PRÁCTICA 0. Repaso al lenguaje Java	*2 H (1 Ses)*

Objetivos:

- Repaso al lenguaje de programación Java (Transparencias)
- Desarrollo implementación de varios ejemplos java, clases en Java y Array de objetos Java

Contenido:

Entorno de desarrollo

Para el desarrollo de las prácticas de MDP se utilizará el entorno Eclipse. Existen varias versiones de Eclipse que se pueden descargar desde http://www.eclipse.org/downloads/:

- Eclipse IDE for Java Developers : Desarrollo de aplicaciones básicas de Java. Será usado principalmente en 1º y 2º curso.
- Eclipse Standard: Desarrollo de aplicaciones básicas de Java + plugins para eclipse.
- Eclipse IDE for Java EE Developers : Desarrollo de aplicaciones J2EE (a partir de 3º curso)
- Eclipse IDE for C/C++ Developers : Entorno Eclipse para el desarrollo en programas de C/C++
- ..

De entre todos ellos seleccionaremos el primero de ellos[1]. La instalación consiste en:

- Descargar el fichero .zip o .tar.gz o ejecutable
- Descomprimirlo en un directorio o ejecutar la instalación
- Y finalmente ejecutar eclipse o eclipse.exe

A tener en cuenta:

- En eclipse todo el desarrollo debe realizarse englobado en un proyecto. El primer paso siempre será crear **un proyecto Java** o modificar uno existente.
- Existe una carpeta **src** donde se añadira el código fuente.

- **IMPORTANTE**: es recomendable crear paquetes (en minúscula)(es.unex.cum.mdp) . Para ello seleccionaremos del menú *File → New → Package*
- **IMPORTANTE**: para la creación de clase (primera letra en mayúscula)(es.unex.cum.mdp) se debe seleccionar del menú *File → New → Class*
- Se debe realizar la documentación javadoc según se va realizando el código de la clase.
- Para compilar y ejecutar se debe acceder al menú de compilación (Tecla abreviada para compilar CTR+F11)
- Se debe realizar una tabulación del código fuente que permita una mejor visión del mismo (Tecla abreviada CTR+SHIFT+F).
- Es bueno crear comentarios en el código
  - //: Comentario de una línea
  - /*: Comentario de varias líneas */
  - /** Comentario Javadoc */:

1 http://www.eclipse.org/downloads/

Ejemplo 1. Repaso de Programacion Orientada a Objetos

- Una **abstracción** denota las características esenciales de un objeto, distinguiéndolo de los demás objetos [Booch, 1996].
- **POO**: Se basa en el concepto de modelar objetos reales mediante objetos software.
  - **Clase** Una clase es una implementación total o parcial de un tipo abstracto de datos [Meyer, 1999]
    - es un conjunto de objetos que comparten una estructura común y un comportamiento común
  - **Un objeto** es una instancia de una clase en tiempo de ejecución [Meyer, 1999]
  - **El estado** de un objeto es el valor dinámico de cada una de las atributos definidas como la estructura de su clase [Booch, 1996]
  - **La identidad** de un objeto es aquella propiedad que lo distingue del resto de los objetos [Booch, 1996]
- **Paquetes.**
  - Los paquetes son una forma de organizar grupos de clases. Un paquete contiene un conjunto de clases relacionadas bien por finalidad, por ámbito o por herencia.
  - Los paquetes resuelven el problema del conflicto entre los nombres de las clases. Al crecer el número de clases crece la probabilidad de designar con el mismo nombre a dos clases diferentes.
  - Las clases tienen ciertos privilegios de acceso a los miembros dato y a las funciones miembro de otras clases dentro de un mismo paquete.
- **Encapsulamiento** es el proceso de almacenar en una abstracción los elementos que constituyen su estructura y su comportamiento [Booch, 1996]
  - La ocultación de información permite discernir entre qué partes de la abstracción están disponibles al resto de la aplicación y qué partes son internas a la abstracción [Meyer, 1999]
  - Algunos autores incluyen el concepto de ocultación de información dentro del de encapsulamiento
  - Para ello, los lenguajes de programación ofrecen diversos niveles de ocultación para sus miembros (atributos y métodos)
  - Los lenguajes Java y C++ utilizan las palabras reservadas: **public, protected y private** para realizar el control de acceso o visibilidad de los miembros de una clase
- **Los constructores** permiten garantizar la inicialización correcta y la limpieza de un objeto (no permite su uso si no ha sido creado) aportando un mecanismo de control y seguridad completo
  - El constructor se identifica con el mismo nombre que la clase, para evitar problemas de nomenclatura
  - Es un método especial invocado automáticamente en la creación de un objeto de forma transparente al programador
  - El constructor puede tener parámetros para permitir especificar como se crean los objetos
  - El constructor es un método especial que devuelve un objeto de la clase (aunque no se especifica)
  - El método constructor puede estar sobrecargado, es decir, puede exitir más de un método constructor, lógicamente con el mismo nombre pero con signatura diferente (número de parámetros y tipos diferentes)
  - En el caso de no implementar ningún constructor, Java proporciona automáticamente un constructor por dfecto.
  - This . Al acceder a variables de instancia de una clase, la palabra clave this hace referencia a los miembros de la propia clase
- **Destructor**: es bastante importante porque los objetos creados con new deben ser destruidos explicitamente. En java el recolector de basura libera automáticamente de todos los objetos
  - El recolector de basura solamente elimina toda la memoria creada mediante new.
  - Existe en todos los objetos un método finalize que se encarga de alguna limpieza importante relacionado con la clase, por ejemplo cerrar un fichero si esta abierto, finalizar una comunicación si esta abierta, pues de la memoria se encarga automáticamente Java.
  - De este modo el recolector de basura primero invocará al método finalize() y posteriormente libera la memoria.

- En la implementación de cualquier clase deben implementarse los siguientes métodos:
  - Constructores: por defecto, copia y paramatrizado/s
  - Selectores: getX() donde X es todo atributo establecido en la clase
  - Modificadores: setX() donde X es todo atributo establecido en la clase
  - equals(): para comparar objetos de la clase
  - toString(): Devolver la información asociada a un objeto

Importante: En todas las entregas finales de este año es obligatorio utilizar Javadoc en el código fuente así documentar internamente el código

```
/**
 * Implementacion de la clase Persona formada por el nombre, el dni y la edad.
 *
 * @author Luis Arévalo
 *
 */

package es.unex.mdp.practica0;

public class Persona {

	private String nombre;
	private String dni;
	private int edad;



	/**
	 *  Constructor por defecto de Persona. Inicializa a un valor por defecto todos sus atributos
	 */
	public Persona() {
		nombre = null;
		dni    = null;
		edad   = 0;
	}


	/**
	 * Constructor parametrizado encargado de inicializar a un valor recibido los atributos
	 *
	 * @param nombre Nombre de la persona
	 * @param dni DNI de la persona
	 * @param edad Edad de la persona
	 */
	public Persona(String nombre, String dni, int edad) {
		this.nombre = nombre;
		this.dni = dni;
		this.edad = edad;
	}

	/**
	 *  Constructor de copia que inicializa los atributos de una persona a partir de otro objeto persona
	 *
	 *  @param p Persona
	 */
	public Persona(Persona p) {
		nombre = p.nombre;
		dni    = p.dni;
		edad   = p.edad;
	}


	/**
	 * Devuelve el nombre de la persona
	 * @return El nombre de la persona
	 */
	protected String getNombre() {
		return nombre;
	}
```

```java
/**
 * Establece el nombre de la persona
 * @param nombre El nombre de la persona
 */
protected void setNombre(String nombre) {
	this.nombre = nombre;
}


/**
 * Devuelve el dni de la persona
 * @return DNI de la persona
 */
protected String getDni() {
	return dni;
}


/**
 * Establece el dni de la persona
 * @param dni DNI de la persona
 */
protected void setDni(String dni) {
	this.dni = dni;
}


/**
 * Devuelve la edad de la persona
 * @return Edad de la persona
 */
protected int getEdad() {
	return edad;
}


/**
 * Establece la edad de la persona
 * @param edad Edad de la persona
 */
protected void setEdad(int edad) {
	this.edad = edad;
}


/**
 * Recupera en formato String los valores de los atributos de la clase
 * toString
 */
@Override
public String toString() {
	return "Persona [nombre=" + nombre + ", dni=" + dni + ", edad=" + edad
			+ "]";
}


/**
 * Compara dos personas por DNI
 */
@Override
public boolean equals(Object o) {
	Persona p = (Persona) o;
       /* MAL, pues dni es un String y estariamos comparando sus referencias
       * if (dni==p.dni) return true;
```

```
            * else return false;
            */
            /* BIEN */
            return dni.equals(p.dni);
        }

}
```

Ejemplo 2. Ejemplo de uso de la Clase Persona

```
/**
 * Programa principal para usar la clase Persona
 */
package es.unex.mdp.practica0;

/**
 * @author Luis Arévalo
 *
 */
public class Main {

        /**
         * @param args Parametros de entrada de la consola
         */
        public static void main(String[] args) {

                Persona p  = new Persona();
                Persona p1 = new Persona("Paco", "12345678A", 37);
                Persona p2 = new Persona(p1);
                System.out.println(p);
                System.out.println(p1);

                Persona p3 = p2;
                p3.setEdad(100);
                System.out.println(p2);
                System.out.println(p3);

                Persona p4 = new Persona(p2.getNombre(), p2.getDni(), p2.getEdad());

                Persona[] vector = new Persona[10];

                vector[0] = new Persona();
                vector[1] = new Persona("", "",10);
                vector[2] = p4;
                for (int i=0;i<v.length;i++){
                        if (v[i]!=null)
                                System.out.println(v[i]);
                }
        }
}
```

VPL: ¿Cómo se realizarán las entregas?

Las entregas se realizarán en el campusvirtual alternativo (http://vpl-cum.unex.es/moodle) en la asignatura de Metodología y Desarrollo de Programas. Se trata de un espacio virtual donde existe una tarea que evalúa automáticamente las entregas proporcionando una calificación inmediata. Este plugin del campusvirtual alternativo tiene las siguientes particularidades:

- La evaluación se realiza en base a las entradas por teclado y a las salidas esperadas, es decir, si se esta implementando una calculadora y se recibe “2 4 +”, a partir de los valores de entrada el resultado esperado sería 6, de tal forma que cualquier otro valor implicaría una implementación incorrecta de la calculadora.

- Cuidado: Si la salida es “Suma: 10” las siguientes salidas serán erróneas: “suma: 10” (minúscula), “Suma:10”(sin espacio), “Suma: 10 “ (espacio al final), etc.

- NO SE puede realizar ningún syso en la introducción de valores por teclado, los únicos syso permitidos son los que se espera en la salida. Es decir, si se desea pedir un valor String sería:

```
Scanner in = new Scanner ( System.in );
syso("Dame la edad");
String n= in.nextLine();
```

- Se recomienda antes de subir la entrega, probarla en un ordenador local. Por el punto anterior, la ejecución puede ser muy tediosa, por lo que se recomienda utilizar la clase **Teclado** que se proporciona:

```
public class Teclado {
    ….
  private boolean CV= false; //Este atributo estará a false cuando se hagan las pruebas locales (en
                             //ordenador) y se pondrá a true cuando se suba al campusvirtual
    ..
```

- La lectura por teclado se debe realizar si tenemos un objeto Teclado t de la siguiente forma:
  - Para leer cadena: String x= t.literalConString(“Dame el nombre”);
  - Para leer enteros: int x=t.literalConEntero(“Dame la edad”);
  - ….

- Hay que tener cuidado con la codificación de los caracteres. En principio esta configurado para que soporte acentos (sólo en los comentarios) y ñ (aunque la ñ NUNCA se debe usar en programación). Si se observa que en una ejecución da fallo por un acento, quitarlos y avisar al profesorado.

- Cuidado con el siguiente ejemplo, **sólo puede haber una instancia Teclado**

Código que falla en CV alternativo, no en Eclipse

```
public class XXXX(){
  public void yyyy(){
        Teclado t= new Teclado();
        ....
   }
   public void zzzz(){
        Teclado t= new Teclado();
        ....
  }
}
```

Solución

```
public class XXXX(){
  private Teclado t= new Teclado();
  public void yyyy(){

        ....
   }
   public void zzzz(){

        ....
  }
}
```

¿Qué se debe entregar? (**Mirar siempre el enunciado del CV alternativo**)

Cada alumno deberá implementar una clase que se le ocurra distinta que este formada:

- Un atributo String, un atributo entero y un atributo float (en este orden)
- Los constructores: por defecto y al menos un parametrizado
- Los accesores y mutadores
- El método toString() y equals()

IMPORTANTE: El código fuente tiene que estar en un paquete denominado: **es.unex.cum.mdp.sesion0**

A continuación se realizará una clase principal denominada **Main.java** donde se definirá un vector de esa clase e implementar las siguientes operaciones:

1. Inicializar el vector: Se reserva tamaño para el array de objetos pidiendo para ello por teclado el tamaño del array.

2. Rellenar el vector con tanto objetos como tamaño del vector. Se pedirá por teclado en el orden indicado (String, entero, float) y cuando se suba sin realizar ningún syso

3. Listar todos los objetos del vector en el siguiente formato por cada objeto
   [String, int, float] es decir, podría ser algo como: [renault, 110, 18000.45]

4. Obtener la suma y la media del campo de tipo entero con el siguiente formato:
   Suma: XXX Media: YYY

Donde XXX es la suma de los valores numéricos de los objetos del vector e YYY es la media.

5. Leer un valor float y comprobar si se encuentra en el array. La salida será la siguiente siendo XX el valor introducido:
   Si esta → "El valor XX se encuentra en el array"
   Si no esta → "El valor XX no se encuentra en el array"

6. Mostrar el valor más alto del array del campo entero mostrando "El valor mas alto es XXX y esta en la posicion YYY" donde XXX es el valor e YYY es su posición dentro del array. Si hay dos valores o más valores iguales, se seleccionará el último de ellos.

Bibliografía

- Piensa en Java. 4ª Edición. Bruce Eckel. Pearson Prentice Hall.
- Análisis y diseño orientado a objetos. Cuaderno Didácticos. Juan Manuel Cueva. Capítulo 3.
- [Larman99] UML y Patrones. C. Larman. Prentice Hall, 1999.
- [Booch99]: El Lenguaje Unificado de Modelado. G. Booch, J. Rumbaugh, I. Jacobson. Addison Wesley Iberoamericana, 1999.