Ciclo Formativo de Grado Superior de Administración de Sistemas Informáticos en red

Módulo Profesional: IAW

U.T. 3.- Programación Orientada a Objetos. Programación JAVA

Departamento de Informática y Comunicación
IES San Juan Bosco (Lorca-Murcia)
Profesor: Juan Antonio López Quesada


I ES
SAN JUAN BOSCO

INDICE DE CONTENIDOS

- Introducción al Paradigma OO
- Introducción a JAVA
- Herramientas de Desarrollo
- Ejercicios Propuestos y Resueltos
- Ejercicios

Objetivos

Conocer los conceptos fundamentales de la teoría de **orientación** a **objetos.**

Diferenciar los distintos conceptos utilizados en la programación orientada a objetos (POO).

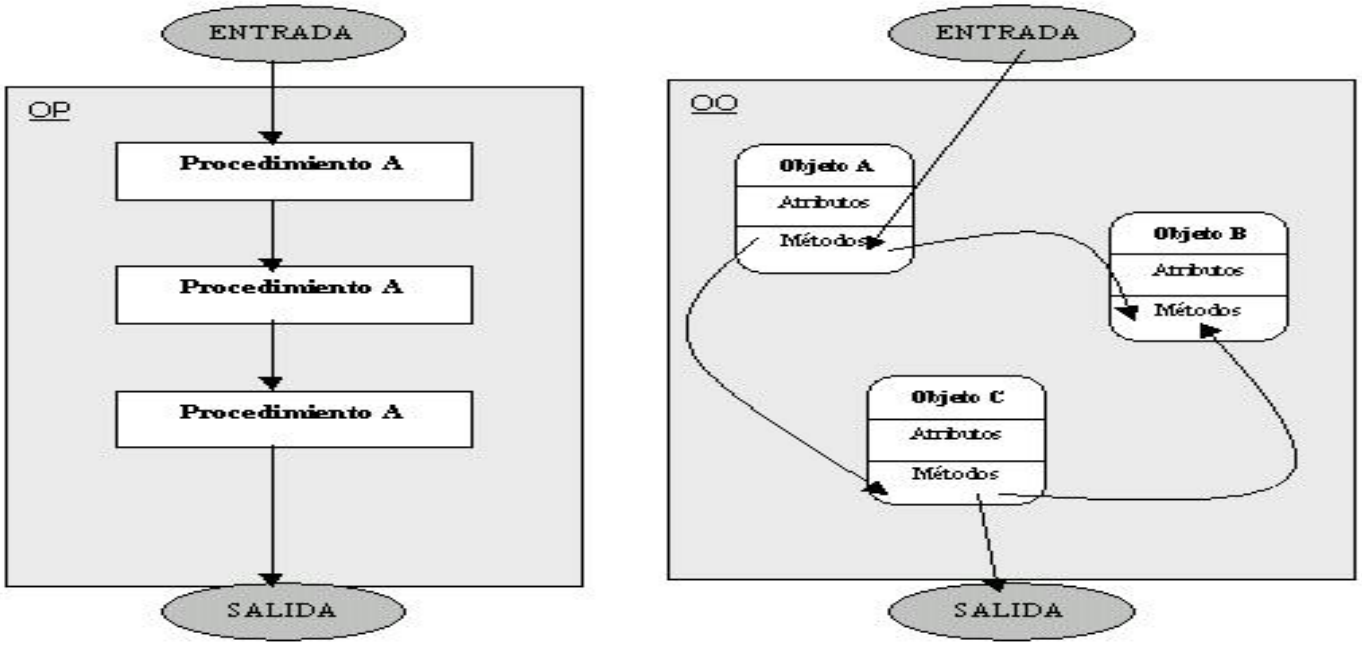
Entender el paradigma de **POO**. Su aplicación en distintos tipos de problemas.

Analizar como la orientación a objetos contribuye en los diferentes niveles de reutilización.

Aplicar los conceptos de orientación a objetos para el diseño y programación de sistemas.

Abstract

La técnica orientada a objetos sigue con frecuencia el mismo método que aplicamos en la resolución de problemas de la vida diaria.
El *análisis y diseño orientado a objetos* modela el mundo en términos de objetos que tienen **estado** y **comportamiento**, y **eventos** que activan operaciones que modifican el estado de esos objetos. Los objetos interactúan de manera formal con otros objetos mediante **mensajes**.



Introducción al Paradigma O.O

Una nueva forma de pensar.-

Es muy importante destacar que cuando hacemos referencia a la programación orientada a objetos no estamos hablando de unas cuantas características nuevas añadidas a un lenguaje de programación. *Estamos hablando de una nueva forma de pensar acerca del proceso de descomposición de problemas y de desarrollo de soluciones de programación*.

Introducción al Paradigma O.O

Una nueva forma de pensar.-

Surge en la historia como un intento para dominar la complejidad que, de forma innata, posee el software. Tradicionalmente, la forma de enfrentarse a esta complejidad ha sido empleando lo que llamamos *programación estructurada*, que consiste en descomponer el problema objeto de resolución en subproblemas y más subproblemas hasta llegar a acciones muy simples y fáciles de codificar. Se trata de descomponer el problema en acciones, en verbos.

En el ejemplo de un programa que resuelva ecuaciones de segundo grado, descomponíamos el problema en las siguientes acciones: primero, pedir el valor de los coeficientes a, b y c; después, calcular el valor del discriminante; y por último, en función del signo del discriminante, calcular ninguna, una o dos raíces.

Introducción al Paradigma O.O

Una nueva forma de pensar.-

La programación orientada a objetos es otra forma de descomponer problemas. Este nuevo método de descomposición es la descomposición en objetos; *vamos a fijarnos no en lo que hay que hacer en el problema, sino en cuál es el escenario real del mismo, y vamos a intentar simular ese escenario en nuestro programa.*

Los lenguajes de programación tradicionales no orientados a objetos, como C, Pascal, BASIC, o Modula-2, *basan su funcionamiento en el concepto de procedimiento o función*. Una función es simplemente un conjunto de instrucciones que operan sobre unos argumentos y producen un resultado. De este modo, un programa no es más que una sucesión de llamadas a funciones, ya sean al sistema operativo, proporcionadas por el propio lenguaje, o desarrolladas por el mismo usuario.

Introducción al Paradigma O.O

Una nueva forma de pensar.-

En el caso de los lenguajes orientados a objetos, como es el caso Java, el elemento básico no es la función, sino un ente denominado precisamente ***objeto***.

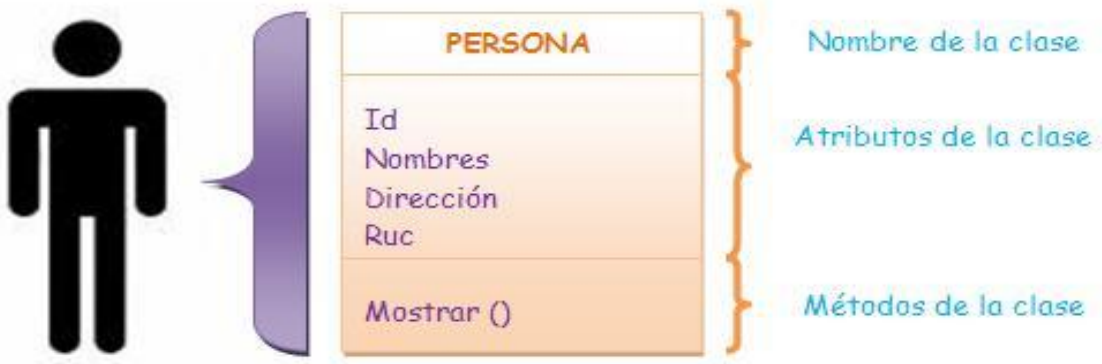
> Un objeto es la representación en un programa de un concepto, un hecho una entidad, cualquier elemento relevante en el dominio de mi problema y contiene toda la información necesaria para gestionarlo, administrarlo, manipularlo: *datos que describen sus atributos y operaciones que pueden realizarse sobre los mismos*.

La programación orientada a objetos es una nueva forma de pensar, una manera distinta de enfocar los problemas.

Introducción al Paradigma O.O

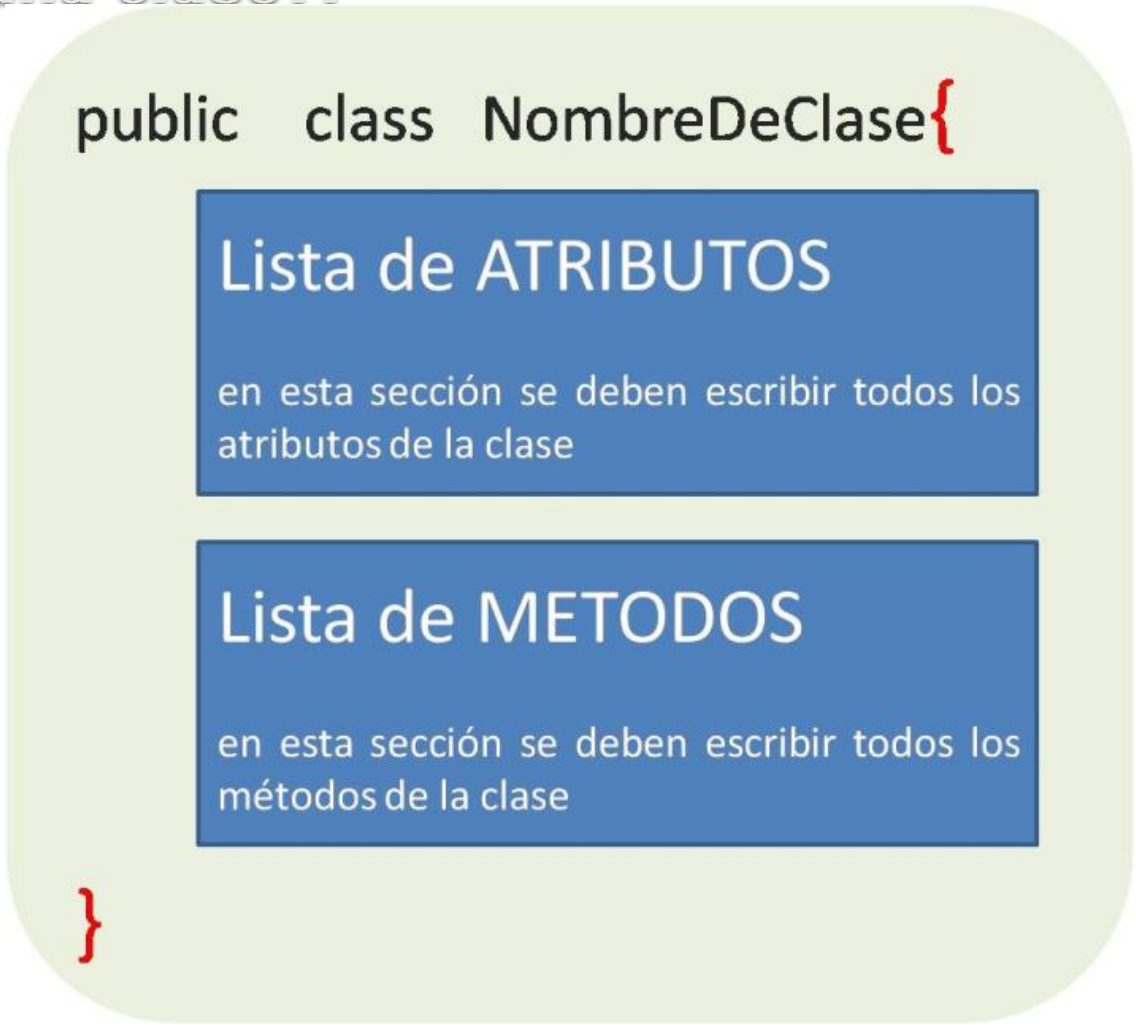
Una nueva forma de pensar.-

- Por otra parte si nos detenemos a pensar sobre cómo se nos plantea un problema cualquiera en la realidad, podremos ver que lo que hay en la realidad son **entidades (Clases)**.
- Estas entidades poseen un conjunto de propiedades o atributos (atributos de clase), y un conjunto de métodos mediante los cuales muestran su comportamiento.



Introducción al Paradigma O.O

¿Qué es una clase?.-



Introducción al Paradigma O.O

¿Qué es una clase?.-

- En el mundo real existen varios objetos de un mismo tipo, o como diremos enseguida, de una misma clase.
- Por ejemplo, mi bicicleta es una de las muchas bicicletas que existen en el mundo. Usando la terminología de la programación orientada a objetos, diremos que mi bicicleta es una instancia de la clase de objetos conocida como bicicletas.
- Todas las bicicletas tienen algunos estados o atributos (color, marcha actual, cadencia actual, dos ruedas) y algunos métodos (cambiar de marcha, frenar) en común.
- Sin embargo, el estado particular de cada bicicleta es independiente del estado de las demás bicicletas. La particularización de estos atributos puede ser diferente. Es decir, una bicicleta podrá ser azul, y otra roja, pero ambas tienen en común el hecho de tener una variable “color”.
- De este modo podemos definir una plantilla de variables y métodos para todas las bicicletas. ***Las plantillas para crear objetos son denominadas clases.***

Introducción al Paradigma O.O

¿Qué es una clase?.-

- **Una clase es una plantilla que define las variables y los métodos que son comunes para todos los objetos de un cierto tipo.**

En nuestro ejemplo, la clase bicicleta definiría variables miembro comunes a todas las bicicletas, como la marcha actual, la cadencia actual, etc. Esta clase también debe declarar e implementar los métodos o funciones miembro que permiten el ciclista para cambiar de marcha, frenar, y cambiar la cadencia de pedaleo:

```
public class bicicleta
 {
    // Atributos
          public double velocidad;
          .....
          public int marcha;
     // metodos
          cambio_marchas(){}
          ……
}
```

Introducción al Paradigma O.O

¿Qué es una clase?.-

- Después de haber creado la clase bicicleta, podemos crear cualquier número de objetos bicicleta a partir de la clase. Cuando creamos una instancia de una clase, el sistema reserva suficiente memoria para el objeto con todas sus variables miembro. Cada instancia tiene su propia copia de las variables miembro definidas en la clase.

Instanciación de la clase Bicicleta

```
Bicicleta var1=new Bicicleta(1,10,”azul”);
Bicicleta var2=new Bicicleta(2,16,”amarillo”);
Bicicleta var3=new Bicicleta(1,45,”roja”);
```

var1
var2
var3

Variables que hacen referencia a una instanciación e la clase bicicleta

Introducción al Paradigma O.O

¿Qué es un objeto?.-

- En filosofía un objeto es aquello que puede ser observado, estudiado y aprendido, en contraposición a la representación abstracta de ese objeto que se crea en la mente a través del proceso de generalización.

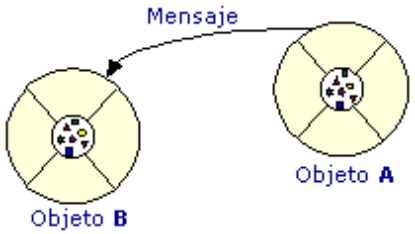
  - *En programación orientada a objetos (POO), una instancia de programa (por ejemplo un programa ejecutándose) es un conjunto dinámico de objetos interactuando entre sí.*
  - ***Los objetos en la POO son la instanciación, particularización construcción de las clases** accesibles por el programa (proporcionada por el compilador, diseñadas para el programa o codificadas por otro programador) , que estarán compuestos por **atributos** que representan los datos asociados al objeto, o lo que es lo mismo sus propiedades o características, y **métodos** que acceden a los atributos de una manera predefinida e implementan el comportamiento del objeto.*

Introducción al Paradigma O.O

¿Qué es un mensaje?.-

- Normalmente un único objeto por sí solo no es muy útil. En general, un objeto aparece como un componente más de un programa o una aplicación que contiene otros muchos objetos.
  - *Es precisamente haciendo uso de esta interacción como los programadores consiguen una funcionalidad de mayor orden y modelar comportamientos mucho más complejos. Una bicicleta (a partir de ahora particularizaremos) colgada de un gancho en el garaje no es más que una estructura de aleación de titanio y un poco de goma. Por sí sola, tu bicicleta (por poner una bicicleta en concreto) es incapaz de desarrollar ninguna actividad. Tu bicicleta es realmente útil en tanto que otro objeto interactúa con ella.*

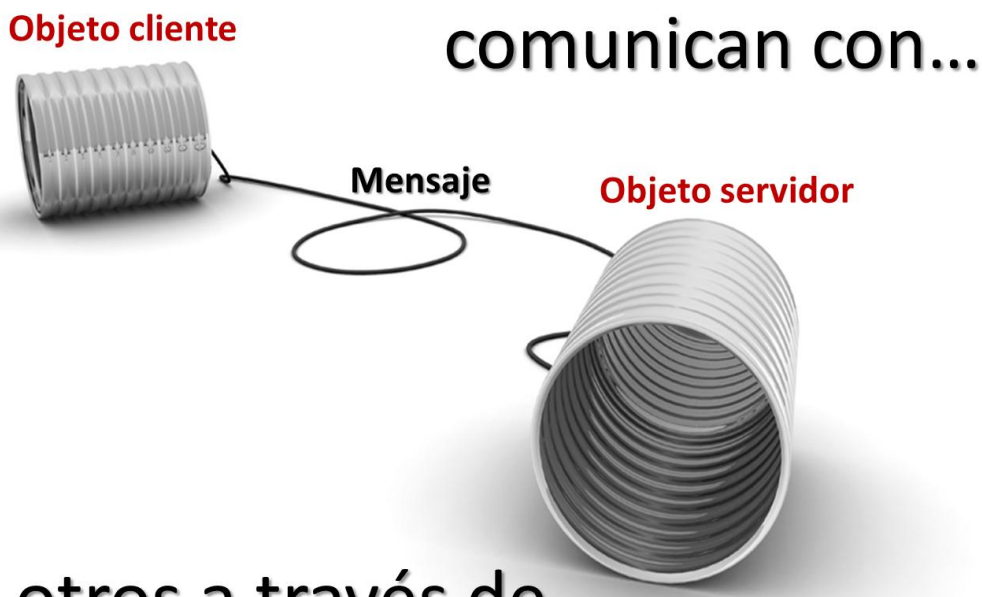
Bicicleta var=new Bicicleta(1,20,"amarillo");
….
var.parar();



Introducción al Paradigma O.O

¿Qué es un mensaje?.-

Los objetos se comunican con…



…otros a través de mensajes

Los objetos de un programa interactúan y se comunican entre ellos por medio de mensajes. Cuando un objeto A quiere que otro objeto B ejecute una de sus funciones miembro, métodos de B), el objeto A manda un mensaje al objeto B.

Introducción al Paradigma O.O

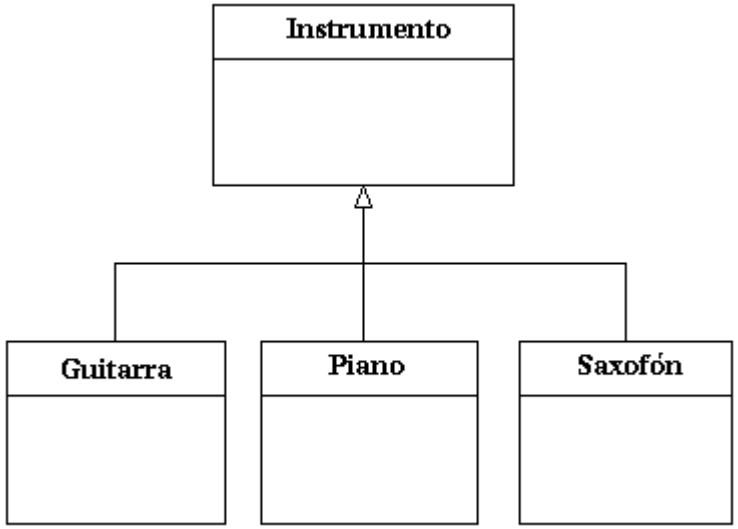
Cómo resumen de POO.-

- *Organización de los programas de manera que representan la interacción de las cosas en el mundo real.*
- *Un programa consta de un conjunto de Clases.*
- *Un programa en ejecución está formado por un conjunto de objetos interactuando entre ellos*
- *Los objetos son instanciaciones de clases.*
- *Cada objeto es responsable de unas tareas.*
- *Los objetos interactúan entre sí por medio de mensajes.*
- *Cada objeto es un ejemplar de una clase.*
- *Las clases se pueden organizar en una jerarquía de* ***herencia.***

La programación OO es una simulación de un modelo del universo.

Introducción al Paradigma O.O

Cómo resumen de POO.-



Introducción al Paradigma O.O

Beneficios la orientación a objetos.-

- ***Reutilización**. Permite la reusabilidad de código, el ahorro de dinero y el empleo de menos tiempo en de desarrollo.*
- **Integridad**. Los mecanismos de encapsulación protegen sus propios componentes contra los procesos que no tengan derecho a acceder a ellos.
- La **forma de pensar** en **objetos** es más natural. El diseñador piensa en términos de objetos y no en detalles de bajo nivel.
- **Programación más sencilla**. Los programas se crean a partir de piezas pequeñas.
- Es más **sencillo modificar código existente**, cada clase efectúa sus funciones independientemente de las demás.
- ..etc

Introducción a la Programación Java

Introducción.-

Java se creó como parte de un proyecto de investigación para el desarrollo de software avanzado. La meta era diseñar una plataforma operativa sencilla, fiable, portable, distribuida y de tiempo real. Cuando se inició el proyecto, C++ era el lenguaje del momento. Pero a lo largo del tiempo, las dificultades encontradas con C++ crecieron hasta el punto en que se pensó que los problemas podrían resolverse mejor creando una plataforma de lenguaje completamente nueva. Se extrajeron decisiones de diseño y arquitectura de una amplia variedad de lenguajes como Eiffel, SmallTalk, Objetive C y Cedar/Mesa.

El resultado es un lenguaje que se ha mostrado ideal para desarrollar aplicaciones de usuario final seguras, distribuidas y basadas en red en un amplio rango de entornos desde los dispositivos de red embebidos hasta los sistemas de sobremesa e Internet.

Introducción a la Programación Java

Características.-

- Lenguaje de propósito general.
- Lenguaje Orientado a Objetos.
- Sintaxis inspirada en la de C/C++.
- Lenguaje multiplataforma: Los programas Java se ejecutan sin variación (sin recompilar) en cualquier plataforma soportada (Windows, UNIX, Mac...)
- Lenguaje interpretado: El intérprete a código máquina (dependiente de la plataforma) se llama Java Virtual Machine (JVM). El compilador produce un código intermedio independiente del sistema denominado *bytecode*.
- Lenguaje gratuito: Creado por SUN Microsystems, que distribuye gratuitamente el producto base, denominado JDK (Java Development Toolkit) o actualmente J2SE (Java 2 Standard Edition).
- .. etc

Introducción a la Programación Java

Características.-

Qué incluye el J2SE (Java 2 Standard Edition)

- Herramientas para generar programas Java. Compilador, depurador, herramienta para documentación, etc.
- La JVM, necesaria para ejecutar programas Java.
- La API de Java (jerarquía de clases).
- Código fuente de la API (Opcional).
- Documentación.

Qué es el JRE (Java Runtime Environment)

- JRE es el entorno mínimo para ejecutar programas Java 2. Incluye la JVM y la API. Está incluida en el J2SE aunque puede descargarse e instalarse separadamente. En aquellos sistemas donde se vayan a ejecutar programas Java, pero no compilarlos, el JRE es suficiente.
- El JRE incluye el Java Plug-in, que es el 'añadido' que necesitan lo navegadores (Explorer o Netscape) para poder ejecutar programas Java 2. Es decir que instalando el JRE se tiene soporte completo Java 2, tanto para aplicaciones normales (denominadas 'standalone') como para Applets (programas Java que se ejecutan en una página Web, cuando esta es accedida desde un navegador).

Introducción a la Programación Java

Características.-

Qué se necesita para empezar

- El entorno mínimo necesario para escribir, compilar y ejecutar programas Java es el siguiente:

  - *J2SE (Java 2 Standard Edition) y la documentación (Se descargan por separado). Esto incluye el compilador Java, la JVM, el entorno de tiempo de ejecución y varias herramientas de ayuda. La documentación contiene la referencia completa de la API.*
  - *Un editor de textos. Cualquiera sirve. Pero un editor especializado con ayudas específicas para Java (como el marcado de la sintaxis, indentación, paréntesis, etc.) hace más cómodo el desarrollo.*
  - *De forma opcional puede usarse un Entorno de Desarrollo Integrado para Java (IDE). Una herramienta de este tipo resulta aconsejable como ayuda para desarrollar aplicaciones o componentes.*

Introducción a la Programación Java

Fase de Creación y Ejecución de un programa java.-

Código Java: Fichero Programa.java

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

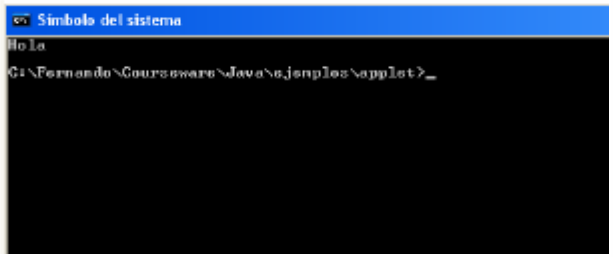
Compilación

```
javac Programa.java
```

Ejecución

```
java Programa
```

Resultado



Fase I: Editor

- Se crea un programa con la ayuda de un editor
- Se almacena en un fichero con extensión .java

Fase II: Compilador

- El compilador lee el código Java (fichero .java)
- Si se detectan errores sintácticos, el compilador nos informa de ello.
- Se generan los bytecodes, que se almacenan en ficheros .class

Fase III: Cargador de clases

El cargador de clases lee los bytecodes (ficheros .class):
Los bytecodes pasan de disco a memoria principal.

Fase IV: Verificador de bytecodes

El verificador de bytecodes comprueba que los bytecodes son válidos y no violan las restricciones de seguridad de la máquina virtual Java.

Fase V: Intérprete de bytecodes o compilador JIT

La máquina virtual Java (JVM) lee los bytecodes y los traduce al lenguaje que el ordenador entiende (código máquina).

NOTA: Conforme se ejecuta el programa, se hace uso de la memoria principal para almacenar los datos con los que trabaja la aplicación.

Introducción a la Programación Java

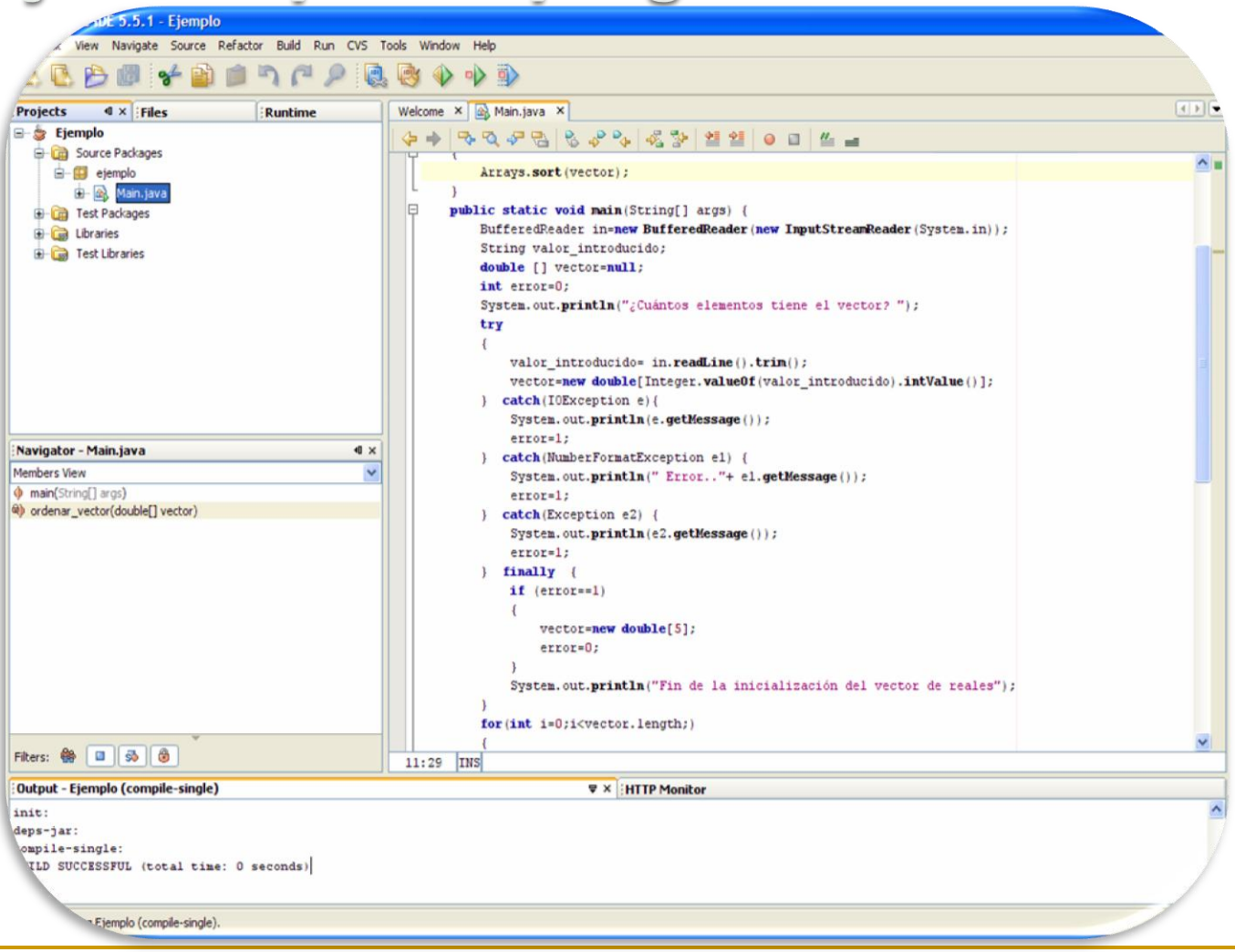
Webgrafías.-

			Tutorial de java realizado los por alumnos del 2º curso ASIR/IAW
			Índice de Contenidos
1	ABELLÁN MARTÍNEZ, MIGUEL	WEB	Clase y Objetos en java. Estructura básica.
2	ALCÁZAR PERIAGO, ALFONSO	WEB	Sistema de Excepciones en Java. Exception.
3	CLEMENTE LÓPEZ, JUAN MARCOS	WEB	Elementos del Lenguaje. Tipos de datos, variables y operadores
4	FUENTES LORCA, JOSÉ JUAN	WEB	Estructuras estáticas: Vectores y Matrices.
5	GARCÍA HERNÁNDEZ, PEDRO	WEB	Clase Object
6	GARCÍA MANCHÓN, PEDRO JOSÉ	WEB	Sentencias de Control.
7	GARCÍA VIDAL, JORGE ISMAEL	WEB	Clase Arrays.
8	GIMÉNEZ SÁNCHEZ, FERNANDO JAVIER	WEB	Clases para tipos primitivos o simples: String, Integer, Double.... java.lang.*
9	GÓMEZ PÉREZ, JUAN FRANCISCO	WEB	Herramientas CASE para el desarrollo de artefactos SW en Java.
10	LARIO SÁNCHEZ, FRANCISCO JESÚS	WEB	Introducción a la programación de escritorio. AWT/SWING
11	MARTÍNEZ CÁNOVAS, ELISABET	WEB	Clase System.
12	MARTÍNEZ CAVA, JUAN JOSÉ	WEB	Clase String
13	MARTÍNEZ PIERNAS, NOEL	WEB	Herencia en Java.
14	ORTEGA BERNABÉ, JUAN JOSÉ	WEB	Paradigma Orientado a Objetos. Fundamentos y origen de JAVA.
15	REVERTE GÓMEZ, PEDRO	WEB	Gestión de cadenas en Java
16	RODRÍGUEZ RODRÍGUEZ, ÓSCAR	WEB	Introducción a la jerarquía de colecciones en JAVA: Clase ArrayList y Clase Vector.
17	ROS GARCÍA, JUAN PEDRO	WEB	Introducción al tratamiento de ficheros en Java. E/S java.io.*
18	SEGURA MARTÍNEZ, JUAN	WEB	Clase StringBuffer.
19	VIVANCOS PÉREZ, JUAN	WEB	Clase Math.

Introducción a la Programación Java

Introducción a la Programación Java

Main.java mí primer programa.-



Introducción a la Programación Java

Main.java mí primer programa.-

Introducción a la Programación Java

Main.java mí primer programa.-

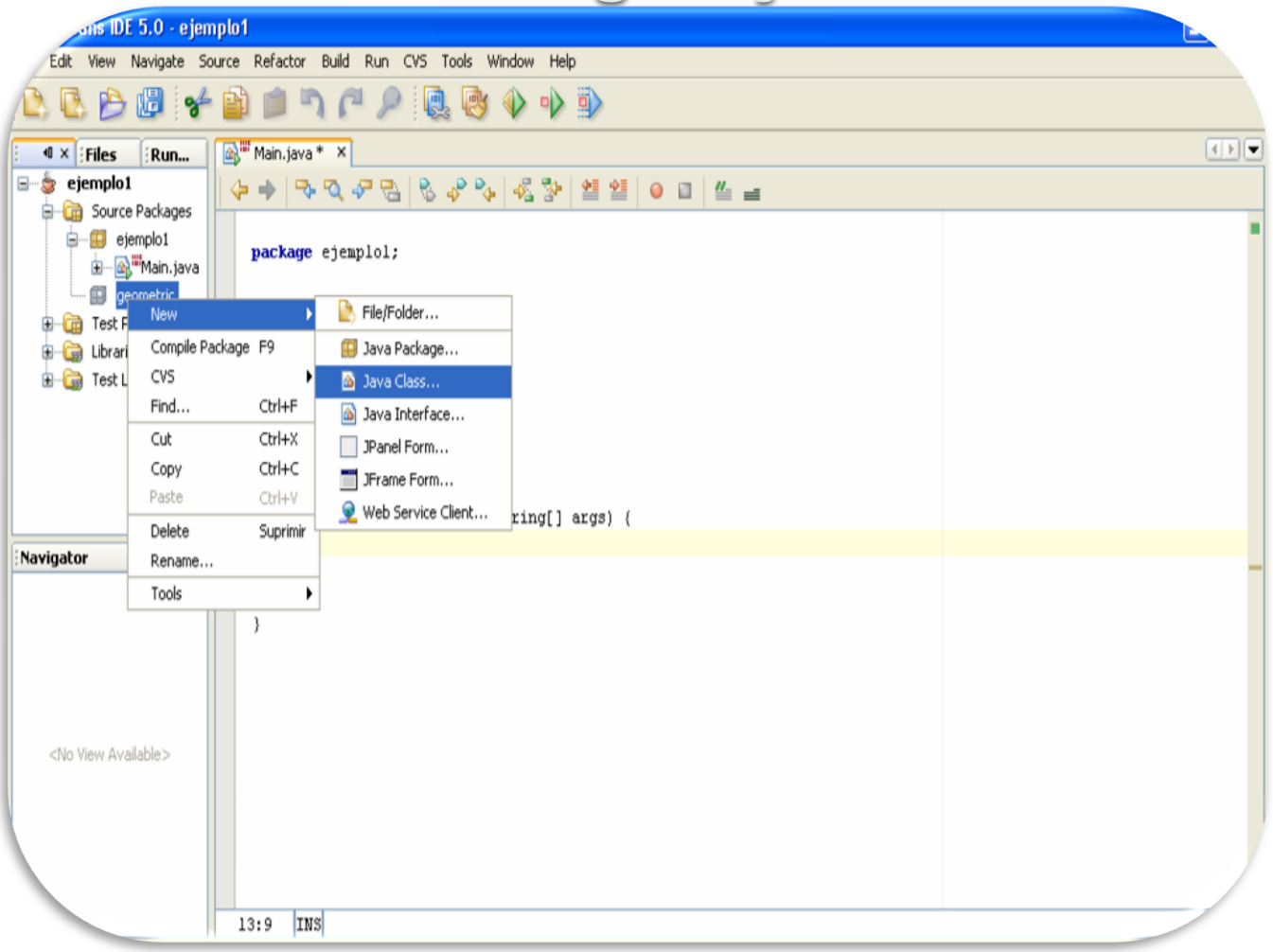
```
package ejemplo;

import java.io.*;
import java.lang.*;
import java.util.*;

.....................
```

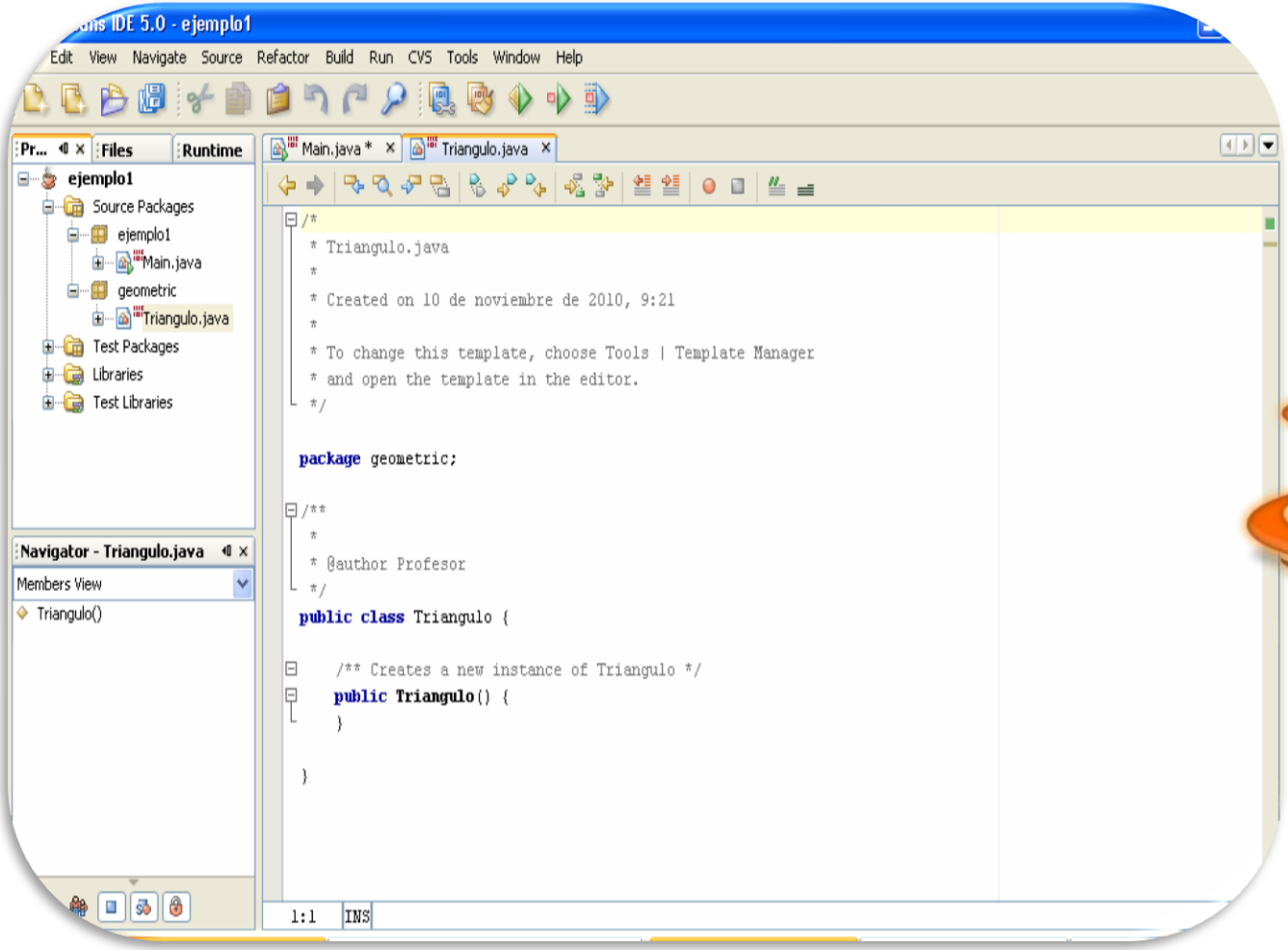
Introducción a la Programación Java

Mi primera clase Triangula.java.-



Introducción a la Programación Java

Mi primera clase Triangula.java.-



Introducción a la Programación Java

Herencia.-

Pero además de esta técnica de composición/creación de clases es posible pensar en casos en los que una clase es una extensión de otra. Es decir una clase es como otra y además tiene algún tipo de característica propia que la distingue. Por ejemplo podríamos pensar en la clase Empleado y definirla como:

```
package departamento;

public class Empleado {
   public String nombre;
   public int numEmpleado, sueldo;

   static private int contador = 0;

   public Empleado() {}

   public Empleado(String nombre, int sueldo)
   {
      this.nombre = nombre;
      this.sueldo = sueldo;
      this.numEmpleado = ++this.contador;
   }

   public void aumentarSueldo(int porcentaje) {
      this.sueldo += (this.sueldo * porcentaje / 100);
   }

   public String toString() {
      return "Num. empleado " + this.numEmpleado + " Nombre: " + this.nombre +  " Sueldo: " +
this.sueldo;
   }
```

Herencia.-

- En el ejemplo el Empleado se caracteriza por un nombre (String) y por un número de empleado y sueldo (enteros).
- La clase define un constructor que asigna los valores de nombre y sueldo y calcula el número de empleado a partir de un contador (variable estática que siempre irá aumentando), y dos métodos, uno para calcular el nuevo sueldo cuando se produce un aumento de sueldo (método aumentarSueldo) y un segundo que devuelve una representación de los datos del empleado en un String.(método toString).
- Con esta representación podemos pensar en otra clase que reúna todas las características de Empleado y añada alguna propia. Por ejemplo, la clase Ejecutivo. A los objetos de esta clase se les podría aplicar todos los datos y métodos de la clase Empleado y añadir algunos, como por ejemplo el hecho de que un Ejecutivo tiene un presupuesto.
- Así diríamos que la clase Ejecutivo extiende o hereda la clase Empleado. Esto en Java se hace con la clausula **extends** que se incorpora en la definición de la clase, de la siguiente forma:

Herencia.-

```
package departamento;

public class Ejecutivo extends Empleado
{
   public int presupuesto;

   public Ejecutivo()
   {
      super();
   }

   public Ejecutivo(String nombre, int sueldo,int p)
   {
     super(nombre,sueldo);
     this.presupuesto=p;
   }

   public String toString() {
      String s = super.toString();
      s = s + " Presupuesto: " + this.presupuesto;
      return s;
   }

   public void asignarPresupuesto(int p) {
      presupuesto = p;
   }
}
```

- Con esta definición un Ejecutivo es un Empleado que además tiene algún rasgo distintivo propio. El cuerpo de la clase Ejecutivo incorpora sólo los miembros que son específicos de esta clase, pero implícitamente tiene todo lo que tiene la clase Empleado.
- *A Empleado se le llama clase base o superclase y a Ejecutivo clase derivada o subclase.*

Redefinición de métodos. El uso de super.

Además se podría pensar en redefinir algunos métodos de la clase padre pero haciendo que métodos con el mismo nombre y características se comporten de forma distinta. Por ejemplo podríamos pensar en rediseñar el método toString de la clase Empleado añadiendo las características propias de la clase Ejecutivo.

```
public String toString()
{
    String s = super.toString();
    s = s + " Presupuesto: " + this.presupuesto;
    return s;
}
```

*Observese en el ejemplo el uso de **super**, que representa referencia interna implícita a la clase base (superclase). Mediante **super**.toString() se invoca el método toString de la clase Empleado*

Inicialización de clases derivadas

Cuando se crea un objeto de una clase derivada se crea implícitamente un objeto de la clase base que se inicializa con su constructor correspondiente. Si en la creación del objeto se usa el constructor no-args, entonces se produce una llamada implícita al constructor no-args para la clase base. Pero si se usan otros constructores es necesario invocarlos explícitamente.
En nuestro ejemplo dado que la clase método define un constructor, necesitaremos también un constructor para la clase Ejecutivo, que podemos completar así:

```
public Ejecutivo(String nombre, int sueldo,int p)
  {
    super(nombre,sueldo);
    this.presupuesto=p;
  }
```

Observese que el constructor de Ejecutivo invoca directamente al constructor de Empleado mediante super(argumentos). En caso de resultar necesaria la invocación al constructor de la superclase debe ser la primera sentencia del constructor de la subclase.

Herencia.-

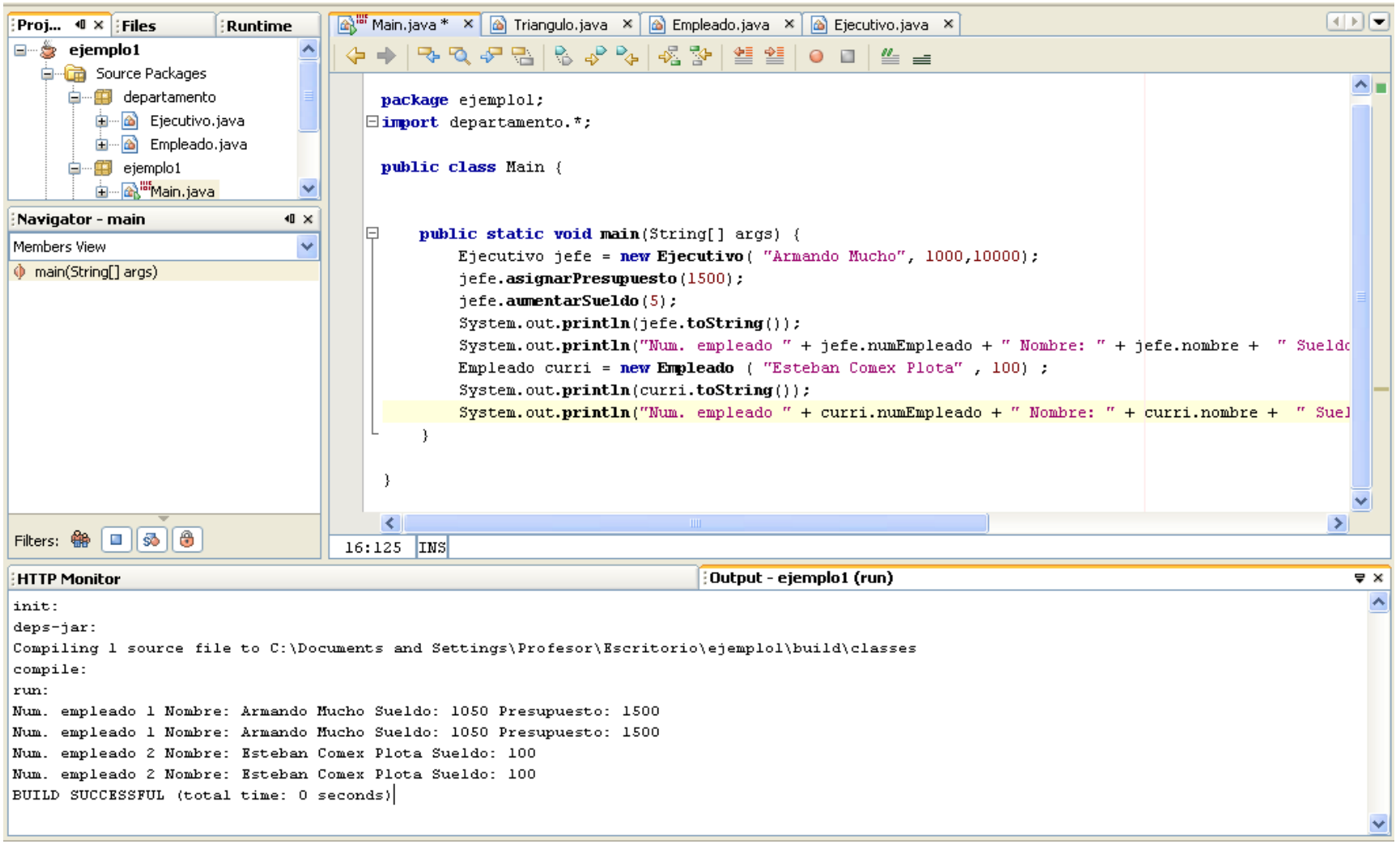
```
package ejemplo1;
import departamento.*;

public class Main {

    public static void main(String[] args) {
        Ejecutivo jefe = new Ejecutivo( "Armando Mucho", 1000,10000);
        jefe.asignarPresupuesto(1500);
        jefe.aumentarSueldo(5);
        System.out.println(jefe.toString());
        System.out.println("Num. empleado " + jefe.numEmpleado + " Nombre: " + jefe.nombre +
" Sueldo: " + jefe.sueldo + " Presupuesto: " + jefe.presupuesto);
        Empleado curri = new Empleado ( "Esteban Comex Plota" , 100) ;
        System.out.println(curri.toString());
        System.out.println("Num. empleado " + curri.numEmpleado + " Nombre: " + curri.nombre
+  " Sueldo: " + curri.sueldo);
    }
}
```

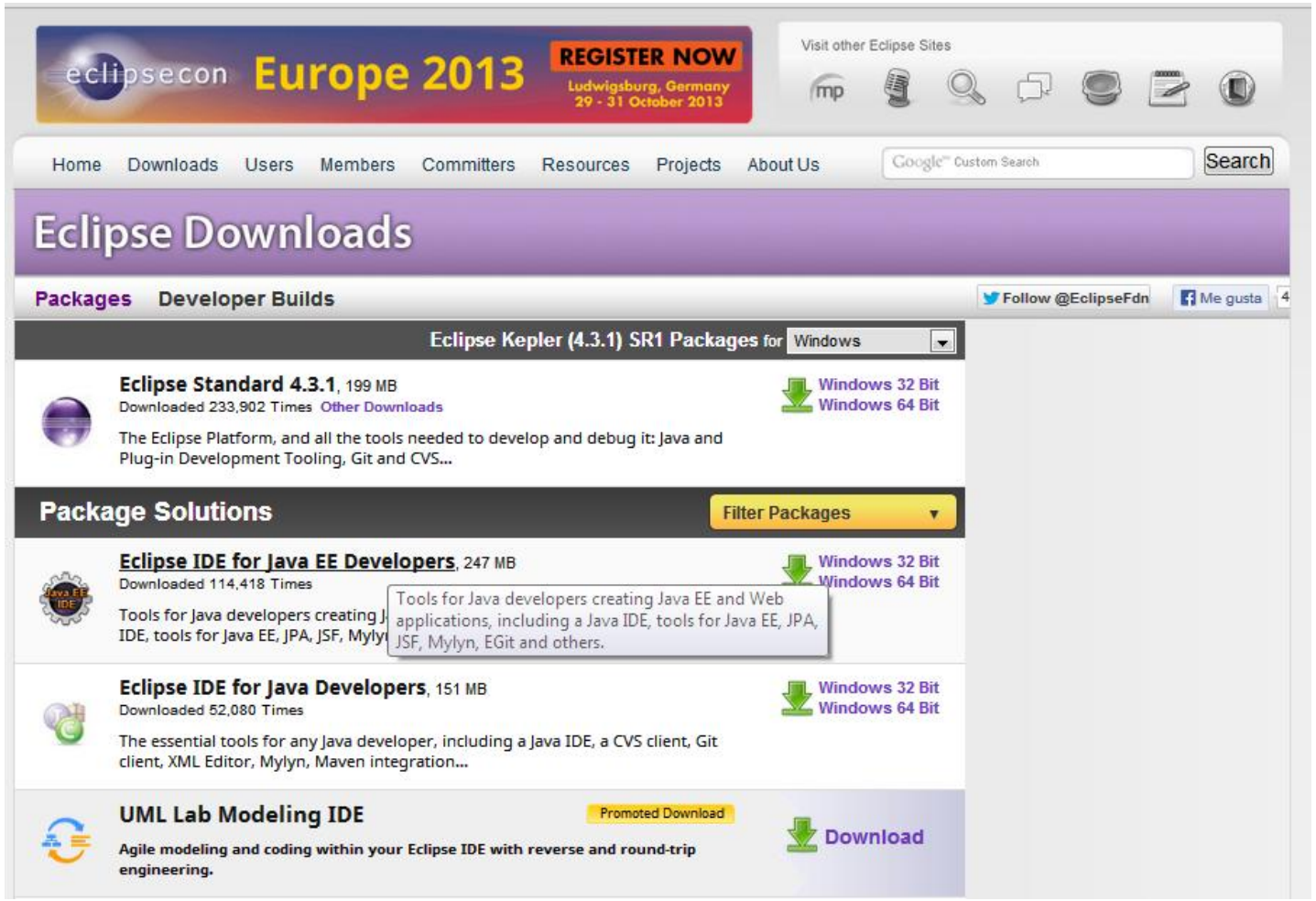
Introducción a la Programación Java

Herencia.-



Herramientas de Desarrollo.- *Netbeans*

Herramientas de Desarrollo.- *Eclipse*



Ejercicios Propuestos y Resueldos

```java
public class MetodosEstaticosEjercicio1{
    static String nombre="Carlos";
    static void dimeNombre(){
        String mote="cucu";
        System.out.println("Tu nombre es "+nombre+" y tu mote es "+mote);
    }
    static int mostrarEdad(int anNacimiento){
        return 2003-anNacimiento;
    }
    public static void main(String args[]){
        dimeNombre();
        System.out.println(nombre+" tiene "+mostrarEdad(1960)+
        " a"+(char)164+"os");
        System.out.println("FIN DEL PROGRAMA");
    }
}
```

Ejercicios


```
1  public class EjemplosMatematicos{
2      public static void main(String args[]){
3          System.out.println(Math.ceil(10.8));
4          System.out.println(Math.ceil(1.8956478));
5          System.out.println(Math.ceil(-5.96));
6          System.out.println(Math.ceil(-0.9));
7          System.out.println(Math.floor(10.8));
8          System.out.println(Math.floor(1.8956478));
9          System.out.println(Math.floor(-5.96));
10         System.out.println(Math.floor(-0.9));
11         System.out.println(Math.rint(10.2));
12         System.out.println(Math.rint(10.8));
13         System.out.println(Math.round(10.2));
14         System.out.println(Math.round(10.8));
15         System.out.println(Math.rint(3.891));
16         System.out.println(Math.rint(3.891*100)/100);
17     }
18 }
```