DATOS DE ENTRADA

Lenguaje java

Julio, 2014
Guadalajara, Jalisco. México

Realizado por
Prof. Sabrina Lizbeth Vega Maldonado

Existen dos formas de pedir datos al usuario desde el teclado:

- En un ambiente de texto

- En un ambiente gráfico

En el lenguaje de java no es muy común utilizar el ambiente de texto porque está diseñado para trabajar en un ambiente gráfico.

Se estudiarán las dos formas iniciando con el ambiente de texto a fin de visualizar más claramente los conceptos de la metodología orientada a objetos en el código.

AMBIENTE DE TEXTO

Pedir datos al usuario desde consola

Se puede pedir datos al usuario desde consola en dos formas:

- usando clases que ya existen en java adaptándonos al código establecido
- creando nuestra propia clase programando necesidades específicas

Estudiaremos estas dos formas

- Pero antes retomemos el ejemplo de la actividad pasada

Hasta aquí tenemos el diseño del esquema preliminar de clases que nos ayudará a dar solución a nuestro problema.

El siguiente paso es seguir el diseño para escribir el código en un lenguaje de programación orientada a objetos como es el lenguaje de Java con la sintaxis que ya estudiaste.

Código en java

```
public class Mensaje {
        private String frase;
        public void imprime ( ) {
                frase = “Este es mi mensaje”;
                System.out.println ( “La frase del mensaje es: ” + frase ) ;
        }
}
```

```
public class FuncionPrincipal {

        public static void main (String arg[]){

                Mensaje miMensaje = new Mensaje ( );

                miMensaje.imprime ( );

}
```

En el lenguaje java desde que se define la clase existe el constructor que es una función especial que no tiene tipo de dato y se llama igual que la clase, como su nombre lo indica construye o crea al objeto y realiza las siguientes actividades:

- Se activa una “sección automática” que aparta memoria RAM para el objeto, crea una tabla de símbolos y crea un formato lógico para esa tabla de símbolos; de esta manera sabe que direcciones en memoria aunque estén separados corresponden a un objeto

- Se activa una “sección usuario” que es la parte donde el programador como usuario del lenguaje puede decirle al constructor que haga alguna instrucción después de ejecutar la sección automática

Para este caso no es necesario programar nada (por lo tanto no se definió en la clase Mensaje), sólo se manda llamar donde se quiera crear al objeto que en nuestro ejemplo será dentro de la función principal por lo que el código queda como se describe arriba en la clase FuncionPrincipal

La llamada del constructor la puedes observar con negritas en el código

NOTA IMPORTANTE:

Observa que en la clase Mensaje, sólo se encuentra encapsulado de forma abstracta el objeto, esta clase no tienen nada más. Esta clase sólo nos sirve para encapsular al objeto que se va a programar en una clase. Si escribimos este código en un archivo llamado Mensaje.java y lo tratamos de ejecutar nos va a decir que no puede porque no encuentre la función principal, por lo tanto este archivo solamente lo compilamos para obtener un archivo Mensaje.class

Para la clase FuncionPrincipal, observa que no se están encapsulando objetos, ¿cómo se sabe esto? porque no tiene ni atributos, ni métodos. Esto significa que esta clase sólo nos servirá para ejecutar el programa con la función principal. No significa que esta clase no puede tener objetos, más bien significa que no le hemos programado objetos.

Está segunda clase la tenemos que programar en un archivo llamado FuncionPrincipal.java, este archivo sí se tendrá que compilar y luego ejecutar.

Pero hasta aquí, estamos escribiendo el dato de entrada dentro del código, entonces ¿cómo se lo pedimos al usuario? Lo veremos a continuación.

CLASE SCANER

- La clase que ya está creada en java para pedir datos al usuario es Scanner y se encuentra en el paquete java.util. Permite leer datos desde el teclado asociandolos a la clase System.in

- Para utilizar esta clase es necesario crear un objeto (instancia de referencia) de la clase Scanner y así acceder a sus métodos. Con esta instancia se llamará al comportamiento en donde se recibe un valor desde el teclado

Planteamiento:

Imprime en pantalla una tabla de multiplicar, el programa recibirá los datos de qué tabla se va a multiplicar y de el limite de cuantas veces se va a multiplicar

Objetos que intervienen:
- Pantalla
- Tabla

Objetos a programar:
- Tabla

Tabla

- numTabla: entero
- limite: entero

+ multTabla : void

Archivo: Tabla.java

```
import java.util.Scanner;

class Tabla {
    private int numTabla;
    private int limite;
    public void multTabla( ) {
        for (int i=1; i<=limite; i++)
            System.out.println(numTabla + “x” + i + “=” + (numTabla*i));
    }
    public Tabla(int num, int limite) {
        numTabla = num;
        this.limite = limite;
    }
}
```

Archivo: Principal.java

```
class Principal {
    public static void main(String ar [] ) {
        Scanner leeDato = new Scanner(System.in);
        System.out.print (“Que tabla quieres multiplicar: ”);
        int valorTabla = leeDato.nextInt( );
        System.out.print (“Cuantas veces?: “);
        int limite = leeDato.nextInt( );
        Tabla miTabla = new Tabla(valorTabla, limite);
        miTabla.multTabla( );
    }
}
```

- # ¿Qué es el código que está en color? - Definición y llamada del Constructor

¿Recuerdas que es el constructor?, es una función especial que no tiene tipo de dato, se llama igual que el nombre de la clase y que nos ayuda a crear o construir al objeto.

Bueno en este ejemplo, sí es necesario definir al constructor en la clase Tabla (código en azul), porque a parte de que nos ayude a construir al objeto necesitamos pasarle los valores que serán nuestros datos de entrada y que se estarán recibiendo desde consola.

Si te fijas estos datos de entrada se están leyendo en la función principal, entonces es necesario enviar esos datos a los atributos y el constructor nos ayuda a programar esto. Es por eso que en el código en azul que es la definición del constructor tiene dos parámetros.

Ahora fijare que en la llamada del constructor dentro de la función principal, (código en verde) se hace la llamada con dos argumentos que son las variables que contienen los valores que queremos enviar al constructor.

Por último. Cuando el parámetro tiene el mismo identificador que el atributo es necesario indicar al compilador quién es el que forma parte del objeto. Por supuesto es el atributo el que forma parte del objeto por lo tanto al atributo se le asigna la instrucción “this” que podríamos traducirlo como “este es”

Si observas el código que está en azul puedes encontrar que el valor del parámetro “limite” se le está asignando al atributo “limite” - this.limite = limite;

Cuando tiene diferentes identificadores no hay problema; por ejemplo, el valor del parámetro num se le asigna al atributo numTabla - numTabla = num;

CÓMO HACER MÁS FUNCIONAL EL CÓDIGO

Ahora bien, estamos pidiendo los datos de entrada desde consola en la función principal y como la función principal no ve a los atributos (recuerda que están encapsulados en clases diferentes y tienen un ocultamiento de información para no romper con el encapsulamiento); pues tenemos que pasar esos valores a través de parámetros mediante el constructor.

La forma de hacer más práctico y funcional esto. Es re-diseñar la solución del problema y agregar al objeto el comportamiento de leer los valores además del comportamiento de imprimir en pantalla la tabla de multiplicar.

A continuación tenemos cómo quedaría el esquema preliminar de clases con este nuevo diseño para la solución de nuestro problema.

Tabla
- numTabla: entero - limite: entero
+ leeNums : void + multTabla : void

Archivo: Tabla.java

```
import java.util.Scanner;

class Tabla {
    private int numTabla;
    private int limite;
    public void leeNums( ) {
        Scanner leeDato = new Scanner(System.in);
        System.out.print ("Que tabla quieres multiplicar: ");
        numTabla = leeDato.nextInt( );
        System.out.print ("Cuantas veces?: ");
        limite = leeDato.nextInt( );
    }
    public void multTabla( ) {
        for (int i=1; i<=limite; i++)
            System.out.println(numTabla + "x" + i + "=" + (numTabla*i));
    }

}
```

Archivo: Principal.java

```
class Principal {
    public static void main(String ar [] ) {
        Tabla miTabla = new Tabla( );
        miTabla.leeNums( );
        miTabla.multTabla( );
    }
}
```

MANEJO DE EXCEPCIONES

Al recibir un valor desde el teclado puede provocar que el programa rompa su ejecución por un error de entrada de datos mediante el buffer del teclado, entre otros errores.

Este error en tiempo de ejecución se le conoce como que se genera una excepción. Por lo que es necesario manejar las excepciones, a través de una clase que existe en java llamada Exception

Esta clase nos facilita el manejo de estos errores. En el API de java que se encuentra en la dirección http://docs.oracle.com/javase/1.5.0/docs/api/index.html podrás buscar la clase Exception y encontrarás la referencia que necesitas para conocer todos los tipos de excepciones que están creadas en java.

Para el manejo de excepciones se usan las instrucciones “try” - “catch”.

La instrucción “try” encierra el código donde es posible que exista una excepción (se puede utilizar en cualquier parte del código)

La instrucción “catch” es una función especial en donde se recibe la excepción y permite que continúe la ejecución del programa. Puede quedar vacío y que no haga nada o bien se puede programar que se imprima un mensaje o que haga algún proceso

No se te olvide hacer pruebas al código, es la única manera de saber si existen Excepciones

Sintaxis:

```
try {
        /* codigo con una
           posible excepción */
} catch ( <TipoExcepcion> <identificadorParametro> )
{
        /* codigo que realizará cuando
           se salte la parte que generá la excepción
           indicada en el TipoExcepción del catch*/
}
```

Revisemos nuestro ejemplo de tabla

Cuando ejecutas este programa (recuerda el archivo que vas a ejecutar será Principal.java) en consola te aparece lo siguiente:

```
Que tabla quieres multiplicar: _
```

Si escribes un valor entero no habrá problema, pero como programador no tenemos control sobre lo que el usuario final va a escribir, entonces si se escribe una letra u otro valor que no sea entero entonces el compilador intenta guardar en un formato que no corresponde y surge una excepción (error en tiempo de ejecución).

¿Entonces cómo controlamos esto?

Una vez que hacemos pruebas al ejecutar nuestro programa si existe el error nos marcará la excepción correspondiente y si te fijas dice en qué línea de qué archivo se produce y que tipo de excepción es.

```
Que tabla quieres multiplicar: no se

Exception in thread "main" java.lang.NumberFormatException: For input string: “no se"
     at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
     at Tabla.leeNums(Tabla.java:9)
```

Para este caso se produce la excepción NumberFormatException en la línea 9 del archivo Tabla.java

Entonces los cambios a realizar en el código serán los siguientes:

Archivo: Tabla.java

```
import java.util.Scanner;

class Tabla {
    private int numTabla;
    private int limite;
    public void leeNums( ) {
        Scanner leeDato = new Scanner(System.in);
        try
        {
            System.out.print (“Que tabla quieres multiplicar: ”);
            numTabla = leeDato.nextInt( );
            System.out.print (“Cuantas veces?: “);
            limite = leeDato.nextInt( );
        } catch ( NumberFormatException e )
            {
                System.out.println (“Error: escribiste un valor incorrecto “);
            }
    }
    public void multTabla( ) {
        for (int i=1; i<=limite; i++)
            System.out.println(numTabla + “x” + i + “=” + (numTabla*i));
    }

}
```

NUESTRA PROPIA CLASE PARA LEER DATOS

- Como ya se menciono también podemos crear nuestra propia clase para pedir datos de entrada desde el teclado en un ambiente de texto desde consola. Para ello necesitamos conocer las siguientes clases: InputStreamReder y BufferedReader (puedes consultar el API de java)

- InputStreamReader, nos permite leer bytes y los descodifica en caracteres (desde el teclado)

- BufferedReader, lee una cadena de texto que se encuentre en un buffer (espacio de memoria donde se almacenan datos)

- Para mayor eficiencia se pueden contener una instancia de InputStreamReader dentro de una instancia de BufferedReader, de esta manera se leerá una cadena de texto con los bytes descodificados que se reciban desde el teclado, como se describe a continuación:

- BufferedReader leeDato = new BufferedReader(new InputStreamReader(System.in));
  String miVariable = leeDato.readLine( );

CREANDO UNA PARA LEER DATOS

Se necesitan dentro de la clase funciones estáticas que regresen el valor recibido que se haya escrito en el teclado

Archivo: DatosEntrada.java

```
import java.io.*;
class DatosEntrada {
	public static String cadena( ) {
		try{	BufferedReader leeDato=new BufferedReader(new InputStreamReader(System.in));
			return leeDato.readLine();
		} catch(IOException e) {
			return " ";
		 }
	}
	public static int entero( ) {
		try{	return Integer.parseInt(cad());
		} catch(NumberFormatException e) {
			return -1;
		 }
	}
	public static double real( ){
		try{ return Double.parseDouble(cad());
		} catch(NumberFormatException e) {
			return -1.0;
		 }
	}
	public static float flotante(){
		try { return Float.parseFloat(cad());
		} catch (NumberFormatException e) {
			return -1.0f;
		 }
	}
	public static char caracter(){
		try{ return cad().charAt(0);
		} catch(IndexOutOfBoundsException e) {
			return ' ';
		 }
	}
}
```

Una vez que se compila, se genera el archivo DatosEntrada.class

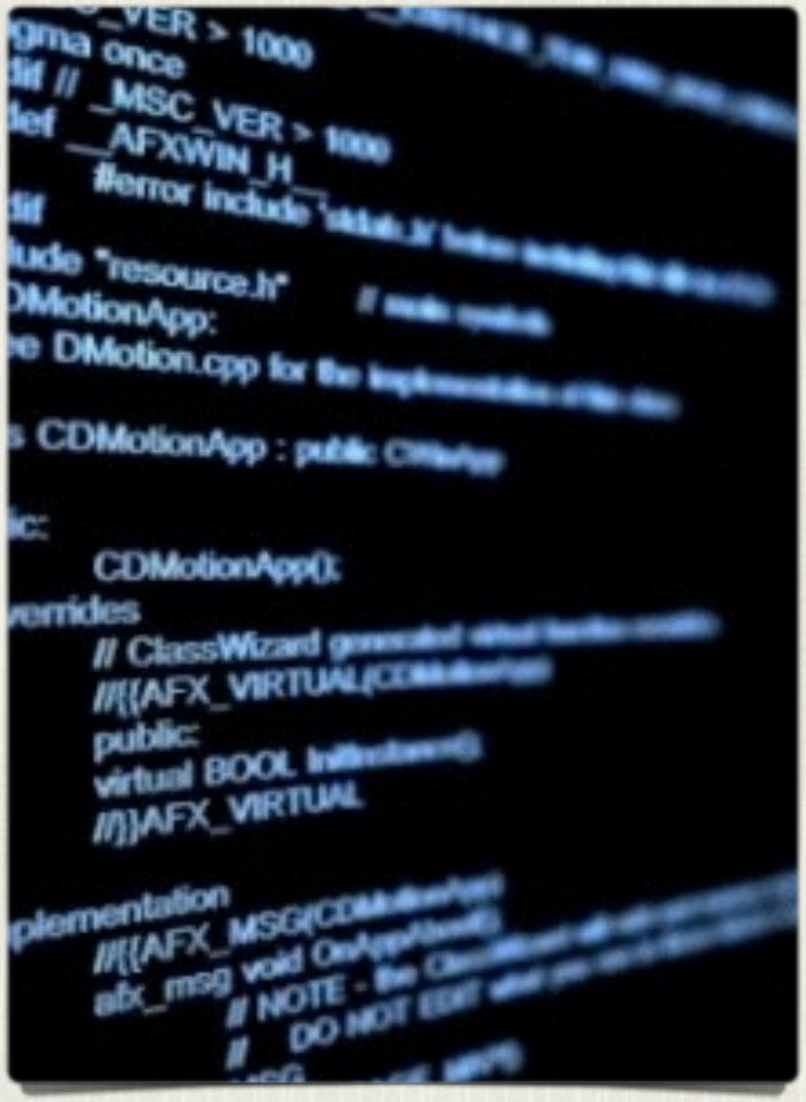
Entonces ya se puede hacer uso de este programa con sólo llamar a sus funciones estáticas. Modifiquemos nuestro ejemplo de Tabla para ver cómo se implementa:

Archivo: Tabla.java

```
class Tabla {
    private int numTabla;
    private int limite;
    public void leeNums( ) {
            System.out.print ("Que tabla quieres multiplicar: ");
            numTabla = DatosEntrada.entero( );
            System.out.print ("Cuantas veces?: ");
            limite = DatosEntrada.entero( );
    }
    public void multTabla( ) {
      if ( numTabla == -1 || limite == -1 )
                System.out.println( "Error: el valor escrito es incorrecto");
       else {
            for (int i=1; i<=limite; i++)
                System.out.println(numTabla + "x" + i + "=" + (numTabla*i));
          }
    }

}
```

Para mandar llamar una función estática no es necesario crear una instancia. La llamada se hace con el nombre de la clase donde se encuentra, como puedes observar con negritas en el código de arriba. Puedes agregar al archivo DatosEntrada.java más funciones estáticas



Este archivo puedes compilarlo y guardarlo en la misma carpeta donde se encuentre el programa que estés creando y utilizarlo las veces que sean necesarias mandando llamar sus funciones únicamente