Área de Ingeniería Telemática
Departamento de Ingeniería de Sistemas y Automática
Escuela Superior de Ingenieros
Universidad de Sevilla

ENTORNO DE PROGRAMACIÓN (II)

OBJETIVO

En esta práctica se presenta la herramienta `jdb` para la depuración. En esta práctica se realiza la implementación de una interfaz para mostrar la depuración del código.

1. Depurador (jdb).

El programa `jdb` es una herramienta muy útil para la depuración de aplicaciones escritas en Java. La sintaxis de este programa es la siguiente:

`jdb [opciones] clase [argumentos]`

donde:

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

  `-classpath ruta` Opción encargada de indicar al depurador la ruta a los archivos `.class` que se desean depurar. 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 :.

  `-sourcepath ruta` Opción encargada de indicar al depurador la ruta donde se encuentran los ficheros de código fuente. Cada una de las rutas se indicarán separados por :.

- **`clase`** indica el nombre de la clase en la que se desea empezar la depuración.

- **`[argumentos]`** indica los argumentos que se desean pasar al método `main` de la clase indicada anteriormente.

A continuación se van a mostrar los comandos principales que se pueden utilizar dentro del depurador `jdb`:

- Orden **`help`**:

  La orden help se encarga de mostrar la lista de comandos válidos dentro del depurador.

- Orden **`run`** (**`run [class [args]]`**):

  La orden run se encarga de iniciar la ejecución de la aplicación Java que se está depurando, comenzando la ejecución en la clase `main` de la aplicación.

- Orden **`cont`**:

  La orden `cont` indica al depurador que se desea continuar la ejecución de la aplicación desde el breakpoint.

- Órdenes **`stop`** y **`clear`**:

  La orden `stop` nos permite fijar puntos de parada en nuestro programa. La sintaxis de esta orden es:

`stop at clase:linea`	Crea un punto de parada en la línea `linea` del fichero que contiene el código fuente de la clase `clase` (incluyendo el paquete al que pertenece).
`stop in metodo`	Crea un punto de parada en el método `metodo`. El método generalmente se indicará de la forma `clase.metodo` (incluyendo también el paquete al que pertenece).
`stop in clase.<init>`	Crea un punto de parada en el constructor de la clase `clase` (incluyendo el paquete al que pertenece).
`stop`	Muestra los puntos de parada establecidos hasta el momento.

  La orden `clear` nos permite eliminar puntos de parada fijados anteriormente.

- Órdenes **`catch`** e **`ignore`**:

  La orden `catch` nos permite indicar al depurador que este debe detener la ejecución del programa en el caso de que se lance la excepción que le indiquemos. La sintaxis de esta orden es `catch excepcion` (incluyendo el paquete al que pertenece la excepción `excepcion`). De forma opuesta, la orden ignore nos permite anular el efecto de una orden `catch` previa. Las excepciones en Java se verán en una práctica posterior.

- Orden **`step`**:

  La orden `step` nos permite ejecutar la línea de código en la que estamos detenidos, tras lo cual el control vuelve al depurador. En caso de que la línea sea una llamada a un método se ejecutarán todas las sentencias del método una a una mediante sucesivas llamadas a **`step`**.

- Orden **`stepi`**:

  Ejecuta sólo una instrucción.

- Orden **`next`**:

  La orden `next` nos permite ejecutar la siguiente línea de código en la función en la que estamos detenidos, tras lo cual el control vuelve al depurador. En caso de que la línea sea una llamada a un método se ejecutará con una sola ejecución del comando **`next`**.

- Orden **`print`**:

  A través de la orden `print` podemos obtener información sobre las variables de instancia de los objetos de nuestra aplicación. En el caso de utilizar esta orden para pedir información sobre un objeto, esta nos devolvería una breve descripción del objeto. Por último, esta orden soporta la mayoría de las expresiones que se pueden utilizar en Java.

  Por ejemplo:

```
print MiClase.miAtributoEstatico
print miObjeto.miAtributo
print i + j + k  (i, j, k son tipos primitivos y ambos son
                  atributos o variables locales)
print miObjeto.miMetodo() (si miMetodo devuelve
                           distinto de null)
print new java.lang.String("Hello").length()
```

- Orden **`dump`**:

  A través de la orden `dump` podemos obtener información de cada una de las variables de un objeto.
  En el caso de utilizar la orden `dump` con variables de instancia, el funcionamiento es igual que el de la orden `print`. Además, esta orden, al igual que la orden anterior, soporta la mayoría de las expresiones que se pueden utilizar en Java.
  Para objetos, imprime el valor actual de cada campo definido en el objeto. Incluidos los campos estáticos y las variables de instancia.
  El comando `dump` soporta las mismas expresiones que el comando `print`.

- Orden **`eval <expr>`**:

  Evalúa una expresión (igual que **`print`**).

- Orden **`set`** (Sintaxis **`set <lvalue> = <expr>`**):

  Asigna un nuevo valor al elemento campo/variable/array.

- Orden **`where`**:

  La orden `where` nos permite conocer el contenido de la pila del programa que estamos depurando.

- Orden **up** y **down** (**up [n frames]** y **down [n frames]**):

  Con el comando `up` y `down`, selecciona qué trama de la pila es la actual.

- Orden **list**:

  La orden **list** muestra por pantalla el código fuente que estamos depurando.

- Orden **quit**:

  La orden `quit` finaliza la ejecución del depurador.

- Orden **locals**:

  Imprime todas las variables en la trama actual de la pila.

- Orden **classes**:

  Muestra las clases actualmente conocidas

- Orden **class <class id>**:

  Muestra detalles de la clase nombrada.

- Orden **methods <class id>**:

  Lista los métodos de una clase.

- Orden **fields <class id>**:

  Lista los campos de una clase.

- Orden **!!**:

  Repite el ultimo comando.

- Orden **<n> <comando>**:

  Repite `<n>` veces el comando `<comando>`.

- Orden **read <fichero_de_comandos>**:

  Lee un fichero que contiene los comandos a ejecutar.

Ejercicios propuestos.

1. Descargue el código de la plataforma para realizar esta práctica. Escriba el código Java de la clase **`Usuario.java`** en el paquete **`fp2.poo.practica3`**, para que implemente los métodos de la interfaz **`Persona.java`**. Implemente al menos un constructor.

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

package fp2.poo.practica3;

/**
 * Descripcion: Esta una interfaz con los metodos de Persona.
 *
 * @version version 1.0 Mayo 2011
 * @author  Fundamentos de Programacion II
 */
public interface Persona {

    public String getNombre( );
    public String getPrimerApellido( );
    public String getSegundoApellido( );
    public String getDNI( );
    public String getDomicilio( );

    public void setNombre( String nombre );
    public void setPrimerApellido(  String primerApellido );
    public void setSegundoApellido( String segundoApellido );
    public void setDNI( String dni );
    public void setDomicilio( String domicilio );
}
```

2. Escriba el código Java de la clase **`Main.java`** (en el paquete **`fp2.poo.practica3`**) que contenga el método **`main`**, e instancie al menos un objeto de la clase **`Usuario`** mediante el constructor implementado. Obtenga el valor de los atributos y sáquelos por la salida estándar mediante el método **`System.out.println`**.

   ¿Se puede utilizar variables referencia de la interfaz **`Persona`** para acceder a los atributos y métodos de un objeto de la clase **`Usuario`**? Pruébelo.

3. Escriba el fichero **`makefile`** para realizar la compilación, y la creación del fichero **`Usuario.jar`**, de las clases implementadas y de la interfaz. Utilice la estructura de directorios de las prácticas anteriores (código en el directorio **`src`**, un directorio **`jar`** con el código empaquetado y el código compilado en **`bin`**).
   Nota: Para evitar ***`warning`*** en la compilación debido a los acentos de los comentarios compile con la opción **`-encoding ISO-8859-1`**.

4. Ejecute el depurador **`jdb`**, de la siguiente forma:

```
jdb -classpath ./jar/Usuario.jar -sourcepath ./src fp2.poo.practica3.Main
```

5. Ponga un punto de parada al comienzo del método **`main`** de la clase **`Main`** del paquete **`fp2.poo.practica3`**.

6. Añada otro punto de parada en el constructor de la clase **`Usuario`**.

7. Compruebe de que realmente tiene el punto de parada.

8. Ejecute el programa.

9. Vuelva a realizar la misma operación escribiendo en un fichero de texto los comandos para crear los puntos de parada, y ejecútelos con el comando **`read`**.

10. Haga un listado del código del programa en el punto en el que se ha parado.

11. Mire el contenido de la traza de la pila y las variables locales.

12. Continúe la ejecución hasta el constructor de la clase **`Usuario`**.

13. Ejecute línea a línea.

14. Mire el contenido de la traza de la pila y las variables locales.

15. Muévase en la trama de pila para obtener el contexto del método **`main`**. Muestre las variables locales del método.

16. Muévase en la trama de pila para obtener el contexto del constructor de **`Usuario`**. Muestre las variables locales del método.

17. Continúe la ejecución del programa.

18. Ejecute el `jdb` desde el `ddd` (Data Display Debugger) con la siguiente orden:

```
ddd -jdb -classpath ./jar/Usuario.jar -sourcepath ./src fp2.poo.practica3.Main &
```

19. En el código de la practica 2, en la clase `Cuenta`, cambie el atributo `nombre` de tipo `String` (`String nombre`) por el atributo `titular` de tipo `Usuario` (`Usuario titular`) implementado en esta práctica, y compílela mediante un `makefile`.

20. (parte opcional) Implemente el método `factorial_r` en la clase `Calculadora` de la practica 1 que calcule de forma recursiva el factorial de un número. Use el `jdb` para comprobar la trama de la pila en cada momento de la ejecución del programa.