Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC1103 Introducción a la Programación (II/2011)

Librería del curso

Gabriel Vidal
gpvidal@uc.cl

1. Introducción

La librería (*package*) del curso es una colección de herramientas que nos permitirá agilizar ciertas operaciones recurrentes que haremos en nuestros programas. Es importante conocer qué es lo que compone a este paquete y cuáles son las funciones que cumplen cada uno de los componentes de esta herramienta.

2. Obtener el paquete del curso

Para obtener el paquete basta con dirigirse al sitio web del curso y descargar el archivo **iic1103.jar**, el cual estará disponible en las carpetas de recursos del sitio.

3. Utilización del paquete del curso

Para utilizar el paquete del curso lo primero que hay que hacer es llevarlo a nuestro proyecto en Eclipse. Para esto se debe arrastrar el archivo `iic1103.jar` a la carpeta del proyecto y debe quedar como la figura 1. Posteriormente sobre el archivo hacemos clic secundario y seleccionamos *Build Path/Add to Build Path*. Con este paso lograremos agregar las herramientas (librería) a nuestro proyecto para que puedan ser ocupadas.

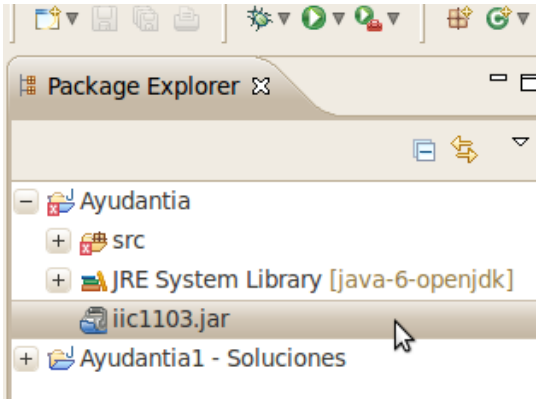

Figura 1: Importando el paquete del curso

Como último paso, cada vez que necesitemos ocupar el paquete del curso, debemos incluir una cabecera al inicio de nuestro archivo:

```
import iic1103.*;
```

Y con esto ya podemos utilizar las clases que vienen pre-fabricadas.

4. Herramientas del Paquete

Luego que ya tenemos configurada la librería y lista para usar, procederemos a explicar las distintas clases que componen este paquete, las cuáles son: **InputOutput**, que nos facilitará la recolección y el despliegue de datos al usuario de nuestro programa, y **GeneradorAleatorio**, que nos permitirá crear distintas situaciones (números).

4.1. Clase InputOutput

Esta clase nos permitirá, tal como lo explicábamos antes, recibir y mostrar información al usuario. Para esto existen métodos pre-definidos que nos permiten realizar estas acciones.

Es necesario aclarar que para poder hacer uso de este elemento es necesario crear un objeto de la clase `InputOutput` de la siguiente forma:

```
InputOutput <nombre> = new InputOutput();
```

Por ejemplo:

```
InputOutput io = new InputOutput();
```

En cada parte de este documento en donde aparezca una llamada a `io`, supondremos que un objeto de la clase `InputOutput` esta referenciado por esta variable.

4.1.1. Métodos de Entrada de datos

- **PideUnEntero**

Este método nos permitirá rescatar del usuario alguna preferencia que responde a un tipo de dato entero. La sintaxis de utilización es la siguiente:

```
io.pideUnEntero("Ingrese un numero: ");
```

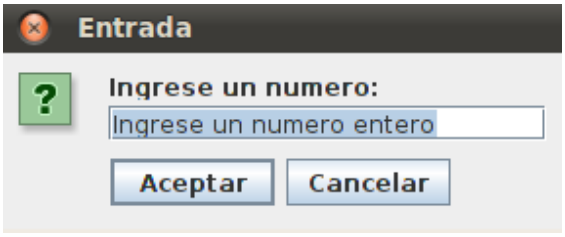

Figura 2: Usando el método pideUnEntero de InputOutput

Esto nos mostrará una ventana como se ve en la Figura 2. Este método nos retornará el dato ingresado por el usuario y que corresponde al tipo `int`.

Normalmente, querremos guardar lo que nos entregue como respuesta el usuario, por lo que el resultado de este método lo almacenaremos en una variable:

```
int eleccion = io.pideUnEntero("Ingrese un numero: ");
```

Además podemos cambiar el texto que aparece en donde el usuario ingresa el número de la siguiente manera:

```
io.pideUnEntero("Ingrese un numero : ", "Ingresa aqui");
```

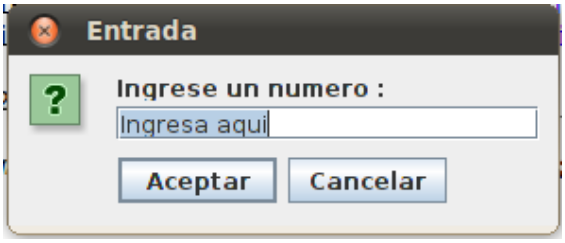

Figura 3: Usando el método pideUnEnterio de InputOutput cambiando el texto de ingreso

Y obtendremos el resultado que muestra la Figura 3. Esto también nos servirá para poder colocar un valor por defecto en la entrada de datos.

En el caso de que lo ingresado por el usuario no se pueda interpretar como un número entero válido, entonces se desplegará un mensaje de error y el método entegará -1 como resultado.

- **PideUnReal**

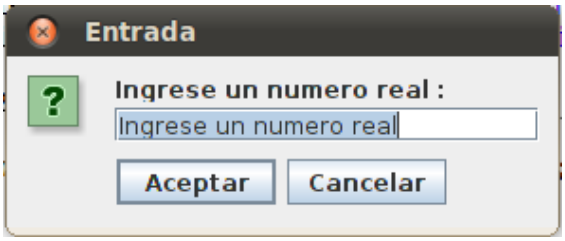

Figura 4: Usando el método pideUnReal de InputOutput

Este otro método permite que el usuario ingrese datos del tipo `double` como 1.5, 3.1415, etc. Para poder llevar a cabo esto debemos utilizar la siguiente sintaxis:

```
io.pideUnReal("Ingrese un numero real");
```

Con esto nos aparecerá una ventana similar al método *pideUnEntero* con la diferencia que podemos ingresar datos numéricos reales. La Figura 4 muestra como es la ventana que se despliega.

También podemos almacenar lo ingresado por el usuario de la siguiente forma:

```
double real = io.pideUnReal("Ingrese un numero real");
```

Al igual que en el método *pideUnEntero* podemos cambiar el texto que aparece en donde el usuario ingresa el valor de la siguiente forma:

```
io.pideUnReal("Ingrese un numero real", "Ingresa aqui");
```

Y se obtendrá una ventana similar a la que aparece en la Figura 3, pero ahora para el caso de ingresos de reales.

Si lo ingresado por el usuario no puede interpretarse como un número real, entonces se mostrará una pantalla de error y el método retornará -1.

- **PideUnTexto**

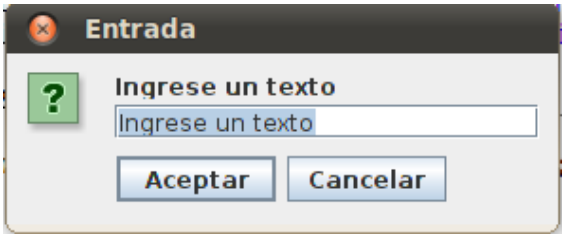

Figura 5: Usando el método pideUnTexto de InputOutput

El último método para el ingreso de datos por parte del usuario, nos permite que se ingresen cadenas de texto como `Esto es una cadena de texto` para poder manipularlas. Para consultar al usuario este tipo de datos (`String`) debemos realizar:

```
io.pideUnTexto("Ingrese un texto")
```

Y con esto obtendremos una ventana que nos permita consultar al usuario por este tipo de datos. La Figura 5 muestra el ejemplo de la ventana desplegada.

Podemos almacenar lo ingresado por el usuario en una variable de la siguiente forma:

```
String texto = io.pideUnTexto("Ingrese un texto")
```

Al igual que los métodos anteriores podemos cambiar el texto que aparece en el lugar donde el usuario ingresa los datos de una forma similar a las anteriores:

```
io.pideUnTexto("Ingrese un texto", "Ingrese aqui");
```

Y se obtiene una ventana semejante a la de los casos anteriores.

4.1.2. Métodos de Salida de datos

- **Muestra**

Figura 6: Usando el método muestra de InputOutput

Nos permitiá desplegar información al usuario. Para poder ocuparlo debemos escribir:

```
io.muestra("Este es un mensaje para el usuario");
```

y esto desplegará una ventana como se puede ver en la Figura 6.

Este método puede también desblegar distintos tipos de datos como `int`, `double`, `char` y `String`. Para esto sólo debemos colocar el nombre de la variable dentro de los paréntesis de la sentencia, como ejemplo:

```
int valor = 10;
io.muestra(valor);
```

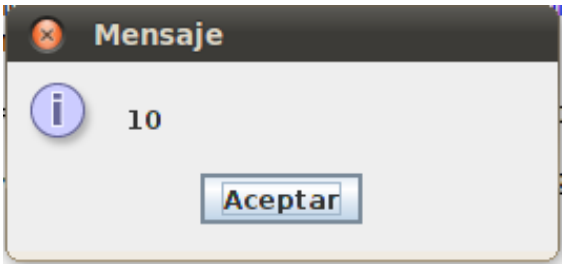

Figura 7: Ejemplo de método muestra con variables

Y el resultado quedará como lo muestra la Figura 7.

- **muestraEnConsola**

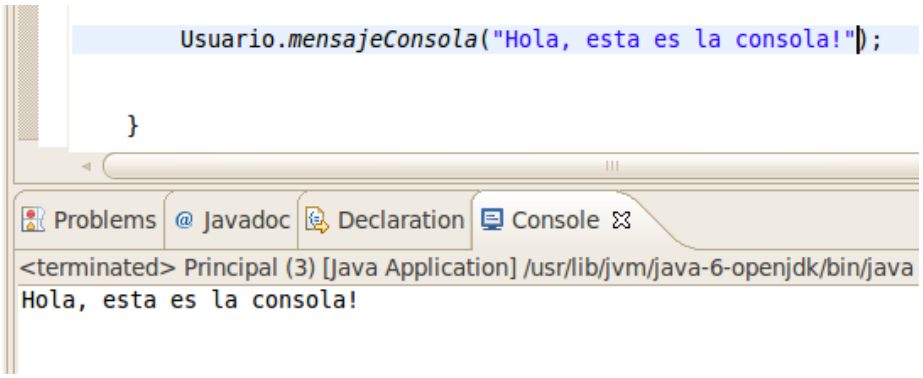

Figura 8: Ejemplo de método muestraEnConsola

Tiene la misma funcionalidad de la acción *muestra* que revisamos anteriormente, sólo que en este caso la salida de los datos ocurrirá por la consola. Para poder ocupar esta herramienta debemos utilizar la siguiente sintaxis:

```
io.muestraEnConsola("Hola, esta es la consola!");
```

Lo que provocará que la salida de los datos sea en consola tal como lo muestra la Figura 8.

Al igual que el método *muestra*, también puede recibir variables (del mismo tipo mencionadas anteriormente) y las desplegará en la consola.

4.2. Clase GeneradorAleatorio

Esta clase nos permitirá generar números de forma aleatoria, ya sea de tipo entero (`int`) o de tipo real (`double`). Esto puede ser muy útil cuando queremos generar distintos escenarios de una forma rápida para poder probar la ejecución de un programa.

Al igual que en el caso de `InputOutput`, para poder hacer uso de este elemento es necesario crear un objeto de la clase `GeneradorAleatorio` de la siguiente forma:

```
GeneradorAleatorio <nombre> = new GeneradorAleatorio();
```

Por ejemplo:

```
GeneradorAleatorio aleatorio = new GeneradorAleatorio();
```

En cada parte de este documento en donde aparezca una llamada a `aleatorio`, supondremos que un objeto de la clase `GeneradorAleatorio` esta referenciado por esta variable.

4.2.1. Generar números aleatorios

Para poder generar números aleatorios, esta herramienta cuenta con 2 métodos: `entero` y `real`. Cada uno de ellos genera los tipos de números a los que hacen referencia y la sintaxis para poder utilizar estos métodos es la siguiente:

```
aleatorio.entero(1, 5);
```

Esto indica que se va a generar un número entero que va entre 1 y 5 (ambos inclusive)

Ahora para el caso de los reales:

```
aleatorio.real(3.5, 6);
```

Indica que se va a generar un número real comprendido entre el valor 3.5 (inclusive) y 6 (exclusive).

Como ejemplo ejecutaremos el siguiente código:

```
int valor1 = aleatorio.entero(1, 10);
double valor2 = aleatorio.real(3.5, 6);
io.muestraEnConsola(valor1);
io.muestraEnConsola(valor2);
```

En una primera ejecución obtuvimos los valores

```
9
4.626924862754875
```

Y en una segunda ejecución (sin cambiar el código)

```
6
5.805083099573943
```

Ésta es la forma en cómo opera esta herramienta.