y métodos

El modelado de objetos no sólo tiene en consideración los objetos de un sistema, sino también sus interrelaciones.

- **Mensaje**

  Los objetos interactúan enviándose mensajes unos a otros. Tras la recepción de un mensaje el objeto actuará. La acción puede ser el envío de otros mensajes, el cambio de su estado, o la ejecución de cualquier otra tarea que se requiera que haga el objeto.

- **Método**

  Un método se implementa en una clase, y determina cómo tiene que actuar el objeto cuando recibe un mensaje.

Cuando un objeto A necesita que el objeto B ejecute alguno de sus métodos, el objeto A le manda un mensaje al objeto B.

Al recibir el mensaje del objeto A, el objeto B ejecutará el método adecuado para el mensaje recibido.

· **Implementación de métodos**

Los métodos de una clase determinan los mensajes que un objeto puede recibir.

Las partes fundamentales de un método son el valor de retorno, el nombre, los argumentos (opcionales) y su cuerpo. Además, un método puede llevar otros modificadores opcionales que van al inicio de la declaración del método y que se analizarán más adelante. La sintaxis de un método es la siguiente:

```
<otrosModificadores> valorRetorno nombreMetodo( <lista de argumentos> )
{
/* Cuerpo del método */
sentencias;
}
```

Los signos <> indican que no son obligatorios.

Los métodos en Java pueden ser creados únicamente como parte de una clase. Cuando se llama a un método de un objeto se dice comúnmente que se envía un mensaje al objeto.

Ejemplo

```
/* Usuario.java */

class Usuario
{
 String nombre;
 int edad;
 String direccion;

void setNombre(String n)
{

nombre = n;
}

String getNombre()
{
return nombre;
}

void setEdad(int e)
{
edad = e;
}

int getEdad()
{
return edad;
}

void setDireccion(String d)
{
direccion = d;
}

String getDireccion()
{
return direccion;
}
}
```

Constructores y creación de objetos

Una vez que se tiene definida la clase a partir de la cual se crearán los objetos se está en la posibilidad de instanciar los objetos requeridos.

Para la clase Usuario del ejemplo anterior podemos crear un objeto de la siguiente manera:

```
Usuario usr1; //usr1 es una variable del tipo Usuario
usr1 = new Usuario();
```

La primera línea corresponde a la declaración del objeto, es decir, se declara una variable del tipo de objeto deseado.

La segunda línea corresponde a la iniciación del objeto.

- **El operador new**

  El operador **new** crea una instancia de una clase asignando la cantidad de memoria necesaria de acuerdo al tipo de objeto. El operador **new** se utiliza en conjunto con un *constructor*. El operador **new** regresa una referencia a un nuevo objeto.

- **Constructores**

  Un constructor es un tipo específico de método que siempre tiene el mismo nombre que la clase, y que se utiliza cuando se desean crear objetos de dicha clase, es decir, se utiliza al crear e iniciar un objeto de una clase.

- **Constructores múltiples**

  Cuando se declara una clase en Java, se pueden declarar uno o más constructores (*constructores múltiples*) opcionales que realizan la iniciación cuando se instancia un objeto de dicha clase.

Heredando clases en Java

El concepto de herencia que ya vimos en otro material anterior es recurrentes conduce a una estructura jerárquica de clases o estructura de árbol, lo cual significa que en la POO todas las relaciones entre clases deben ajustarse a dicha estructura.

En esta estructura jerárquica, cada clase tiene sólo una clase padre. La clase padre de cualquier clase es conocida como su *superclase*. La clase hija de una superclase es llamada una *subclase*.

De manera automática, una subclase hereda las variables y métodos de su superclase (más adelante se explica que pueden existir variables y métodos de la superclase que la subclase no puede heredar. Véase Modificadores de Acceso). Además, una subclase puede agregar nueva funcionalidad (variables y métodos) que la superclase no tenía.

*** Los constructores no son heredados por las subclases.**

Para crear una subclase, se incluye la palabra clave **extends** en la declaración de la clase.

```
class nombreSubclase extends nombreSuperclase{

}
```

En Java, la clase padre de todas las clases es la clase **Object** y cuando una clase no tiene una superclase explícita, su superclase es Object.

Sobreescritura de métodos (Override)

Una subclase hereda todos los métodos de su superclase que son accesibles a dicha subclase a menos que la subclase sobre escriba los métodos.

Una subclase sobre escribe un método de su superclase cuando define un método con las mismas características (nombre, número y tipo de argumentos) que el método de la superclase.

Las subclases emplean la sobre escritura de métodos la mayoría de las veces para agregar o modificar la funcionalidad del método heredado de la clase padre.

Ejemplo

```
class ClaseA
{
 void miMetodo(int var1, int var2)
 { ... }

 String miOtroMetodo( )
 { ... }
}
```

```
class ClaseB extends ClaseA
{
/* Estos métodos sobreescriben a los métodos
de la clase padre */

void miMetodo (int var1 ,int var2)
{ ... }

String miOtroMetodo( )
{ ... }
}
```

Clases abstractas

Una clase que declara la existencia de métodos pero no la implementación de dichos métodos (o sea, las llaves { } y las sentencias entre ellas), se considera una clase abstracta.

Una clase abstracta puede contener métodos no-abstractos pero al menos uno de los métodos debe ser declarado abstracta.

Para declarar una clase o un método como abstractos, se utiliza la palabra

reservada **abstract**.

```
abstract class Drawing
{
 abstract void miMetodo(int var1, int var2);
 String miOtroMetodo( ){ ... }
}
```

Una clase abstracta no se puede instanciar pero si se puede heredar y las clases hijas serán las encargadas de agregar la funcionalidad a los métodos abstractos. Si no lo hacen así, las clases hijas deben ser también abstractas.