Departamento de Ingeniería Telemática
Escuela Superior de Ingenieros
Universidad de Sevilla

ENTORNO DE PROGRAMACIÓN (I)

OBJETIVO

El objetivo de esta práctica es proporcionar herramientas básicas que permitan desarrollar programas dentro del lenguaje de programación Java. En esta práctica se presentan los comandos más básicos, así como sus opciones más importantes, que se deben conocer y manejar para la correcta realización de las prácticas sobre Java de la asignatura Fundamentos de Programación II.

1. Introducción

Al igual que en las prácticas sobre C realizadas anteriormente en esta asignatura y en la asignatura Fundamentos de Programación I, estas prácticas se realizarán en los equipos disponibles en el Centro de Cálculo de la Escuela Técnica Superior de Ingeniería y se utilizará Linux como sistema operativo. Por ello se recomienda que para resolver cualquier duda relativa al manejo del entorno así como del sistema operativo, se utilice la documentación relativa a las prácticas de la citada asignatura (Apuntes de "*Fundamentos de Programación I*", Departamento de Ingeniería Telemática, Universidad de Sevilla).

1.1. Entorno de ejecución

Al conjunto de todos los elementos necesarios para ejecutar una aplicación Java se denomina JRE (*Java Standard Edition Runtime Envionment*). Este entorno está formado por:

- Máquina virtual de Java (JVM, *Java Virtual Machine*):

  Núcleo básico que contiene el intérprete para el lenguaje Java (java). Además, gracias a la máquina virtual de Java, es posible ejecutar un mismo programa en diferentes máquinas con sistemas operativos diferentes.

- Un conjunto de librerías:

  Implementan las clases e interfaces básicas y fundamentales definidas en Java, por ejemplo los paquetes `java.lang` y `java.io`.

- Otros componentes.

1.2. Entorno de desarrollo

Al conjunto de todos los elementos necesarios para desarrollar y ejecutar una aplicación Java se denomina JDK (*Java Standard Edition Development Kit*). Este entorno está formado por:

- Entorno de ejecución de Java (*JRE*):
  Nos permitirá ejecutar una aplicación Java mediante la ejecución del intérprete (`java`).
- Herramientas para la creación y desarrollo de aplicaciones Java:
  Compilador (`javac`), depurador (`jdb`), empaquetador (`jar`) o generador de documentación (`javadoc`) son algunas de estas herramientas. Todas ellas tienen en común que su ejecución se realiza mediante la línea de comandos (ninguna de ellas ofrece una interfaz gráfica).

1.3. Proceso de creación de una aplicación en Java

El primer paso para crear una aplicación es la obtención del algoritmo que va a implementar la aplicación. Una vez realizadas las fases de análisis (comprensión del problema) y diseño del programa, ya se puede acometer la codificación del programa en Java. Una vez que tenemos los ficheros de código fuente (ficheros `.java`), se utiliza el compilador (`javac`) para obtener los ficheros bytecode (ficheros `.class`). Cuando se han obtenido todos los ficheros `.class` a partir de los ficheros `.java`, recurrimos al intérprete (`java`) de la máquina virtual de Java para la ejecución de la aplicación que hemos creado. Este proceso se muestra en la Figura 1.

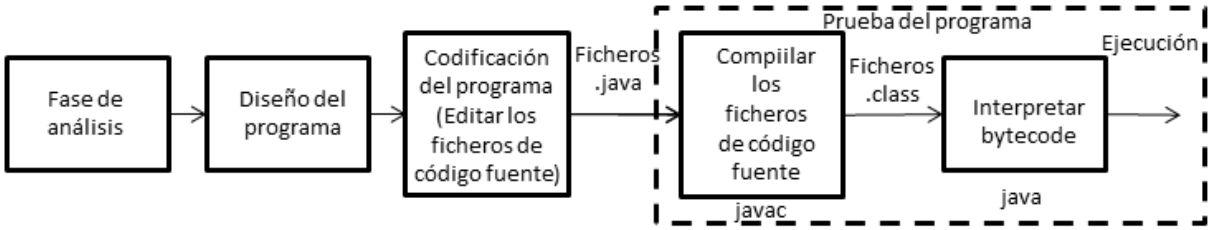

Figura 1: Proceso de creación de una aplicación Java

Por último, ya sólo queda abordar la fase de prueba del programa, con el fin de detectar los errores de codificación cometidos y realizar las modificaciones oportunas.

1.4. Documentación

Existe múltiple bibliografía, tanto en formato físico como formato digital, que proporciona abundante información sobre Java. Oracle ofrece su propia documentación sobre el lenguaje Java, que se recoge en las siguientes páginas web:

- Página principal:
  `http://www.oracle.com/technetwork/java/index.html`.

- Documentación general de Java SE (Standard Edition):
  `http://download.oracle.com/javase/6/docs/index.html`.

- API (*Application Programming Interface*) de Java SE 6:
  `http://download.oracle.com/javase/6/docs/api/`.

2. Herramientas

Aunque el entorno de desarrollo de Java nos proporciona diversas herramientas, en el ámbito de la asignatura sólo será necesario conocer las herramientas que se detallan a continuación. Se recomienda analizar y comprender en profundidad la finalidad de cada herramienta así como las opciones más importantes que ofrece, de forma que su futuro uso no suponga una dificultad añadida en el desarrollo de un programa.

2.1. Compilador (javac)

El compilador javac se encarga de generar ficheros *bytecode* (ficheros `.class`) a partir de los ficheros de código fuente. La sintaxis de este programa es la siguiente:

`javac [opciones] ficheros_java`

donde:

- **`[opciones]`** modifican el comportamiento del compilador. Algunas de las opciones más frecuentes son las siguientes:

  `-d directorio` Opción estándar encargada de indicar al compilador el directorio donde ubicar los archivos `.class` tras la compilación de los ficheros fuente `.java`.

  `-g` Opción estándar encargada de generar toda la información necesaria para depurar el programa mediante el uso del depurador.

  `-classpath` *ruta* Opción estándar encargada de indicar al compilador la ruta donde se encuentran los ficheros `.class` necesario para compilar el fichero de código fuente indicado.

  `-v` Opción estándar encargada de imprimir la versión del compilador que se está utilizando.

  `-Xlint` Opción no estándar3 encargada de imprimir todos los avisos que se produzcan al compilar el código fuente.

- **`ficheros_java`** indica los ficheros de código fuente que se desean compilar. Estos ficheros deben tener extensión `.java`. De esta forma, la clase `ClaseA` se debería guardar en un fichero con nombre `ClaseA.java` para que al compilarlo obtengamos el fichero `ClaseA.class`.

Ejemplo de uso

A continuación se muestra el fichero de código fuente `Ejercicio00.java`.

```
public class Ejercicio00 {
    public static void main( String args[] ) {
        System.out.println( "El programa se ha ejecutado." );
    }
}
```

La orden que se debe ejecutar para realizar la compilación del fichero `Ejercicio01.java`, indicándole al depurador que nos muestre todos los avisos y que genere toda la información necesaria para poder utilizar el depurador, es la siguiente:

```
>javac -g -Xlint ./Ejercicio00.java
```

En el caso anterior la compilación se ha realizado encapsulando la clase en un paquete por defecto y que no tiene nombre. Se puede modificar el lugar donde se ubica el fichero fuente. Para ello se ubica el fichero `Ejercicio00.java` en el directorio `fp2/poo/practica1` en el directorio de inicio de sesión. Esta estructura de directorios va asociado a la estructura de paquetes correspondiente. Se deberá, por tanto, añadir la línea `package fp2.poo.practica1;` para la creación de un paquete con el nombre `fp2.poo.practica1` que contendrá la clase `Ejercicio00`. Esto se muestra en el código siguiente:

```
./*
 *  Fichero: Ejercicio00.java
 *
 *  Fundamentos de Programacion II. GITT.
 *  Departamento de Ingenieria Telematica
 *  Universidad de Sevilla
 *
 *  Sin copyright
 */

package fp2.poo.practica1;

import java.lang.*;


/**
 * Descripcion: Esta es una clase de ejemplo.
 *
 * @version version 1.0 Abril 2011
 * @author  Fundamentos de Programacion II
 */
public class Ejercicio00 {
    public static void main( String args[] ) {
        System.out.println( "El programa se ha ejecutado." );
    }
}
```

La orden que se debe ejecutar para realizar la compilación del fichero `Ejercicio00.java`, indicándole al depurador que nos muestre todos los avisos y que genere toda la información necesaria para poder utilizar el depurador, es la siguiente:

```
>javac -g -Xlint ./fp2/poo/practica1/Ejercicio00.java
```

2.2. Intérprete (java)

El intérprete `java` se encarga de ejecutar una aplicación Java. Para ello, inicia todo el entorno necesario para la ejecución, carga la clase indicada y comienza la ejecución del método `main` que debe estar presente en el fichero `fichero_class`. La sintaxis de este programa es la siguiente:

```
java [opciones] fichero_class [argumentos]
java [opciones] -jar fichero_jar [argumentos]
```

donde:

- `[opciones]` modifican el comportamiento del compilador. Algunas de las opciones más frecuentes son las siguientes:

  `-classpath` *ruta* Opción estándar encargada de indicar al intérprete la ruta a los archivos `.class` o `.jar` que se desean ejecutar. El contenido de esta opción sobreescribe el contenido de la variable de entorno `CLASSPATH`. En caso de que no se haya especificado la mencionada variable y tampoco se utilice esta opción, la ruta utilizada será el directorio de trabajo (.). Cada uno de los caminos se indicarán separados por : (dos puntos). Esta opción también se denomina `-cp ruta`.

  `-showversion` Opción estándar encargada de imprimir, previamente a la ejecución de la aplicación, la versión del intérprete que se está utilizando.

  `-verbose` Opción estándar que indica al intérprete que muestre de manera explícita la información de cada clase cargada y cada evento del recolector de basura.

  `-version` Opción estándar encargada de imprimir la versión del intérprete que se está utilizando.

- `clase` indica al intérprete el nombre de la clase en la que se desea empezar la ejecución.

- `-jar` fichero.jar indica al intérprete el fichero `.jar` que contiene la aplicación Java a ejecutar.

- `[argumentos]` indica los argumentos que se desean pasar a la aplicación por línea de comandos.

Ejemplo de uso

A partir del ejemplo anterior la orden que se debe ejecutar para iniciar la ejecución de la aplicación es la siguiente:

```
>java fp2.poo.practica1.Ejercicio00
El programa se ha ejecutado .
```

2.3. Empaquetador (jar)

El programa **`jar`** (*Java ARchive*) se encarga de empaquetar un conjunto de archivos dentro de un único archivo jar, utilizando el formato de fichero ZIP. La sintaxis de este programa depende de la acción que deseamos realizar:

`jar c[v]f[e] archivo_jar [clase_inicial] [-C directorio] f_class`
(Crear un archivo jar)

`jar u[v]f[e] archivo_jar [clase_inicial] [-C directorio]` `f_class`
(Actualizar un archivo jar)

`jar x[v]f archivo_jar`
(Extraer los archivos de un archivo jar)

`jar t[v]f archivo_jar`
(Ver el contenido de un archivo jar)

donde:

- La opción `v` indica al empaquetador que muestre por pantalla toda la información relativa a la ejecución de la orden del usuario.

- La opción `f` indica al empaquetador el fichero `.jar` que se desea crear (con la opción `c`), actualizar (con la opción `u`), extraer (con la opción `x`) o ver el contenido (con la opción `t`).

- La opción `e` indica al empaquetador cuál es la clase que contiene la función que inicia el funcionamiento del programa (método `main`).

- `archivo_jar` indica el nombre del fichero `.jar` que se desea crear. Si es necesario, este parámetro contendrá la ruta a la ubicación en la que se debe crear el archivo.

- `clase_inicial` indica al empaquetador el nombre de la clase que contiene la función main por la que se debe empezar la ejecución de la aplicación.

- La opción `-C directorio` indica al empaquetador el directorio donde se encuentran los ficheros que se indican a continuación en la orden que se está ejecutando. Este opción podrá aparecer, por tanto, delante del nombre de cada uno de los ficheros a empaquetar.

- `ficheros_class` indican al empaquetador los ficheros a incluir en el archivo `.jar`. Se puede indicar tanto los ficheros como los directorios completos (que contendrán archivos `.class`) que se desean incluir.

Ejemplo de uso

A partir del ejemplo anterior, la orden que se debe ejecutar para empaquetar en un archivo `Ejercicio00.jar` la aplicación creada es la siguiente:

```
 >jar cvfe Ejercicio00.jar  fp2.poo.practica1.Ejercicio00 ./fp2/poo/practica1/Ejercicio00.class
manifest agregado
agregando: fp2/poo/practica1/Ejercicio00.class (entrada = 589) (salida = 367) (desinflado 37%)
```

Una vez creado el archivo `Ejercicio00.jar`, la orden necesaria para la ejecución de la aplicación directamente a través de este archivo es la siguiente:

```
> java -jar Ejercicio00.jar
El programa se ha ejecutado .
```

2.4. Gestión de documentación (`javadoc`)

El programa `javadoc` se encarga de generar de manera automática la documentación correspondiente al código fuente que se ha creado. Para ello el programador debe añadir ciertos comentarios en su código fuente (siguiendo unas sencillas reglas) de forma que el programa `javadoc` pueda reconocerlas y así crear la documentación correspondiente. Se ampliará en una práctica posterior.

2.5. Depurador (jdb)

El programa `jdb` es una herramienta muy útil para la depuración de aplicaciones escritas en Java. Se ampliará en una práctica posterior.

3. Entorno

Aunque generalmente no es necesario configurar ningún parámetro del entorno de desarrollo, sí es conveniente conocer determinados aspectos del entorno. Estos aspectos son la variable de entorno `CLASSPATH` y la estructura de directorios en la que se almacenan todos los elementos que componen el entorno.

3.1. Ruta de ejecución (classpath)

Ya se ha comentado anteriormente la posibilidad de indicar a las herramientas del entorno (`java`, `javac`. . .) la ubicación de los ficheros con los que se desea trabajar. Existe otra posibilidad para realizar esta indicación: la variable de entorno `CLASSPATH`. A continuación se va a indicar cómo trabajar con esta variable de entorno:

- Consultar el valor almacenado en la variable de entorno `CLASSPATH`:
  El comando con el que se consulta el valor almacenado en la variable de entorno es:

  ```
  echo $CLASSPATH
  ```

  En el resultado que obtenemos se encuentran todos los directorios, separados por el carácter :, que dan valor a la variable de entorno.

- Establecer la variable de entorno `CLASSPATH`:
  El comando con el que se da contenido a la variable de entorno es:

  ```
  export CLASSPATH=directorio:directorio:…
  ```

  representa al `directorio` que se desea añadir como valor a la variable de entorno.

- Eliminar el contenido de la variable de entorno `CLASSPATH`:
  El comando con el que se elimina el contenido de la variable de entorno es:

  ```
  unset CLASSPATH
  ```

Por último, es necesario indicar que el tiempo de vida de la variable de entorno es el terminal en el que se ha fijado. Por tanto, cada vez que se abra un terminal es necesario fijar de nuevo el valor que queramos dar a la variable de entorno `CLASSPATH`.

3.2. Estructura de archivos del JDK

Con el objetivo de proporcionar una visión más completa del entorno de programación JDK, a continuación se muestran los directorios más importantes que forman parte del mismo:

`/usr/lib/jvm/java-6-sun/`	Directorio donde se almacena todo el software que compone el entorno.
`/usr/lib/jvm/java-6-sun/bin/`	Directorio donde se almacenan todos los ejecutables del entorno.
`/usr/lib/jvm/java-6-sun/lib/`	Directorio donde se almacenan las clases del entorno que no pertenecen al núcleo de la máquina virtual.
`/usr/lib/jvm/java-6-sun/jre/`	Directorio donde se almacena todo el software que necesita la máquina virtual de Java para su ejecución.

4. Ejercicios

4.1. Uso de las herramientas básicas

1. Descargue de la plataforma virtual los ficheros proporcionados para esta práctica (`Calculadora.java` y `Ejercicio01.java`) cuyo contenido se muestra a continuación. Estos ficheros al descomprimirlos se ubicarán en los directorios.

```
/*
 *  Fichero: Ejercicio01.java
 *
 *  Fundamentos de Programacion II. GITT.
 *  Departamento de Ingenieria Telematica
 *  Universidad de Sevilla
 *
 */


package fp2.poo.practica1;

import java.lang.*;


/**
 * Descripción: Esta es una clase de prueba que contiene el metodo
 *              main para probar la clase Calculadora.
 *
 * @version version 1.0 Abril 2011
 * @author  Fundamentos de Programacion II
 */
public class Ejercicio01 {

    /**
     *  Este metodo invoca a la clase Calculadora.
     *  Realiza la instanciación de la clase e invoca
     *  sus metodos.
     */
    public static void main( String args[] ) {
        int i = 0; /* Variables locales a main */
        int j = 0;
        Calculadora calc = new Calculadora ();

        /* Primera operacion : 1 + 2 ( Resultado : 3).*/
        i = 1;
        j = 2;
        System.out.println ("El resultado de sumar " + i + " y "
                            + j + " es " + calc.suma( i, j));

        /* Segunda operacion : Factorial del resultado anterior .
         * ( Resultado : 3! = 6)
         */
        i = calc.getMemoria(  );
```

```
        j = calc.factorial( i );
        System.out.println( "El factorial de " + i + " es " + j);

        /* Tercera operacion : Dividimos el ultimo resultado por 4.
         * ( Resultado : 1).
         */
        System.out.println( "El resultado de dividir " + j +
                            " y 4 es " + calc.divide( j, 4));

       /* Cuarta operacion : Sumamos el contenido de la memoria a 20.
        * ( Resultado : 23).
        */
      i = 20;
      System.out.println( "El resultado de sumar "
                          + calc.getMemoria() + " y " + i
                          + " es " + calc.suma( i ));
    }
}
```

```
/*
 *  Fichero: Calculadora.java
 *
 *  Fundamentos de Programacion II. GITT.
 *  Departamento de Ingenieria Telematica
 *  Universidad de Sevilla
 *
 */


package fp2.poo.practica1;

import java.lang.*;


/**
 * Descripcion: Esta una clase es un ejemplo de implementacion y uso
 *              de los metodos de una clase.
 *
 *
 * @version version 1.0 Abril 2011
 * @author  Fundamentos de Programacion II
 */
public class Calculadora {

    /** Atributo privado donde se almacena los resultados obtenidos. */
    private int memoria ;

    /**
     *  Constructor de la clase Calculadora.
     *
     *  Parametros: No hay par metros.
     */
    public Calculadora() {
        this.memoria = 0;
    }

    /**
     *  Descripcion: Este metodo realiza la suma de los dos parametros
     *               proporcionados.
```

```
     *
     */
    public int suma( int param1 , int param2 ) {
        int resultado = 0; // Almacena el resultado de la suma .

        resultado = param1 + param2 ;
        this.memoria = resultado ;   /* Se almacena en memoria.*/
        return resultado ;
    }

    /**
     *  Descripcion: Este metodo realiza la suma del valor
     *               proporcionado como par metros con el valor
     *               almacenado en memoria.
     */
    public int suma( int param ) {
        int resultado = 0;

        resultado = param + this.getMemoria();
        this.memoria = resultado ; /* Lo almacenamos en memoria. */
        return resultado ;
    }

    /**
     *  Descripcion: Este metodo realiza la division de dos valores
     *               proporcionados como par metros.
     *
     */
    public int divide( int param1 , int param2 ) {
        int resultado = 0; // Almacena el resultado de la division.

        resultado = param1 / param2 ;
        this.memoria = resultado ; /* Lo almacenamos en memoria. */
        return resultado ;
    }

    /**
     *  Descripcion: Este metodo calcula el factorial de un numero.
     *
     */
    public int factorial( int param ) {
        int resultado = 1;

        for (int i = param; i > 0; i -- ){
            resultado = resultado * i;
        }
        return resultado;
    }

    /**
     *  Descripcion: Este metodo devuelve el valor almacenado en
     *               memoria.
     */
    public int getMemoria() {
        return this.memoria;
    }
}
```

2. Compile el fichero `Calculadora.java` utilizando las opciones `-g` y `-Xlint`. Observe el resultado.
3. Compile el fichero `Ejercicio01.java` utilizando las opciones `-g` y `-Xlint`. Observe el resultado, debe ser parecido al que se proporciona a continuación:

```
Ejercicio01.java:33: cannot find symbol
        Calculadora calc = new Calculadora ();
        ^
  symbol:   class Calculadora
  location: class Ejercicio01
Ejercicio01.java:33: cannot find symbol
        Calculadora calc = new Calculadora ();
                           ^
  symbol:   class Calculadora
  location: class Ejercicio01
2 errors
```

Incluya la siguiente sentencia import en el fichero `Ejercicio01.java` y vuelva a compilar.

```
import fp2.poo.practica1.Calculadora;
```

4. Ejecute la aplicación cuya función principal se encuentra en la clase Ejercicio01. Observe el resultado.
5. Cree el archivo `Ejercicio01.jar` en el que se encuentren todos los ficheros `.class` necesarios para la ejecución de la función principal que se encuentra en la clase Ejercicio01.
6. Elimine todos los ficheros `.class` del directorio `fp2/poo/practica1`.
7. Ejecute la aplicación que se encuentra almacenada en el fichero `Ejercicio01.jar`. Observe el resultado.
8. Utilice el siguiente fichero `makefile` que automatiza las tareas de compilación de los ficheros `.class` y la construcción del fichero `Ejercicio01.jar`, proporcionado en esta práctica, y que se detalla a continuación.

```
#
#   Makefile de ejemplo para la compilación, creación del jar y ejecución
#

# CLASEPPAL es el nombre de la clase que contiene el metodo principal (main)
CLASEPPAL=Ejercicio01

# CLASEAUX es el nombre de la clase que se esta probando.
CLASEAUX=Calculadora

RUTACLASE=fp2/poo/practica1/


PRINCIPAL=fp2.poo.practica1.Ejercicio01


ejecutaJ: $(CLASEPPAL).jar
	java -jar $(CLASEPPAL).jar

$(CLASEPPAL).jar: $(RUTACLASE)$(CLASEPPAL).class $(RUTACLASE)$(CLASEAUX).class
	jar    cvfe        $(CLASEPPAL).jar    $(PRINCIPAL)        $(RUTACLASE)$(CLASEPPAL).class
$(RUTACLASE)$(CLASEAUX).class


$(RUTACLASE)$(CLASEPPAL).class: $(RUTACLASE)$(CLASEPPAL).java
	javac -g -Xlint $(RUTACLASE)$(CLASEPPAL).java


$(RUTACLASE)$(CLASEAUX).class: $(RUTACLASE)$(CLASEAUX).java
	javac -g -Xlint $(RUTACLASE)$(CLASEAUX).java
```

9. Utilice la herramienta `make` para comprobar que el fichero `makefile` funciona correctamente. Por ejemplo, pruebe a realizar sucesivas ejecuciones.

4.2. Organización de los ficheros.

Se propone, a partir del ejercicio anterior, construir un entorno de trabajo que le permita trabajar de manera ordenada con el código que realizará en cada una de las prácticas de la asignatura.

1. Cree el directorio **`p1Mejorada`** en el directorio de inicio de sesión y sitúese en él.
2. Cree los siguientes subdirectorios:
   - `src` En este directorio se va a almacenar todo el código fuente que se escriba (ficheros `.java`) durante las sesiones prácticas.
   - `bin` En este directorio se van a almacenar todos los ficheros bytecode que se genere (ficheros .class) a partir de los ficheros almacenados en los directorios `src` y Ejercicios.
   - `jar` En este directorio se van a almacenar todos los ficheros `.jar` que se generen.

3. Cuelgue del directorio `src` de **`p1Mejorada`** el directorio y subdirectorios de `fp2`, y deje solamente el código fuente (`.java`), de forma que en el directorio **`p1Mejorada/src/fp2/poo/practica1`** aparezcan los ficheros `Calculadora.java` y `Ejercicio01.java`.
4. Elimine todos los ficheros `.class` y `.jar` que se hayan generado.

5. Partiendo del fichero makefile proporcionado anteriormente, crear un fichero makefile en el directorio **`p1Mejorada`**, que automatice las tareas de compilación, creación del jar y ejecución, generando los ficheros .class en el subdirectorio `./bin`, y el fichero `.jar` en el directorio `./jar`.
6. Ejecute la aplicación que se encuentra almacenada en el fichero `Ejercicio01.jar`.