GUÍA DE LABORATORIO 01_02
Configuración de Entorno de Desarrollo – POO – Clases/Objetos Java

Asignatura	Datos del alumno	Fecha y Firma
Algoritmos y solución de problemas	Apellidos:	
	Nombre:	

Instrucciones:

Desarrollar las actividades que indica el docente en base a la guía de trabajo que se presenta.

1. **Objetivos:**
   - Instalación y Configuración del JDK
   - instalación y configuración de visual Studio Code.

2. **Equipos, Herramientas o Materiales**
   - Computador
   - Software: JDK – VSCode - Extension Pack for Java.

3. **Fundamento Teórico**

   3.1. Programación Orientada a Objetos de Java

   OOP significa **Programación Orientada a Objetos**.

   La programación procedimental consiste en escribir procedimientos o métodos que realicen operaciones sobre los datos, mientras que la programación orientada a objetos consiste en crear objetos que contengan tanto datos como métodos.

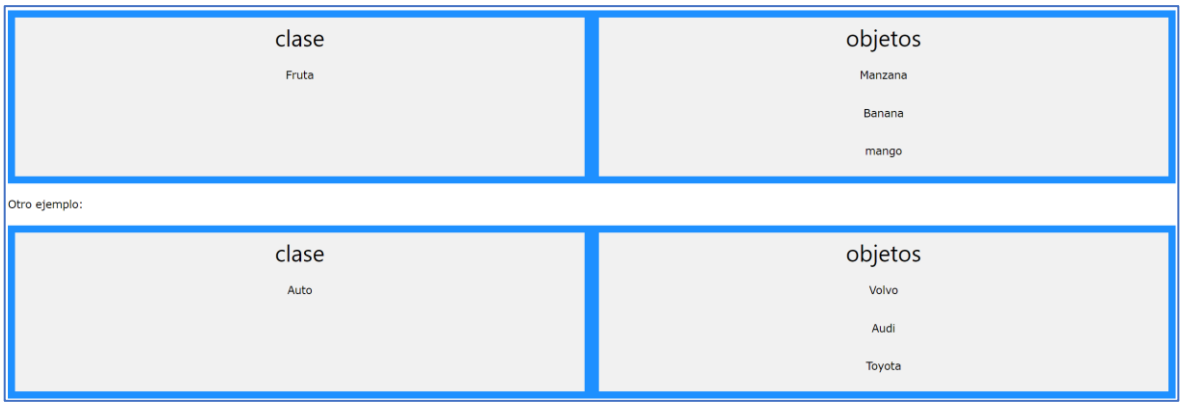
   La programación orientada a objetos tiene varias ventajas sobre la programación procedimental:
   - La programación orientada a objetos es más rápida y fácil de ejecutar
   - La programación orientada a objetos proporciona una estructura clara para los programas.
   - La programación orientada a objetos ayuda a mantener el código Java DRY ("No te repitas") y hace que el código sea más fácil de mantener, modificar y depurar.
   - La programación orientada a objetos permite crear aplicaciones totalmente reutilizables con menos código y un tiempo de desarrollo más corto.

   Consejo: El principio "No te repitas" (DRY) trata de reducir la repetición de código. Debes extraer los códigos que son comunes para la aplicación, colocarlos en un solo lugar y reutilizarlos en lugar de repetirlos.

   3.2. Java: ¿Qué son las clases y los objetos?

   Las clases y los objetos son los dos aspectos principales de la programación orientada a objetos.

   Observe la siguiente ilustración para ver la diferencia entre clase y objetos:



Entonces, una clase es una plantilla para objetos y un objeto es una instancia de una clase. Cuando se crean los objetos individuales, heredan todas las variables y métodos de la clase.

4. Instalación de Software Esencial:

4.1. Java Development Kit (JDK):

- Descarga la última versión del JDK desde la página oficial de Oracle (o adoptium.net para OpenJDK).
- Sigue las instrucciones del instalador para completar la instalación.
- Verifica la instalación abriendo una terminal y ejecutando java -version y javac -version. Deberías ver los números de versión.

4.2. Visual Studio Code:

- Descarga e instala VS Code desde la página oficial.

5. Configuración de VS Code:

5.1. Extensión Java de Microsoft:

- Abre VS Code.
- Ve a la vista de extensiones (icono de cuatro cuadrados en la barra lateral izquierda o Ctrl+Shift+X).
- Busca "Java Extension Pack" y haz clic en "Install". Esta extensión instalará automáticamente todas las extensiones necesarias para Java.

5.2. Configuración de Java Home:

- Presiona Ctrl+Shift+P (o Cmd+Shift+P en macOS) para abrir la paleta de comandos.
- Escribe "Java: Configure Java Runtime" y selecciona la opción.
- Si el JDK ya está detectado, asegúrate de que esté seleccionado. Si no, haz clic en "Add Java Runtime" y selecciona la carpeta de instalación del JDK.

5.3. Busca la ubicación del JDK:

- Abre el Explorador de archivos y busca la carpeta donde instalaste el JDK. Normalmente se encuentra en C:\Program Files\Java\jdk-xx (donde xx es la versión del JDK).
- Copia la ruta completa de la carpeta (por ejemplo, C:\Program Files\Java\jdk-17.0.1).

5.4. Accede a las variables de entorno:

- Haz clic derecho en "Este equipo" o "Mi PC" y selecciona "Propiedades".

- Haz clic en "Configuración avanzada del sistema".
- En la pestaña "Opciones avanzadas", haz clic en "Variables de entorno".

5.5. Crea o edita la variable JAVA_HOME:

- En la sección "Variables del sistema", busca la variable JAVA_HOME. Si no existe, haz clic en "Nueva".
- En el campo "Nombre de la variable", escribe JAVA_HOME.
- En el campo "Valor de la variable", pega la ruta que copiaste en el paso 1.
- Haz clic en "Aceptar" en todas las ventanas abiertas.

5.6. Edita la variable Path (opcional pero recomendado):

- En la sección "Variables del sistema", busca la variable Path y haz clic en "Editar".
- Haz clic en "Nuevo" y agrega la ruta %JAVA_HOME%\bin.
- Haz clic en "Aceptar" en todas las ventanas abiertas.

6. Clases/Objetos Java

Java es un lenguaje de programación orientado a objetos.

Todo en Java está asociado a clases y objetos, junto con sus atributos y métodos. Por ejemplo: en la vida real, un automóvil es un objeto. El automóvil tiene atributos, como peso y color, y métodos, como tracción y freno. Una clase es como un constructor de objetos, o un "plano" para crear objetos.

6.1. Crear una clase

Para crear una clase, utilice la palabra clave **class**:

Main.java

Crea una clase llamada " **Main**" con una variable x:

```
Main.java 1
Main.java > ...
1  public class Main {
2      int x = 5;
3  }
```

Recuerde la Sintaxis de Java que una clase siempre debe comenzar con una letra mayúscula y que el nombre del archivo Java debe coincidir con el nombre de la clase.

6.2. Crear un objeto

En Java, un objeto se crea a partir de una clase. Ya hemos creado la clase denominada **Main**, por lo que ahora podemos usarla para crear objetos.

Para crear un objeto de **Main**, especifique el nombre de la clase, seguido del nombre del objeto y utilice la palabra clave **new**:

Ejemplo

Crea un objeto llamado " **myObj**" e imprime el valor de **x**:

```java
public class Main {
    int x = 5;

    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        Main myObj = new Main();
        System.out.println(myObj.x);
      }
  }
```

6.3. Objetos múltiples

Puedes crear varios objetos de una clase:

Ejemplo

Crea dos objetos de **Main**:

```java
public class Main {
    int x = 5;

    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
      Main myObj1 = new Main();  // Object 1
      Main myObj2 = new Main();  // Object 2
      System.out.println(myObj1.x);
      System.out.println(myObj2.x);
    }
  }
```

6.4. Uso de múltiples clases

También puedes crear un objeto de una clase y acceder a él en otra clase. Esto se suele utilizar para una mejor organización de las clases (una clase tiene todos los atributos y métodos, mientras que la otra clase contiene el **main()**método (código que se ejecutará)).

Recuerde que el nombre del archivo java debe coincidir con el nombre de la clase. En este ejemplo, hemos creado dos archivos en el mismo directorio o carpeta:

- Main.java
- Second.java

```java
public class Main {
    int x = 5;

  }
```

Ejecute el archivo Second.java:

```java
class Second {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
      Main myObj = new Main();
      System.out.println(myObj.x);
    }
  }
```

7. Atributos de clase de Java

7.1. Atributos de clase de Java

Usamos el término "variable" x en el ejemplo (como se muestra a continuación). En realidad, es un atributo de la clase. O se podría decir que los atributos de clase son variables dentro de una clase:

Ejemplo

Crea una clase llamada " **Main**" con dos atributos: **x** y **y**:

```java
public class Main {
    int x = 5;
    int y = 3;
  }
```

7.2. Acceder a los atributos

Puede acceder a los atributos creando un objeto de la clase y utilizando la sintaxis de punto **(.)**:

El siguiente ejemplo creará un objeto de la **Main** clase, con el nombre **myObj**. Usamos el atributo **x** del objeto para imprimir su valor:

Ejemplo

```java
public class Main {
  int x = 5;

  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

7.3. Modificar atributos

También puedes modificar los valores de los atributos:

Ejemplo

Establezca el valor en **x** =40

```java
public class Main {
  int x;

  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

O anular los valores existentes:

Ejemplo

Cambie el valor de X a 25:

```java
public class Main {
  int x = 10;

  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

Si no desea tener la capacidad de anular los valores existentes, declare el atributo como **final**:

Ejemplo

```java
public class Main {
  final int x = 10;

  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // will generate an error: cannot assign a value to a final variable
    System.out.println(myObj.x);
  }
}
```

- La palabra **final** clave es útil cuando desea que una variable siempre almacene el mismo valor, como PI (3.14159...).
- La palabra clave **final** se denomina "modificador".

7.4. Objetos múltiples

Si crea varios objetos de una clase, puede cambiar los valores de los atributos en un objeto sin afectar los valores de los atributos en el otro:

Ejemplo

Cambie el valor de **x** a 25 en **myObj2** y déjelo **myObj1 x** sin cambios:

```java
public class Main {
  int x = 5;

  Run main | Debug mai
  public static vo                ] args) {
    Main myObj1 =                  Object 1
    Main myObj2 = new Main();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

7.5. Atributos múltiples

Puedes especificar tantos atributos como quieras:

Ejemplo

```java
public class Main {
  String fname = "Edem";
  String lname = "Terraza";
  int age = 32;

  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
}
```

8. Métodos de clase Java

los métodos se declaran dentro de una clase y que se utilizan para realizar ciertas acciones:

Ejemplo

Crea un método llamado **myMethod()**en **Main**:

```java
public class Main {
  static void myMethod() {
    System.out.println(x:"Hello World!");
  }
}
```

myMethod() imprime un texto (la acción), cuando se **llama** . Para llamar a un método, escriba el nombre del método seguido de dos paréntesis **()** y un punto y coma **;**

Ejemplo

```java
public class Main {
  static void myMethod() {
    System.out.println(x:"Hello World!");
  }

  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    myMethod();
  }
}
```

8.1. Estático vs. Público

A menudo verá programas Java que tienen atributos y métodos **static** o .**public**

En el ejemplo anterior, creamos un método **static**, lo que significa que se puede acceder a él sin crear un objeto de la clase, a diferencia de **public**, al que solo se puede acceder mediante objetos:

Ejemplo

Un ejemplo para demostrar las diferencias entre los **métodos static y :public**

```java
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println(x:"Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println(x:"Public methods must be called by creating objects");
  }

  // Main method
  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

8.2. Métodos de acceso con un objeto

Ejemplo

Cree un objeto **Car** llamado **myCar**. Llame a los métodos **fullThrottle**()y **speed**() del objeto **myCar** y ejecute el programa:

Main.java • Second.java

Main.java > ...

```java
// Create a Main class
public class Main {

  // Create a fullThrottle() method
  public void fullThrottle() {
    System.out.println(x:"The car is going as fast as it can!");
  }

  // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }

  // Inside main, call the methods on the myCar object
  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    Main myCar = new Main();   // Create a myCar object
    myCar.fullThrottle();      // Call the fullThrottle() method
    myCar.speed(maxSpeed:200);          // Call the speed() method
  }
}

// The car is going as fast as it can!
// Max speed is: 200
```

Nota.

El punto (.) se utiliza para acceder a los atributos y métodos del objeto.

Para llamar a un método en Java, escriba el nombre del método seguido de un conjunto de paréntesis () , seguido de un punto y coma (;).

Una clase debe tener un nombre de archivo coincidente (Mainy Main.java).

8.3. Uso de múltiples clases

Una buena práctica crear un objeto de una clase y acceder a él en otra clase.

Recuerde que el nombre del archivo java debe coincidir con el nombre de la clase. En este ejemplo, hemos creado dos archivos en el mismo directorio:

- Main.java
- Second.java

Main.java • Second.java

Main.java > ...

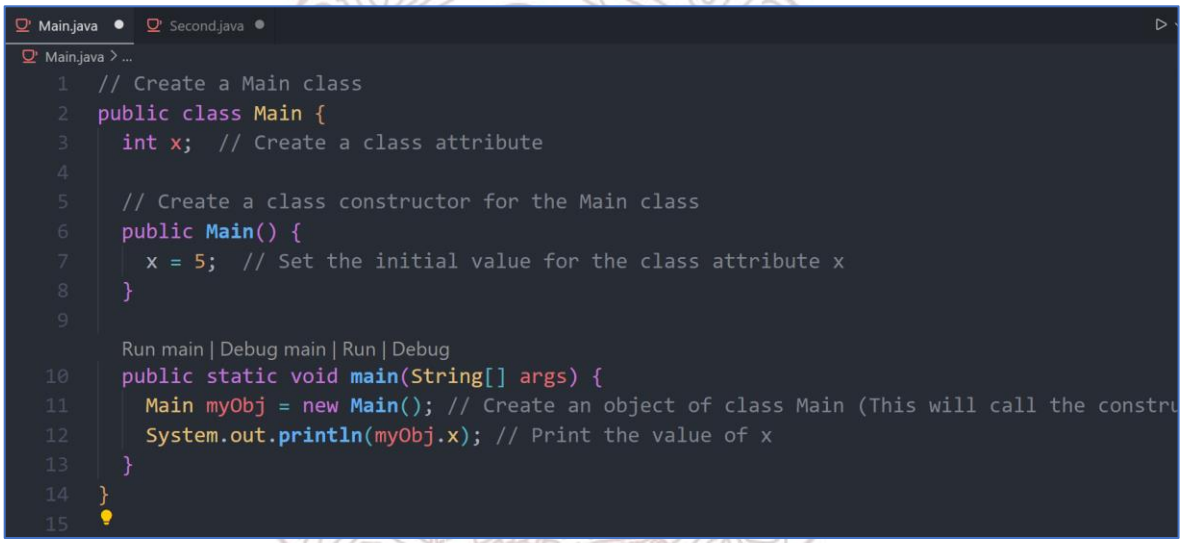
```java
public class Main {
  public void fullThrottle() {
    System.out.println(x:"The car is going as fast as it can!");
  }

  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
}
```

9. Constructores de Java

Un constructor en Java es un método especial que se utiliza para inicializar objetos. El constructor se llama cuando se crea un objeto de una clase. Se puede utilizar para establecer valores iniciales para los atributos de un objeto:

Ejemplo

Crear un constructor:



Nota:

Tenga en cuenta que el nombre del constructor debe coincidir con el nombre de la clase y no puede tener un tipo de retorno (como void).

Tenga en cuenta también que el constructor se llama cuando se crea el objeto.

Todas las clases tienen constructores de forma predeterminada: si no creas un constructor de clase tú mismo, Java crea uno por ti. Sin embargo, no podrás establecer valores iniciales para los atributos de los objetos.

9.1. Parámetros del constructor

Los constructores también pueden tomar parámetros, que se utilizan para inicializar atributos.

El siguiente ejemplo agrega un parámetro **int y** al constructor. Dentro del constructor, establecemos x en y (x=y). Cuando llamamos al constructor, pasamos un parámetro al constructor (5), que establecerá el valor de x en 5:

Ejemplo

```java
public class Main {
  int x;

  public Main(int y) {
    x = y;
  }

  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    Main myObj = new Main(y:5);
    System.out.println(myObj.x);
  }
}
```

Puedes tener tantos parámetros como quieras:

```java
public class Main {
  int modelYear;
  String modelName;

  public Main(int year, String name) {
    modelYear = year;
    modelName = name;
  }

  Run main | Debug main | Run | Debug
  public static void main(String[] args) {
    Main myCar = new Main(year:1969, name:"Mustang");
    System.out.println(myCar.modelYear + " " + myCar.modelName);
  }
}
```

10. Ejercicios

Ejercicio parte 01:

1) Crea una clase CuentaBancaria con los atributos titular, numeroCuenta y saldo. Implementa métodos para depositar, retirar y consultarSaldo. Asegúrate de manejar retiros que excedan el saldo disponible.
2) Agrega un constructor a la clase CuentaBancaria que permita inicializar los atributos titular, numeroCuenta y saldo al crear un objeto.
3) Crea una clase Calculadora con métodos para realizar las operaciones básicas (suma, resta, multiplicación, división). Asegúrate de manejar la división por cero.
4) Crea una clase Empleado con atributos nombre, salario y departamento. El atributo salario debe ser accesible solo dentro de la clase Empleado.