	UNIVERSIDAD DON BOSCO **FACULTAD DE INGENIERIA**
CICLO I	**GUIA DE LABORATORIO #3** **Programación Orientada a Objetos** **POO**

I. OBJETIVOS.

- Conocer la importancia de la implementación de métodos en Java.
- Conocer los conceptos fundamentales de la programación orientada a objetos.

II. INTRODUCCIÓN.

MODULOS DE PROGRAMAS EN JAVA

Hay dos tipos de módulos en Java: **métodos** y **clases**. Para escribir programas en Java, se combinan nuevos métodos y clases escritas por el programador, con los métodos y clases "**preempaquetados**", que están disponibles en la interfaz de programación de aplicaciones de Java (también conocida como la API de Java o biblioteca de clases de Java) y en diversas bibliotecas de clases. La API de Java proporciona una vasta colección de clases que contienen métodos para realizar cálculos matemáticos, manipulaciones de cadenas, manipulaciones de caracteres, operaciones de entrada/salida, comprobación de errores y muchas otras operaciones útiles.

Las clases de la API de Java proporcionan muchas de las herramientas que necesitan los programadores. Las clases de la API de Java forman parte del Kit de desarrollo de software para Java 2 (J2SDK), el cual contiene miles de clases preempaquetadas.

Los métodos (también conocidos como funciones o procedimientos en otros lenguajes de programación) permiten al programador dividir un programa en módulos, por medio de la separación de sus tareas en unidades autónomas; también conocidas como métodos declarados por el programador. Las instrucciones que implementan los métodos se escriben sólo una vez, y están ocultos de otros métodos.

Promoción de argumentos

Otra característica importante de las llamadas a métodos es la promoción de argumentos (o coerción de argumentos); es decir, forzar a que se pasen argumentos del tipo apropiado a un método. Por ejemplo, un programa puede llamar al método **sqrt** de **Math** con un argumento entero, inclusive cuando el método espera recibir un argumento doble.(Como Programar en Java, 2016)

Paquetes de la API de Java.

La palabra clave **package** permite agrupar clases e interfaces. Los nombres de los paquetes son palabras separadas por puntos y se almacenan en directorios que coinciden con esos nombres.

Por ejemplo, los ficheros siguientes, que contienen código fuente Java:

Applet.java, AppletContext.java, AppletStub.java, AudioClip.java

Contienen en su código la línea:

packagejava.applet;

Y las clases que se obtienen de la compilación de los ficheros anteriores, se encuentran con el nombre nombre_de_clase.class, en el directorio:

java/applet

Import

Los paquetes de clases se cargan con la palabra clave **import**, especificando el nombre del paquete como ruta y nombre de clase (análogo a #include de C/C++). Se pueden cargar varias clases utilizando un asterisco.

importjava.Date;

importjava.awt.;*

Si un fichero fuente Java no contiene ningún package, se coloca en el paquete por defecto sin nombre.

Paquetes Java

El lenguaje Java proporciona una serie de paquetes que incluyen ventanas, utilidades, un sistema de entrada/salida general, herramientas y comunicaciones. En la versión actual del JDK, los paquetes Java que se incluyen son:

- java.applet

Este paquete contiene clases diseñadas para usar con applets. Hay una clase Applet y tres interfaces: AppletContext, AppletStub y AudioClip.

- java.awt

El paquete AbstractWindowingToolkit (awt) contiene clases para generar widgets y componentes GUI (Interfaz Gráfico de Usuario). Incluye las clases Button, Checkbox, Choice, Component, Graphics, Menu, Panel, TextArea y TextField.

- java.io

El paquete de entrada/salida contiene las clases de acceso a ficheros: FileInputStream y FileOutputStream.

- java.lang

Este paquete incluye las clases del lenguaje Java propiamente dicho: Object, Thread,Exception, System, Integer, Float, Math, String, etc.

- java.net

Este paquete da soporte a las conexiones del protocolo TCP/IP y, además, incluye las

clases Socket, URL y URLConnection.

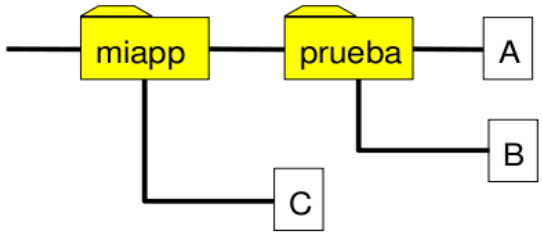
- java.util

Este paquete es una miscelánea de clases útiles para muchas cosas en programación. Se incluyen, entre otras, Date (fecha), Dictionary (diccionario), Random (números aleatorios) y Stack (pila FIFO).

Un paquete puede estar situado dentro de otro paquete formando estructuras jerárquicas. Ejemplo:

miapp.prueba.A

- Java obliga a que exista una correspondencia entre la estructura de paquetes de una clase y la estructura de directorios donde está situada.
- Las clases miapp.prueba.A, miapp.prueba.B y miapp.C deben estar en la siguiente estructura de directorios:



("Paquetes", 2016)

PROGRAMACION ORIENTADA A OBJETOS

Esta tecnología utiliza clases para encapsular (es decir, envolver) datos (atributos) y métodos (comportamientos). Por ejemplo, el estéreo de un auto encapsula todos los atributos y comportamientos que le permiten al conductor del auto seleccionar una estación de radio, o reproducir cintas o CD's. Las compañías que fabrican autos no fabrican los estéreos, sino que los compran y simplemente los conectan en el tablero de cada auto. Los componentes del radio están encapsulados en su caja.

Clase

- Conjunto de datos (atributos) y funciones (métodos) que definen la estructura de los objetos y los mecanismos para su manipulación.
- Atributos y métodos junto con interfaces y clases anidadas constituyen los miembros de una clase.

Declaración

[modificadores] classNombreDeClase{

//Declaración de atributos

//Declaración de métodos

//Declaración de clases anidadas e interfaces

}

- **Instancia de una clase.**

  Para su uso es necesaria la declaración, la instanciación y la inicialización del objeto.

  class Empleado{

  longidEmpleado = 0;

  String nombre = "SinNombre";

  double sueldo = 0;

  }

- **Declaración.**

  Empleado e;

- **Instanciación.**

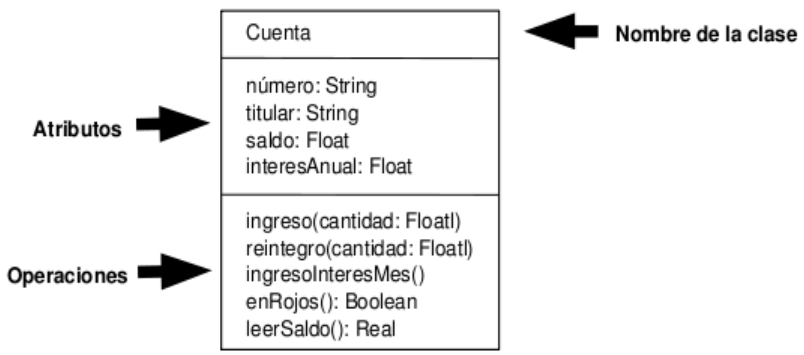
  e = new Empleado();

- **Se puede resumir en una única instrucción:**

  Empleado e = new Empleado();

Las clases de objetos representan conceptos o entidades significativos en un problema determinado.

Una clase describe las características comunes de un conjunto de objetos, mediante dos elementos:



Atributos (o variables miembro, variables de clase). Describen el estado interno de cada objeto.

Operaciones(o métodos, funciones miembro). Describen lo que se puede hacer con el objeto, los servicios que proporciona

El **encapsulamiento** permite a los objetos ocultar su implementación de otros objetos; a este principio se le conoce como oculta-miento de la información. Aunque los objetos pueden comunicarse entre sí, a través de interfaces bien definidas (de la misma forma que la interfaz de un conductor para un auto incluye un volante, pedal del acelerador, pedal del freno y palanca de velocidades), no están consientes de cómo se implementan otros objetos. Por lo general, los detalles de implementación se ocultan dentro de los mismos objetos. Evidentemente es posible conducir un auto de forma efectiva sin conocer los detalles sobre el funcionamiento de los motores, transmisiones y sistemas de escape.

En los lenguajes de programación por procedimientos (como C), la programación tiende a ser *orientada a la acción*. Sin embargo, la programación en Java está orientada a objetos. En los lenguajes de programación por procedimientos, la unidad de programación es la **función** (en Java las funciones se llaman **métodos**). En Java la unidad de programación es la *clase*. Los objetos eventualmente se *instancian* (es decir, se crean) a partir de estas clases, y los atributos y comportamientos están encapsulados dentro de los *límites* de las clases, en forma de campos y métodos.

Los programadores, al crear programas por procedimientos, se concentran en la escritura de funciones. Agrupan acciones que realizan cierta tarea en una función y luego agrupan varias funciones para formar un programa. Evidentemente los datos son importantes en los programas por procedimientos, pero existen principalmente para apoyar las acciones que realizan las funciones.

Los *verbos* en un documento de requerimientos del sistema que describen los requerimientos para una nueva aplicación, ayudan a un programador, que crea programas por procedimientos, a determinar el conjunto de funciones que trabajarán entre sí para implementar el sistema.

En contraste, los programadores en Java se concentran en la creación de sus propios tipos de referencia, mejor conocidos como *clases*. Cada clase contiene como sus *miembros* a un conjunto de campos (variables) y métodos que manipulan a esos campos. Los programadores utilizan esas clases para instanciar objetos que trabajen en conjunto para implementar el sistema.

Implementación de un tipo de dato abstracto con una clase

Las clases en Java facilitan la creación de ADTs (tipos de datos abstractos), los cuales ocultan su implementación a los clientes (o usuarios de la clase). Un problema en los lenguajes de programación por procedimientos es que el código cliente a menudo depende de los detalles de implementación de los datos utilizados en el código. Esta dependencia tal vez requiera de reescribir el código cliente, si la implementación de los datos cambia. Los ADTs eliminan este problema al proporcionar interfaces (conjunto de métodos ofrecidos por las clases) independientes de la implementación a sus clientes. Es posible para el creador de una clase cambiar la implementación interna de esa clase sin afectar a los clientes de esa clase).

III. PROCEDIMIENTO.

INFORMACIÓN:

Constructor:

Un Constructor es un método especial en Java empleado para inicializar valores en Instancias de Objetos, a través de este tipo de métodos es posible generar diversos tipos de instancias para la Clase en cuestión; la principal característica de este tipo de métodos es que llevan el mismo nombre de la clase.

Sobrecarga de métodos y de constructores:

La sobrecarga de métodos es la creación de varios métodos con el mismo nombre pero con diferentes firmas y definiciones. Java utiliza el número y tipo de argumentos para seleccionar cuál definición de método ejecutar.

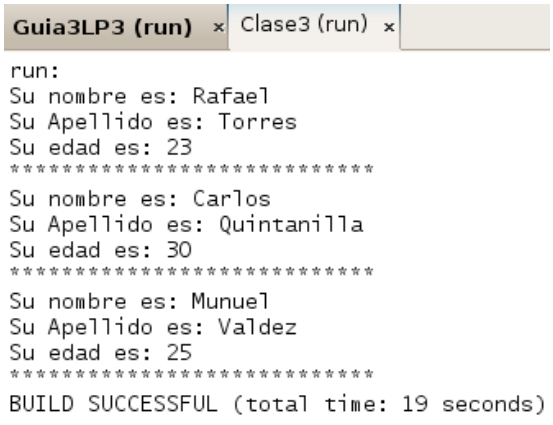
Java diferencia los métodos sobrecargados con base en el número y tipo de argumentos que tiene el método y no por el tipo que devuelve.

1. Crear una nueva clase llamada **"Persona"**.

```
class Persona {
   //Atributos
private String nombre;
private String apellido;
private String edad;

   //Constructor por sin parametros
   //Se utiliza al instaciar el oojeto
public Persona(){
      nombre="Rafael";
      apellido="Torres";
      edad="23";
   }
   //Constructor con parametros
public Persona(String nombre,String apellido,String edad){
         this.nombre=nombre;
         this.apellido=apellido;
         this.edad=edad;
   }

   //Permite definir datos a los atributos
public void ingresoDatos()
   {
      nombre=JOptionPane.showInputDialog("Ingrese el Nombre");
      apellido=JOptionPane.showInputDialog("Ingrese el Apellido");
```

```
        edad=JOptionPane.showInputDialog("Ingrese su edad");
    }
    //Permite imprimir los valores de los atributos
public void mostrarDatos(){
System.out.println("Su nombre es: "+nombre);
System.out.println("Su Apellido es: "+apellido);
System.out.println("Su edad es: "+edad);
System.out.println("*****************************");
    }

   public static void main(String args[]) {
Persona obj1=new Persona();//Instacia del objeto obj1
        //Instancia del objeto p2
        Persona obj2=new Persona("Manuel", "Valdez", "25");
        //LLamamos a el método mostrar datos de obj1
        obj1.mostrarDatos();
        //Cambiamor valor a los atributos de obj1
        obj1.ingresoDatos();
        //LLamamos a el método mostrardatos de obj1
        obj1.mostrarDatos();
        //LLamamos a el metodomostrardatos de obj2
        obj2.mostrarDatos();
        obj1.apellido="Sanchez";
        obj1.mostrarDatos();
   }
}
```

Ejecutar la clase anterior para ver el resultado en consola, para el método ingresar datos deberá ingresar los datos necesarios para almacenar los valores en los atributos correspondientes, dependiendo de lo que usted ingrese el resultado se vera de esta manera:



2. Constructores en java**,** crear una clase llamada **“Arboles”**.

```
public class Arboles  {

   //Constructor sin parametros
public Arboles() {
        System.out.println("Un árbol genérico");
}
   //Constructors con un parámetro string
public Arboles(String tipo) {
        System.out.println("Un árbol tipo " + tipo);
}
   //Constructor con un parámetro int
public Arboles(int altura) {
        System.out.println("Un árbol de " + altura + " metros");
}
  //Constructor con dos parametros uno int y el otro string
public Arboles(intaltura,Stringtipo) {
        System.out.println("Un " + tipo + " de " + altura + " metros");
}

   public static void main(String args[]) {
Arboles arbol1 = new Arboles(4);//Objeto1
Arboles arbol2 = new Arboles("Roble");//Objeto2
Arboles arbol3 = new Arboles();//Objeto3
Arboles arbol4 = new Arboles(5,"Pino");//Ojeto4
    }
}
```

Resultado en consola

```
run:
Un árbol de 4 metros
Un árbol tipo Roble
Un árbol genérico
Un Pino de 5 metros
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ejemplo3

Ahora hagamos un ejemplo más complicado se necesitara la declaración de la clase Tiempo1.

El siguiente ejemplo consiste de dos clases: **Tiempo1** y **PruebaTiempo1**. La clase **Tiempo1** (declarada en el archivo **Tiempo1.java**) se utiliza para crear objetos que representen la hora. La clase **PruebaTiempo1** (declarada en un archivo separado, llamado **PruebaTiempo1.java**) es una clase de aplicación, en la cual el método **main**crea un objeto de la clase Tiempo1 e invoca a sus métodos. Estas clases deben declararse en archivos separados, ya que ambas son clases **public**.

De hecho, cada declaración de clase que comience con la palabra clave public debe guardarse en un archivo que tenga exactamente el mismo nombre que la clase, y que termine con la extensión .java en su nombre de

archivo.

```
1  // DECLARACION DE LA CLASE Tiempo1.java
2  // Declaración de la clase Tiempo1 que mantiene la hora en formato de 24 horas.
3  import java.text.DecimalFormat;
4  public class Tiempo1 extends Object {
5  private int hora; // 0 - 23
6  private int minuto; // 0 - 59
7  private int segundo; // 0 - 59
8  // El constructor de Tiempo1 inicializa cada variable de instancia en cero;
9  // se asegura de que cada objeto Tiempo1 inicie en un estado consistente
10 public Tiempo1()
11 {
12 establecerHora( 0, 0, 0 );//se llama al método establecer hora
13 }
14 // establecer un nuevo valor de hora utilizando hora universal; realizar
15 // comprobaciones de validez en los datos; establecer valores inválidos en cero
16 public void establecerHora( int h, int m, int s )
17 {
18 hora = ( ( h >= 0 && h < 24 ) ? h : 0 );
19 minuto = ( ( m >= 0 && m < 60 ) ? m : 0 );
20 segundo = ( ( s >= 0 && s < 60 ) ? s : 0 );
21 }
22 // convertir a String en formato de hora universal
23 public String aStringUniversal()
24 {
25 DecimalFormat dosDigitos = new DecimalFormat( “00” );
26 return dosDigitos.format( hora ) + “:” +
27 dosDigitos.format( minuto ) + “:” + dosDigitos.format( segundo );
28 }
29 // convertir a String en formato de hora estándar
30 public String aStringEstandar()
31 {
32 DecimalFormat dosDigitos = new DecimalFormat( “00” );
33 return ( (hora == 12 || hora == 0) ? 12 : hora % 12 ) + “:” +
34 dosDigitos.format( minuto ) + “:” + dosDigitos.format( segundo ) +
35 ( hora <12 ? “ AM” : “ PM” );
36 }
```

```
37  } // fin de la clase Tiempo1
```

Explicación del código de la clase Tiempo1

En la línea 4:

publicclass Tiempo1 extendsObject {

Comienza la declaración de la clase **Tiempo1**. En esta declaración se indica que **Tiempo1** extiende a (mediante **extends**) la clase **Object** (del paquete **java.lang**). Los programadores en Java utilizan la herencia para crear clases a partir de clases existentes. De hecho, todas las clases en Java (excepto **Object) extienden** a (heredan de) una clase existente. La palabra clave **extends**seguida por el nombre de la clase **Object** indica que la clase **Tiempo1** hereda los atributos y comportamientos existentes de la clase **Object**.

El cuerpo de la declaración de la clase está delimitado por las llaves izquierda y derecha (**{ y }**).

Cualquier información que coloquemos en el cuerpo se dice que está encapsulada (es decir, envuelta) en la clase. Por ejemplo, la clase **Tiempo1** contiene tres valores enteros: **hora, minuto y segundo**, los cuales representan la hora en formato de *hora universal*. Anteriormente nos referimos a las variables declaradas en la declaración de una clase, pero no dentro de la declaración de un método, como campos. A un campo que no se declara como **static** se le conoce como variable de instancia; cada instancia (objeto) de la clase contiene su propia copia separada de las variables de instancia de la clase.

Las palabras clave **public** y **private** son *modificadoras de acceso*. Las variables o métodos declarados como **public** pueden utilizarse en cualquier parte del programa en que haya una referencia a un objeto de la clase. Las variables o métodos declarados con el modificador de acceso **private** pueden ser utilizadas solamente por los métodos de la clase en la que están declarados.

Las tres variables de instancia enteras **hora**, **minuto** y **segundo** se declaran con el codificador de acceso **private**, lo cual indica que estas variables de instancia son accesibles solamente para los métodos de la clase; esto se conoce como *ocultamiento de datos*. Cuando un programa crea (instancia) un objeto de la clase **Tiempo1**, dichas variables se encapsulan en el objeto y pueden utilizarse sólo mediante métodos de la clase de ese objeto (comúnmente a través de los métodos **public** de la clase). Por lo general, los campos se declaran como **private** y los métodos como **public**. Es posible tener métodos **private** y campos **public**. Los métodos **private** se conocen como *métodos utilitarios* o *métodos ayudantes*, ya que pueden ser llamados sólo por otros métodos de esa clase, y se utilizan para dar soporte a la operación de esos métodos. El uso de campos **public** es poco común, además de ser una práctica de programación peligrosa.

Los distintos modificadores de acceso quedan resumidos en la siguiente tabla:

Modificadores de acceso

	La misma clase	Otra clase del mismo paquete	Subclase de otro paquete	Otra clase de otro paquete
public	X	X	X	X
protected	X	X	X	
default	X	X		
private	X			

La clase **Tiempo1** contiene el constructor **Tiempo1** (líneas 10 a 13) y los métodos *establecerHora* (líneas 16 a 21), **aStringUniversal** (líneas 23 a 28) y **aStringEstandar** (líneas 30 a 36). Estos son los métodos **public** (también conocidos como servicios **public** o interfaces **public**) de la clase. Los clientes de la clase **Tiempo1** como la clase **PruebaTiempo1,** utilizan los métodos **public** de **Tiempo1** para manipular los datos almacenados en objetos **Tiempo1** o para hacer que la clase **Tiempo1** realice algún servicio. Los clientes de una clase utilizan referencias para interactuar con un objeto de la clase. Por ejemplo, el método **paint** de un **applet** es un cliente de la clase **Graphics**. El método **paint** utiliza su argumento (una referencia a un objeto **Graphics**, como g) para dibujar el **applet**, llamando a los métodos que son servicios **public** de la clase **Graphics** (como **drawString, drawLine, drawOval y drawRect**). Como se verá en este ejemplo, se utilizará una referencia a **Tiempo1** para interactuar con un objeto **Tiempo1**.

En las (líneas 10 a 13) se declara el constructor de la clase **Tiempo1**. **Un constructor inicializa los objetos de una clase.** Cuando un programa crea un objeto de la clase **Tiempo1**, la instrucción **new** asigna memoria para el objeto y llama al constructor para que inicialice a ese objeto.

El método **establecerHora** (líneas 16 a 21) es un método **public** que declara tres parámetros **int** y los utiliza para ajustar la hora. Una expresión condicional evalúa cada argumento para determinar si el valor se encuentra en el rango especificado por ejemplo para el valor de la hora aparece un código como el siguiente:

hora = ((h >= 0 && h < 24) ? h : 0);

Lo que realiza esta línea es que si el valor de parámetro llamado h pasado al método está entre 0y 23 entonces deja el valor establecido de h y queda asignado en la variable hora, si las condiciones no son cumplidas entonces la variable hora queda con el valor de cero. Cualquier valor fuera de estos rangos es un valor inválido y se establece en cero de manera predeterminada, para asegurar que un objeto **Tiempo1** siempre contenga datos válidos. (En este ejemplo, cero es un valor válido para **hora**, **minuto** y **segundo**) esto también se conoce como **mantener el objeto en un estado consistente o mantener la integridad del objeto.**

El método **aStringUniversal** (líneas 23 a 28) no toma argumentos y devuelve una cadena en formato de hora universal, el cual consiste de seis dígitos: dos para la hora, dos para el minuto y dos para el segundo. Por ejemplo, si la hora fuera 1:30:07 PM, el método **aStringUniversal** devolvería 13:30:07. En la línea 32 se crea una instancia de la clase **DecimalFormat** (importada en la línea 3 del paquete **java.text**) para dar formato a la hora universal. A la referencia **dosDigitos** se le asigna una referencia a un objeto de la clase **DecimalFormat**, la cual se inicializa con el patrón **“00”.** Esto indica que un número con formato debe consistir de dos dígitos; cada 0 representa la posición de un dígito. Si el número al que se está dando formato

es de un solo dígito, se le agrega automáticamente un 0 a la izquierda (es decir 8 se forma como 08).

Como utilizar la clase Tiempo1:

Después de declarar la clase, podemos utilizarla como un tipo en las declaraciones como:

Tiempo1 puestaDeSol; // referencia a un objeto Tiempo1

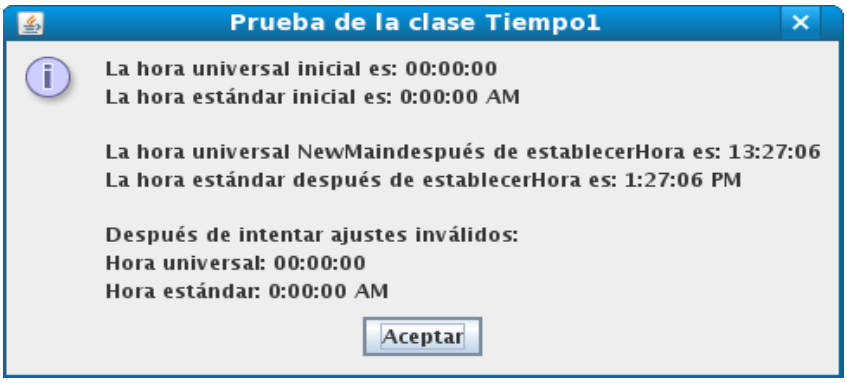
El nombre de la clase (**Tiempo1**) es un nombre de tipo. Una clase puede producir muchos objetos, de igual forma que un tipo primitivo, como **int**, puede producir muchas variables. Los programadores pueden declarar nuevos tipos de clases conforme lo necesiten; esta es una de las razones por las que Java se conoce como un *lenguaje extensible*.

Ahora se procederá a escribir el código de la clase de aplicación

PruebaTiempo1.

```
1.  // PruebaTiempo1.java
2.  // Clase PruebaTiempo1 que utiliza la clase Tiempo1.
3.  import javax.swing.JOptionPane;
4.  public class PruebaTiempo1 {
5.  public static void main( String args[] )
6.  {
7.  Tiempo1 tiempo = new Tiempo1(); // llamar al constructor de Tiempo1
8.  // anexar versión String de tiempo a salida String
9.  String salida = "La hora universal inicial es: " +
10. tiempo.aStringUniversal() + "\nLa hora estándar inicial es: " +
11. tiempo.aStringEstandar();
12. // cambiar tiempo y anexar hora actualizada a salida
13. tiempo.establecerHora( 13, 27, 6 );
14. salida += "\n\nLa hora universal después de establecerHora es: " +
15. tiempo.aStringUniversal() +
16. "\nLa hora estándar después de establecerHora es: " +
tiempo.aStringEstandar();
17. // establecer tiempo con valores inválidos; anexar hora actualizada a salida
18. tiempo.establecerHora( 99, 99, 99 );
19. salida += "\n\nDespués de intentar ajustes inválidos: " +
20. "\nHora universal: " + tiempo.aStringUniversal() +
21. "\nHora estándar: " + tiempo.aStringEstandar();
22. JOptionPane.showMessageDialog( null, salida,
23. "Prueba de la clase Tiempo1", JOptionPane.INFORMATION_MESSAGE );
24. System.exit( 0 );
```

```
25.     } // fin de main
26.     } // fin de la clase PruebaTiempo1
```



HERENCIA

La herencia es una forma de reutilización de software en la que las clases se crean absorbiendo los datos (atributos) y métodos (comportamientos) de una clase existente, y se mejoran con las nuevas capacidades, o con modificaciones en las capacidades ya existentes.

Al crear una clase, en vez de declarar miembros (variables y métodos) completamente nuevos, el programador puede designar que la nueva clase herede los miembros de una clase existente. Esta clase existente se conoce como superclase, y la nueva clase se conoce como subclase. Una vez creada, cada subclase puede convertirse en superclase de futuras subclases. Una subclase generalmente agrega sus propias variables y métodos. Por lo tanto, una subclase es más específica que su superclase y representa a un grupo más especializado de objetos. Generalmente, la subclase exhibe los comportamientos de su superclase junto con comportamientos adicionales específicos de esta subclase.

La superclase directa es la superclase a partir de la cual la subclase hereda en forma explícita. Una **superclase indirecta** se hereda de dos o más niveles arriba en la jerarquía de clases, la cual define las relaciones de herencia entre las clases. En java, la jerarquía de clases empieza con la clase Object (en el paquete **java.lang**), a partir de la cual heredan todas las clases en Java, ya sea en forma directa o indirecta. En el caso de la *herencia simple*, una clase se deriva de una superclase Java, a diferencia de C++, no soporta la *herencia múltiple* (que ocurre cuando una clase se deriva de más de una superclase directa).

Las relaciones de herencia forman estructuras jerárquicas en forma de árbol. Una superclase existe en una relación jerárquica con sus subclases. Aunque las clases pueden existir de manera independiente, cuando participan en relaciones de herencia se afilian con otras clases. Una clase se convierte ya sea en una superclase proporcionando datos y comportamientos a otras clases, o en una subclase, heredando sus datos y comportamientos de otras clases.

Ejemplo 4

Se crear una clase Empleado y una clase profesor la cual hereda de Empleado, el archivo fuente sera llamado Empleado.java, esto se debe a que la clase Empleado tiene el modificador de acceso public.

```
import javax.swing.JOptionPane;

public class Empleado {
private String nombre;
prívate String apellido;

//Metodo que permite mostrar el contenido de los atributos
public void mostrardatos()
{
        JOptionPane.showConfirmDialog(null,nombre +" "+apellido);
}
//Metodo que permite cambiar los datos de dlos atributos
public void ingresodatos()
{
      nombre=JOptionPane.showInputDialog("Ingrese el Nombre");
      apellido=JOptionPane.showInputDialog("Ingrese el Apellido");
}

}
   //Clase profesor que hereda de Empleado
class Profesor extends Empleado {
int sueldo;
publicvoid mostrar2()
{
        mostrardatos();//Metodo heredado de Empleado
JOptionPane.showMessageDialog(null,sueldo);
}

public void ingreso2()
{
        ingresodatos();//Metodo heredado de Empleado
        String s=JOptionPane.showInputDialog("Ingrese el sueldo");
  sueldo= Integer.parseInt(s);
}
}
```

Ahora se deberá crear otra clase en otro archivo fuente llamada pruebaempleado.java, la cual contiene el main y la instancia de los objetos creados para cada clase.

```
import javax.swing.JOptionPane;
public class PruebaEmpleado {
public static void  main(String[] args)
{
        Empleado emp=  new Empleado(); //Objeto de tipo Empleado
         Profesor pro=new Profesor();  //Objeto de tipo profesor
        JOptionPane.showMessageDialog(null,"claseempleado");
        //Utilizando los metodos de Empleado
        emp.ingresodatos();
        emp.mostrardatos();

        JOptionPane.showMessageDialog(null,"clase profesor");
        //Utilizando los metodos de profesor
```

```
        pro.ingreso2();
  pro.mostrar2();
    }
}
```

Ejemplo 5

Para ese ejemplo utilizaremos el modificador static para ver una implementación de él, para este ejemplo crearemos un archivo fuente llamado **Codigo.java**.

```
class Clase {
static intc ontador;
Clase() { //Constructor
contador++;
}

intgetContador() { //Obtenemos el valor de contador
        return contador;
}
}

public class Codigo {
  public static void main(String[] args) {
        Clase uno = new Clase();
         Clase dos = new Clase();
         Clase tres = new Clase();
         Clase cuatro = new Clase();
        System.out.println("Hemos declarado " + dos.getContador() + " objetos.");
  }
}
```

En este ejemplo el atributo **contador** de la clase llamada **Clase** tiene el modificador static el cual establecerá este atributo fijo para todas las objetos que se creen, eso quiere decir que al crear un objeto este llama al constructor el cual contiene a contador lo incremente y al crear otro objeto este modificara el contenido del contador establecido por el objeto anterior.

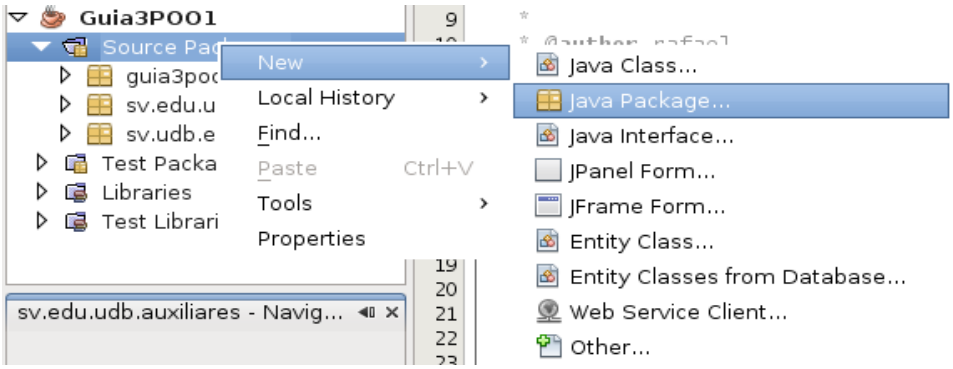
Uno de los posibles usos del modificador static es compartir el valor de una variable miembro entre objetos de una misma clase. Si declaramos una variable miembro de una clase, todos los objetos que declaremos basándonos en esa clase compartirán el valor de aquellas variables a las que se les haya aplicado el modificador static, y se podrá modificar el valor de este desde todas.

Ejemplo 6: Crear Paquetes en Java.

En este ejemplo crearemos dos paquetes con dos clases distintas para luego importarlas y ver el resultado.Para crear paquetes seguir los siguientes pasos

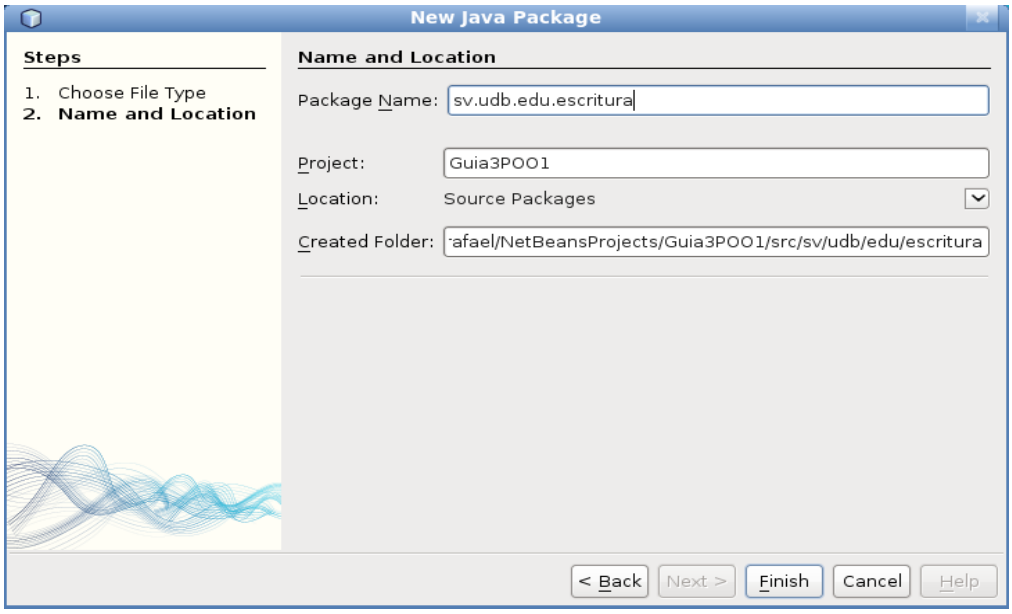
Paso 1

Dentro del proyecto hay una carpeta que se llama **"SourcePackages"**, dar click derecho sobre ella y elegir la pestaña de **"New"** luego seleccionar la opción **"Java Package"** como se muestra en la siguiente figura.



Paso 2

Aparecer una pantalla como la siguiente y en la cual ingresara el siguiente nombre del paquete **"sv.edu.udb.escritura"**



Paso 3

En el proyecto deberá crearse el paquete, ahora lo que haremos será crear una nueva clase dentro de ese paquete la cual se llamara **Pantalla**.

```
packagesv.edu.udb.escritura;  //Paquete en el que se encuentra la clase Pantalla

public class Pantalla {
      public  void sinSalto(String s) {
system.out.print(s);
}
      public void conSalto(String s) {
system.out.println(s);
}
```

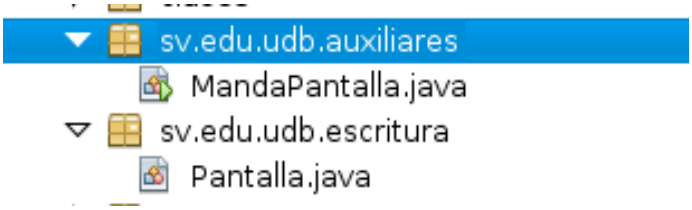
```
}
```

Paso 4

Crear otra paquete con el nombre **"sv.edu.udb.auxiliares"** y dentro del crear una clase llamada **MandaPantalla.**

```
package sv.edu.udb.auxiliares; //Paquete en el que se encuentra la clase MandaPantalla

import sv.edu.udb.escritura.*; //Importamos la clase Pantalla

public class MandaPantalla  {
public static void main(String args[]) {
        Pantalla primera = new Pantalla();
        primera.conSalto("Esto es un renglon CON salto de linea");
        primera.conSalto("Esta lineatambien tiene salto");
        primera.sinSalto("Linea Continua");
        primera.sinSalto("Linea Continua");
        primera.conSalto("Esta linea si tiene salto!");
        primera.sinSalto("Termina sin salto");
        System.out.println(" Se termina el uso de funciones");

}
}
```

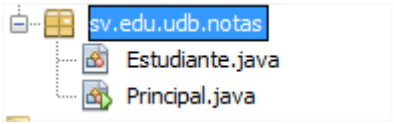
Los paquetes y las clases quedaran de la siguiente manera



Ejemplo 7: Uso de encapsulamiento:

Pare éste ejemplo deberá crear un paquete nuevo denominado

package sv.edu.udb.notas



Además de crear las dos Clases: **Estudiante.java** y **Principal.java**

Estudiante.java

```
packa gesv.edu.udb.notas;
public class Estudiante {
	String nombre;
	String apellido;
	prívate double nota1;
	prívate double nota2;
	prívate double nota3;

	//Constructor
	Estudiante(String nombre, String apellido){
		this.nombre = nombre;
		this.apellido = apellido;
	}

	//Definicion de Setters y Getters
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	publicStringgetApellido() {
		return apellido;
	}
	publicvoidsetApellido(String apellido) {
		this.apellido = apellido;
	}
	public double getNota1() {
		return nota1;
	}
	public void setNota1(double nota1) {
		//Nota validada
		if(nota1>=0 && nota1 <=10){
		this.nota1 = nota1;
		}else{
			this.nota1 = 0;
			System.out.println("Nota no valida!!! ("+ nota1 +")");
		}
	}
	public double getNota2() {
		return nota2;
	}
	public void setNota2(double nota2) {
		//Nota validada
		if(nota2>=0 && nota2 <=10){
			this.nota2 = nota2;
			}else{
				this.nota2 = 0;
				System.out.println("Nota no valida!!! ("+ nota2 +")");
			}
```

```
	}
	public double getNota3() {
		return nota3;
	}
	public void setNota3(double nota3) {
		//Nota validada
		if(nota3>=0 && nota3 <=10){
			this.nota3 = nota3;
			}else{
				this.nota3 = 0;
				System.out.println("Nota no valida!!! ("+ nota3 +")");
			}
	}
}
```

Principal.java

```
package sv.edu.udb.notas;

public class Principal {

	public static void main(String [] args ){
		Estudiante estudiante = new Estudiante("Walter","Samuel");

		estudiante.setNota1(8);

		estudiante.setNota2(15);

		estudiante.setNota3(10);

		System.out.println("Notas de " + estudiante.nombre + " "+ estudiante.getApellido() );

		System.out.println("Nota 1: " + estudiante.getNota1());
		System.out.println("Nota 2: " + estudiante.getNota2());
		System.out.println("Nota 3: " + estudiante.getNota3());
	}
}
```

IV. EJERCICIOS COMPLEMENTARIOS.

- Cree una clase llamada Estudiante, en ella debe asignar desde su constructor los datos personales del alumno. Crear un método en esta clase para ingresar 5 materias que cursa el estudiante y otro método para toda la información (datos personales y materias). Usar una clase separada donde debe instanciar y mostrar los datos (Objetivo: dominar clases e instanciar objetos).

- Se le ha encargado crear una calculadora con las operaciones aritméticas básicas (Clase: CalculadoraBasica) en la cual deben existir métodos que acepten 2 operandos para suma, resta, división y multiplicación. Ejemplo: cbasica.suma(numero1,numero2). A la vez le solicitan crear otra calculadora que contendrá otras operaciones avanzadas (Clase: CalculadoraAvanzada) en esta debe hacer

operaciones de potencia, opuesto y factorial. Ejemplo: cavanzada.potencia(numero1,numero2), cavanzada.factorial(numero);

Tome en cuenta que el opuesto de un número: opuesto de 1 es -1, opuesto de -2 es 2.

Tome en cuenta Factorial de un número: n!= 1*2*3*…*(n-1)*n. Ejemplo:
3! = 1*2*3 = 6
4!=1*2*3*4=24

Cada clase debe estar en packages (paquetes) separados(Objetivo: dominar packages)

- Edgardo tiene de mascota varios animales de diferente especie (gallo, perro, gato, hámster), los cuales quiere registrar en un sistema. Usted debe ayudarle creando una clase principal llamada Animal que debe contener el nombre, edad y alimento de cada mascota. Debe crear una clase para cada uno que heredará las propiedades de Animal y en ellas definir otras características únicas en cada uno (gallo color_plumas, perro color_pelaje, gato color_ojos, hámster cantidad_pulgas), además, debe establecer un método para mostrar datos en cada clase. Nota: utilice un solo archivo .java (Objetivo: dominar herencia y modificadores de acceso)

V. REFERENCIA BIBLIOGRAFICA.

Como Programar en Java. (2004). 5th ed.Deitel&Deitel.

Paquetes. (2016). Webtaller.com. Accedido 28 Octubre 2016, dehttp://www.webtaller.com/manual-java/paquetes.php

Hoja de cotejo.

HOJA DE EVALUACIÓN

Carnet:	Alumno:
Fecha:	Docente:
No.:	Título de la guía:

Actividad a evaluar	Criterio a evaluar	Cumplió		Puntaje
		SI	NO	
Desarrollo	Realizó los ejemplos de guía de práctica (40%)			
	Presentó todos los problemas resueltos (20%)			
	Funcionan todos correctamente y sin errores (30%)			
	Envió la carpeta comprimida y organizada adecuadamente en subcarpetas de acuerdo al tipo de recurso (10%)			
	PROMEDIO:			
Investigación complementaria	Entregó la investigación complementaria en la fecha indicada (20%)			
	Resolvió todos los ejercicios planteados en la investigación (40%)			
	Funcionaron correctamente y sin ningún mensaje de error a nivel de consola o ejecución (40%)			
	PROMEDIO:			