IIP (E.T.S. de Ingeniería Informática)
Curso 2016-2017

Práctica 4. Desarrollo y reutilización de clases Java

Profesores de IIP
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Índice

1. Objetivos y trabajo previo a la sesión de prácticas

El objetivo principal de esta práctica es el diseño de una clase *"Tipo de Dato"* incluida en un paquete. Se trabajarán varios de los aspectos presentados en los temas 3 y 4 (*Variables: definición, tipos y uso en Java* y *Métodos: definición, tipos y uso en Java*, respectivamente), más la creación de paquetes para agrupar clases. En concreto, se incidirá en:

- Declaración de la estructura de los objetos de una clase.
- Declaración de los constructores de la clase, y de los métodos consultores, modificadores y de otros propósitos.
- Uso de una clase tipo de datos: declaración de variables referencia, creación de objetos y aplicación de sus métodos.
- Creación de un paquete para tener organizadas las clases que se desarrollen en todo proyecto software.

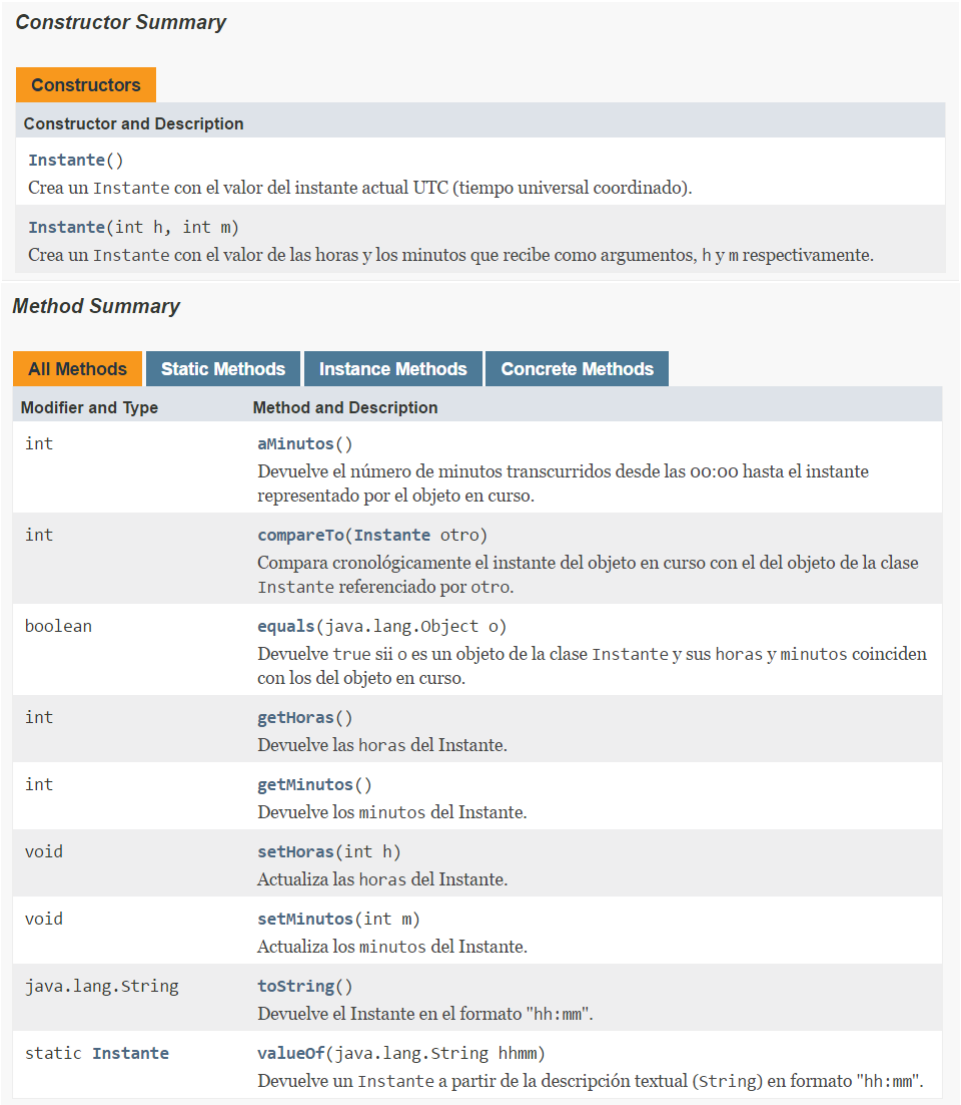
2. Descripción del problema

En esta práctica se va a desarrollar la clase `Instante` para que tenga una funcionalidad como la que se indica en la Figura 1. Para el desarrollo de sus métodos serán de gran utilidad las operaciones trabajadas en la práctica anterior.

Una vez desarrollada esta clase `Instante`, se escribirá una Clase Programa `Practica4` que ejecute una serie de instrucciones similar a la de la práctica anterior, `Practica3`, pero utilizando los objetos y métodos de `Instante`.

El nombre de la clase, `Instante`, refleja lo que se quiere representar: una marca de tiempo (`Timestamp`). Así, la clase representa el momento o instante que define una hora, en este caso, las horas y los minutos de la misma.

Aunque es bueno saber que un `Timestamp` o marca de tiempo incluye mucho más detalle (año, mes, día, horas, minutos, segundos y milisegundos o microsegundos, según la implementación),



Constructor Summary

Constructors

Constructor and Description
`Instante()` Crea un `Instante` con el valor del instante actual UTC (tiempo universal coordinado).
`Instante(int h, int m)` Crea un `Instante` con el valor de las horas y los minutos que recibe como argumentos, `h` y `m` respectivamente.

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`int`	`aMinutos()` Devuelve el número de minutos transcurridos desde las 00:00 hasta el instante representado por el objeto en curso.
`int`	`compareTo(Instante otro)` Compara cronológicamente el instante del objeto en curso con el del objeto de la clase `Instante` referenciado por `otro`.
`boolean`	`equals(java.lang.Object o)` Devuelve `true` sii `o` es un objeto de la clase `Instante` y sus `horas` y `minutos` coinciden con los del objeto en curso.
`int`	`getHoras()` Devuelve las `horas` del Instante.
`int`	`getMinutos()` Devuelve los `minutos` del Instante.
`void`	`setHoras(int h)` Actualiza las `horas` del Instante.
`void`	`setMinutos(int m)` Actualiza los `minutos` del Instante.
`java.lang.String`	`toString()` Devuelve el Instante en el formato "`hh:mm`".
`static Instante`	`valueOf(java.lang.String hhmm)` Devuelve un `Instante` a partir de la descripción textual (`String`) en formato "`hh:mm`".

Figura 1: Application Programming Interface (API) de la clase `Instante`

en esta práctica, para simplificar, la clase `Instante` únicamente manejará dos atributos, `horas` y `minutos`.

Nótese que el término `Timestamp` es el que se usa en las bases de datos como un tipo de dato específico para manejar datos que son marcas de tiempo.

3. Actividades de laboratorio

Actividad 1: Creación del proyecto BlueJ `pract4` y del paquete `iipUtil`

a) Descargar en `Downloads` los ficheros `Instante.java` y `Pract4TestUnit.class`, disponibles en la carpeta de material para la práctica de *PoliformaT*.

El fichero `Instante.java` contiene el esqueleto de la clase a desarrollar, incluyendo los comentarios que deben preceder a cada uno de los métodos. El fichero `Pract4TestUnit.class` sirve para comprobar que se ha realizado una implementación correcta de la clase `Instante` (veáse la Actividad 6).

b) Abrir *BlueJ* en el directorio de trabajo de la asignatura (`iip`) y crear el nuevo proyecto `pract4`.

c) Crear un paquete de nombre `iipUtil` y abrirlo mediante doble clic.

d) Agregar al paquete `iipUtil` la clase `Instante.java`. Comprobar que la primera línea incluye la directiva del compilador para indicar que es una clase perteneciente a un paquete:

```
package iipUtil;
```

e) Copiar en la carpeta `pract4` la clase `Pract4TestUnit.class`.

En el vídeo pract4-create-package.ogv se puede ver la realización de los pasos anteriores.

Actividad 2: Desarrollo y prueba de la clase `Instante`. Atributos y métodos constructores

Como se ha comentado anteriormente, cada objeto de tipo `Instante` mantiene la información de las horas y los minutos que definen una marca o instante de tiempo. Así pues, los atributos serán:

```
private int horas;
private int minutos;
```

En la clase se incluirá un primer método constructor con la cabecera o perfil:

```
/**
 *  Crea un <code>Instante</code> con el valor de
 *  las horas y los minutos que recibe como argumentos,
 *  <code>h</code> y <code>m</code>, respectivamente.
 *
 *  <p> Debe cumplirse la precondición:
 *      <code>0 <= h < 24, 0 <= m < 60</code>. </p>
 */
public Instante(int h, int m)
```

Fíjese el lector como en los comentarios se incluyen algunos *tags* de `HTML` para que se muestre mejor en el navegador. Por ejemplo, `<code>` o `<p>`.

También se debe escribir el constructor por defecto que cree objetos de tipo `Instante` con los valores de horas y minutos correspondientes al instante actual según UTC. Es decir, este método deberá encapsular los cálculos que se realizaban en la práctica anterior para calcular horas y minutos de la hora UTC.

```
/**
 * Crea un <code>Instante</code> con el valor del instante actual
 * UTC (tiempo coordinado universal).
 */
public Instante()
```

Una vez editada y compilada esta parte de la clase, utilizando el *Object Bench*, se probará la creación de objetos y se verificará que son correctos, como en el ejemplo de la Figura 2.

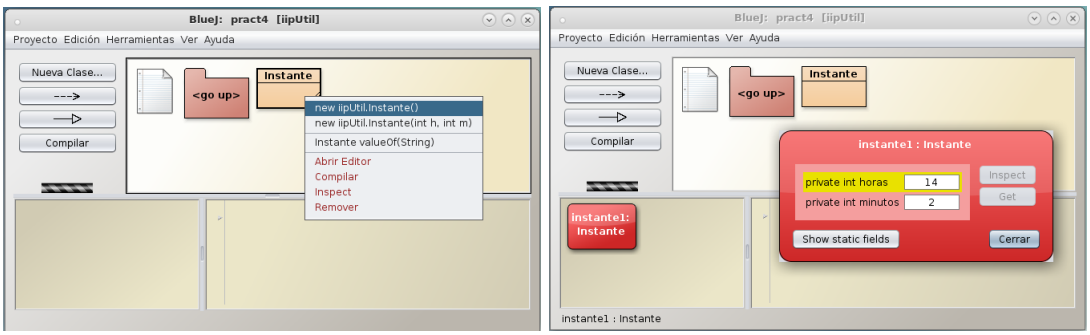


Figura 2: Ejemplo de como crear objetos de tipo `Instante` e inspeccionarlos

Actividad 3: Desarrollo y prueba de la clase `Instante`. Métodos consultores y modificadores

Añadir a la clase los métodos consultores y modificadores con las siguientes cabeceras:

```
/** Devuelve las horas del Instante. */
public int getHoras()

/** Devuelve los minutos del Instante. */
public int getMinutos()

/** Actualiza las horas del Instante. */
public void setHoras(int h)

/** Actualiza los minutos del Instante. */
public void setMinutos(int m)
```

Antes de seguir añadiendo más métodos en la siguiente actividad, se recompilará la clase y se probará que los métodos ya escritos son correctos. Para ello se crearán objetos, bien en el *Object bench* bien en el *Code Pad* de *BlueJ*, y se les aplicarán los métodos verificando su resultado.

Actividad 4: Desarrollo y prueba de la clase `Instante`. Métodos `toString`, `equals`, `aMinutos` y `compareTo`

Añadir a la clase los métodos cuyas cabeceras se muestran a continuación:

```
/** Devuelve el Instante en el formato "hh:mm". */
public String toString()

/** Devuelve <code>true</code> sii <code>o</code> es un objeto de
 *  la classe <code>Instante</code> y sus horas y minutos coinciden
 *  con los del objeto en curso.
 */
public boolean equals(Object o)

/** Devuelve el número de minutos transcurridos desde las 00:00 hasta
 *  el instante representado por el objeto en curso.
 */
public int aMinutos()

/** Compara cronológicamente el instante almacenada por el objeto en curso
 *  con el almacenado en el objeto de la clase <code>Instante</code>
 *  referenciado por <code>otro</code>.
 *  <p>
 *  El resultado es un valor:
 *  <ul>
 *     <li> negativo si el instante del objeto en curso es anterior
 *          al de <code>otro</code>, </li>
 *     <li> cero si son iguales, </li>
 *     <li> positivo si el instante del objeto en curso es posterior
 *          al de <code>otro</code>. </li>
 *  </ul>
 *  </p>
 */
public int compareTo(Instante otro)
```

A la hora de implementar el método `equals` se recuerda que, debido al operador cortocircuitado `&&`, es importante el orden de los operandos en la expresión que compara si `o` es un objeto de la clase `Instante` y, en caso afirmativo, si sus atributos coinciden en valor con los del objeto en curso:

```
o instanceof Instante
&&  this.horas == ((Instante) o).horas
&&  this.minutos == ((Instante) o).minutos
```

De esta manera, sólo se evalúan el segundo y tercer operandos de la expresión si `o` es efectivamente un `Instante` y, por tanto, se le puede aplicar el *casting* que permite a Java tratar `o` como un objeto de la clase `Instante` y acceder a sus atributos.

Para comprobar el comportamiento de `instanceof` se pueden hacer pruebas en el *Code Pad*, como las de la Figura 3.

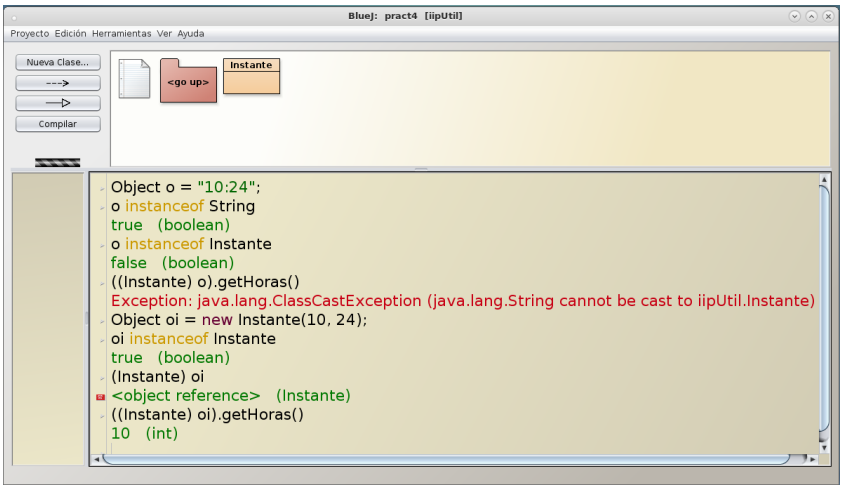


Figura 3: Ejemplo de como probar el comportamiento de `instanceof` mediante *Code Pad*

La clase se deberá recompilar y probar los métodos añadidos. Por ejemplo, para probar `equals` y `compareTo` se pueden crear tres instantes `instante1`, `instante2` e `instante3`, correspondientes a las 00:00, 12:10, 12:10 respectivamente, y comprobar que:

- `instante2` y `instante3` son iguales,
- `instante1` es anterior a `instante2`,
- `instante2` es posterior a `instante1`.

Actividad 5: Comprobación de las normas de estilo de la clase `Instante`

Comprobar que el código de la clase escrita cumple las normas de estilo usando el `Checkstyle` de *BlueJ*, y corregirlo si es el caso.

Actividad 6: Comprobación del funcionamiento de la clase `Instante`

Para comprobar el correcto funcionamiento de los métodos de la clase `Instante`, se ha proporcionado la clase `Pract4TestUnit`. Esta clase es una `TestUnit` que se ejecuta como se puede ver en la figura 4. [1] Si los métodos son correctos, en la ventana *Test Results* de BlueJ, aparecerán marcados con un ✓ (de color verde). Si, por el contrario, algún método no funciona correctamente en la ventana *Test Results*, el test de cada método incorrecto aparecerá marcado con una `X`. Si se selecciona uno de dichos tests, en la parte inferior de la ventana, se muestra un mensaje orientativo sobre la causa del error.

En la Figura 5 puede verse cómo debe quedar la carpeta o directorio que contiene el proyecto *BlueJ* para la práctica 4.

Actividad 7: Obtención de la documentación de la clase `Instante`

Obtener la documentación de la clase pasando del modo de edición o *Implementación* al modo *Interfaz* o *Documentación*, como se muestra en la Figura 6.

[1] Esta `TestUnit` sólo comprobará el método `valueOf()` en caso de que exista dentro de la clase `Instante`. Por tanto, se puede pasar el test antes de realizar la actividad extra, y volver a pasarlo cuando esté implementado dicho método. Entonces la `TestUnit` comprobará su funcionamiento.

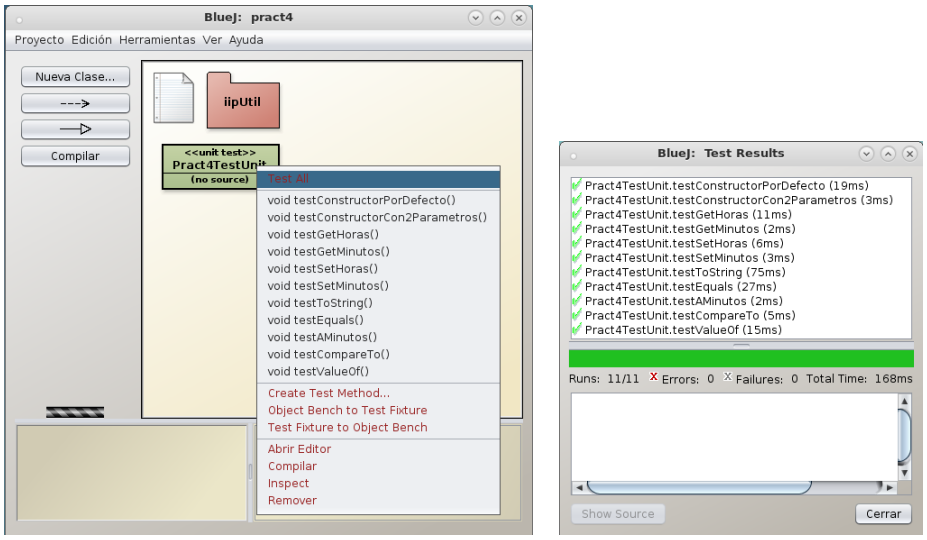


Figura 4: Ejecución de todos los tests incluidos en la *TestUnit* `Pract4TestUnit.class`

```
[alumniip@lxlabs pract4]$ ls -la
total 3072
drwxr-xr-x 4 alumniip alumnos     0 sep 29 13:11 .
drwxr-xr-x 7 alumniip alumnos     0 sep 29 14:07 ..
drwxr-xr-x 4 alumniip alumnos     0 sep 28 16:34 doc
drwxr-xr-x 2 alumniip alumnos     0 sep 29 14:26 iipUtil
-rw-r--r-- 1 alumniip alumnos   686 sep 29 13:10 package.bluej
-rw-r--r-- 1 alumniip alumnos  8450 sep 28 22:40 Pract4TestUnit.class
-rw-r--r-- 1 alumniip alumnos   471 sep 28 14:58 README.TXT
[alumniip@lxlabs pract4]$ ls -la iipUtil
total 4096
drwxr-xr-x 2 alumniip alumnos     0 sep 29 14:26 .
drwxr-xr-x 4 alumniip alumnos     0 sep 29 13:11 ..
-rw-r--r-- 1 alumniip alumnos  2078 sep 29 13:10 Instante.class
-rw-r--r-- 1 alumniip alumnos  2577 sep 29 13:10 Instante.ctxt
-rwxr--r-- 1 alumniip alumnos  4083 sep 29 13:09 Instante.java
-rw-r--r-- 1 alumniip alumnos   535 sep 29 14:28 package.bluej
[alumniip@lxlabs pract4]$
```

Figura 5: Ubicación del fichero `Pract4TestUnit.class` dentro del proyecto *BlueJ*

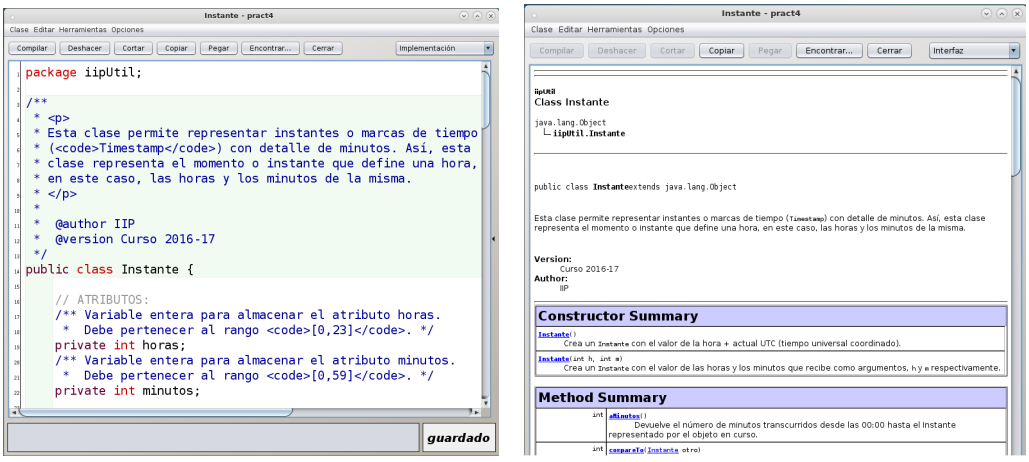


Figura 6: Ejemplo de como pasar al modo *interfaz* o *documentación*

Actividad 8: Desarrollo de la clase `Practica4`

Añadir al proyecto `pract4` una nueva clase `Practica4` que resuelva el mismo problema que en la clase `Practica3`, pero utilizando la clase `Instante`. Es decir, `Practica4` será en buena medida una reescritura literal de la clase `Practica3`, pero tanto la hora cuyos datos se introducen por teclado, como la hora actual serán objetos de la clase `Instante`, y se escribirán en el terminal en el formato deseado usando el método `toString` de la clase.

Para calcular la diferencia en minutos entre ambas horas, se puede resolver de una forma análoga a como se resuelve en `Practica3`, obteniendo los datos de cada hora mediante los métodos consultores, o bien usar el método `aMinutos` de la clase.

Actividad extra: Ampliación de la clase `Instante`. Método `valueOf`

Una vez resueltas las actividades anteriores que contemplan los objetivos básicos de la práctica, se propone la siguiente actividad extra que se puede resolver en el laboratorio si queda suficiente tiempo. En cualquier caso, constituye un ejercicio que puede permitir al alumno repasar en su tiempo de estudio algunos conceptos básicos acerca de los tipos `char` y `String`.

En este ejercicio se pide añadir a la clase `Instante` un método con el perfil:

```
/** Devuelve un <code>Instante</code> a partir de la descripción
 *  textual (<code>String</code>) en formato "<code>hh:mm</code>".
 */
public static Instante valueOf(String hhmm)
```

que dada la representación en formato `"hh:mm"` de un instante, calcule y retorne el objeto de la clase `Instante` correspondiente. Fíjese el lector cómo el método `valueOf` es un método estático que no podrá ejecutarse con respecto a ningún objeto preexistente, y el único dato con el que trabaja el método es el objeto de la clase `String` que recibe como argumento.

El método deberá calcular los valores enteros correspondientes al instante que representa el parámetro `hhmm`, y con ellos, creará y retornará el objeto `Instante` correspondiente. Para el cálculo del valor de ambos valores, es conveniente tener en cuenta las siguientes consideraciones:

- Los caracteres `hhmm.charAt(0)` y `hhmm.charAt(1)` corresponden respectivamente a los dígitos de las decenas y unidades de las horas, mientras que los caracteres `hhmm.charAt(3)` `hhmm.charAt(4)` corresponden a los dígitos análogos de los minutos.

- Aunque existe compatibilidad entre `char` e `int` dado que para Java internamente un `char` es un código numérico entero, hay que recordar que los códigos de los caracteres `'0'` a `'9'` no se corresponden a los valores enteros 0 a 9. No obstante, como dichos códigos son consecutivos, si `d` es un `char` que contiene un dígito cualquiera, la expresión entera `d - '0'` calcula el valor entero correspondiente. Así, si `d` vale `'0'` dicha expresión vale 0, si `d` vale `'1'`, la expresión se evalúa a 1, ..., como se observa en los ejemplos del *Code Pad* de la Figura 7.

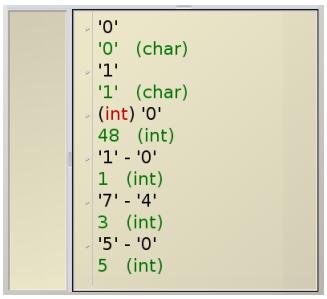


Figura 7: Ejemplo de cómo obtener el valor entero a partir de operaciones con valores de tipo `char`