IIP (E.T.S. de Ingeniería Informática)
Curso 2019-2020

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 **1**

2. Descripción del problema **1**

3. Actividades de laboratorio **2**

1. Objetivos y trabajo previo a la sesión de prácticas

El objetivo principal de esta práctica es el diseño de una clase *"Tipo de Datos"*. 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). 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.

2. Descripción del problema

En esta práctica se va a desarrollar la clase `TimeInstant` 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 la clase `TimeInstant`, se implementará una Clase Programa `Test4` que ejecute una serie de instrucciones similares a las de la práctica anterior, `Test3`, pero utilizando los objetos y métodos de la clase `TimeInstant`.

El nombre de la clase, `TimeInstant`, refleja lo que se quiere representar: una marca de tiempo (`Timestamp`); por lo tanto, la clase representa el momento que define un instante de tiempo, en este caso, las horas y los minutos de un día cualquiera.

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), en esta práctica, para simplificar, la clase `TimeInstant` únicamente manejará dos atributos, `hours` y `minutes`.

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.

Constructor Summary

Constructors

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

Method Summary

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

Modifier and Type	Method and Description
`int`	`compareTo(TimeInstant tInstant)` Compara cronológicamente el instante del objeto en curso con el del objeto de la clase `TimeInstant` referenciado por `tInstant`.
`boolean`	`equals(java.lang.Object o)` Devuelve `true` sii `o` es un objeto de la clase `TimeInstant` y sus horas y minutos coinciden con los del objeto en curso.
`int`	`getHours()` Devuelve las horas del TimeInstant.
`int`	`getMinutes()` Devuelve los minutos del TimeInstant.
`void`	`setHours(int hh)` Actualiza las horas del TimeInstant.
`void`	`setMinutes(int mm)` Actualiza los minutos del TimeInstant.
`int`	`toMinutes()` Devuelve el número de minutos transcurridos desde las 00:00 hasta el instante representado por el objeto en curso.
`java.lang.String`	`toString()` Devuelve el TimeInstant en el formato "`hh:mm`".
`static TimeInstant`	`valueOf(java.lang.String textInstant)` Devuelve un `TimeInstant` a partir de la descripción textual (`String`) en formato "`hh:mm`".

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

3. Actividades de laboratorio

Actividad 1: creación del paquete BlueJ `pract4`

a) Descarga los ficheros `TimeInstant.java` y `TimeInstantUnitTest.class`, disponibles en la carpeta de material para la práctica 4 de *PoliformaT*.

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

b) Abre el proyecto *BlueJ* de trabajo de la asignatura (`iip`).

c) Crea un nuevo paquete (Edición - Nuevo Paquete) de nombre `pract4` y ábrelo con un doble clic.

d) Agrega al paquete `pract4` la clase `TimeInstant.java` (Edición - Agregar Clase desde Archivo). Comprueba que la primera línea incluye la directiva del compilador para indicar que es una clase perteneciente a un paquete (`package pract4;`), y a continuación, compila la clase `TimeInstant`.

e) Copia el archivo `TimeInstantUnitTest.class` a la carpeta `iip/pract4`.

f) Finalmente, cierra el proyecto `iip` de *BlueJ*, y reábrelo para que tenga efecto el paso anterior.

Actividad 2: desarrollo y prueba de la clase `TimeInstant`. Atributos y métodos constructores

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

```
private int hours;
private int minutes;
```

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

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

Fíjate 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 `TimeInstant` con los valores de horas y minutos correspondientes al TimeInstant actual según UTC. Es decir, este método deberá encapsular los cálculos que se realizaban en la práctica anterior para calcular horas y minutos del tiempo UTC.

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

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 .

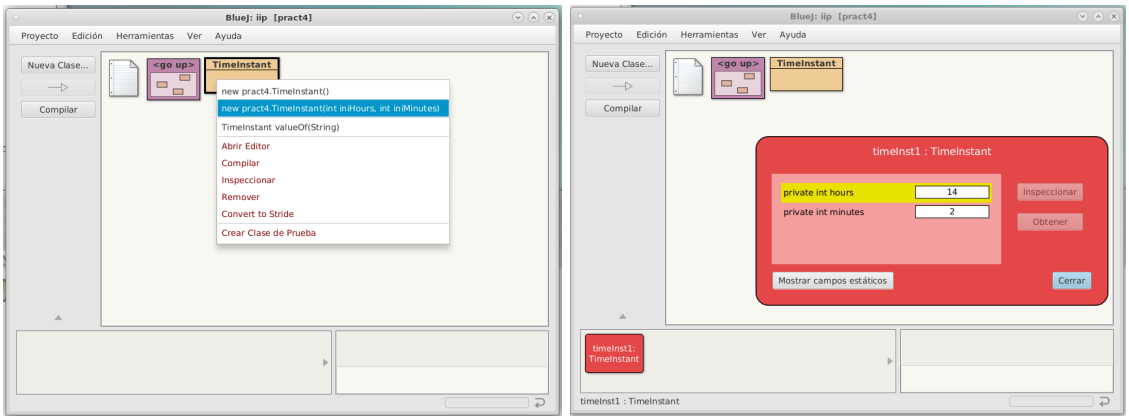

Figura 2: Ejemplo de cómo crear objetos de tipo `TimeInstant` e inspeccionarlos.

Actividad 3: desarrollo y prueba de la clase `TimeInstant`. Métodos consultores y modificadores

Añade a la clase los métodos consultores y modificadores siguientes:

```
/** Devuelve las horas del TimeInstant. */
public int getHours()

/** Devuelve los minutos del TimeInstant. */
public int getMinutes()

/** Actualiza las horas del TimeInstant. */
public void setHours(int hh)

/** Actualiza los minutos del TimeInstant. */
public void setMinutes(int mm)
```

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

Actividad 4: desarrollo y prueba de la clase `TimeInstant`. Métodos `toString`, `equals`, `toMinutes` y `compareTo`

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

```
/** Devuelve el TimeInstant en el formato "<code>hh:mm</code>". */
public String toString()

/** Devuelve <code>true</code> sii <code>o</code> es
 *  un objeto de la clase <code>TimeInstant</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 toMinutes()

/** Compara cronológicamente el instante del objeto en curso
 *  con el del objeto de la clase <code>TimeInstant</code>
 *  referenciado por <code>tInstant</code>.
 *  <p>
 *  El resultado es la resta entre la conversión a minutos
 *  de ambos objetos, en particular, este resultado será un valor:
 *  <ul>
 *     <li> negativo si el instante del objeto en curso es anterior
 *          al del <code>tInstant</code>, </li>
 *     <li> cero si son iguales, </li>
 *     <li> positivo si el instante del objeto en curso es posterior
 *          al del <code>tInstant</code>. </li>
 *  </ul>
 *  </p>
 */
public int compareTo(TimeInstant tInstant)
```

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 el parámetro `o` es un objeto de la clase `TimeInstant` y, en caso afirmativo, si sus atributos coinciden en valor con los del objeto en curso:

```
o instanceof TimeInstant
&& this.hours == ((TimeInstant) o).hours
&& this.minutes == ((TimeInstant) o).minutes
```

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

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

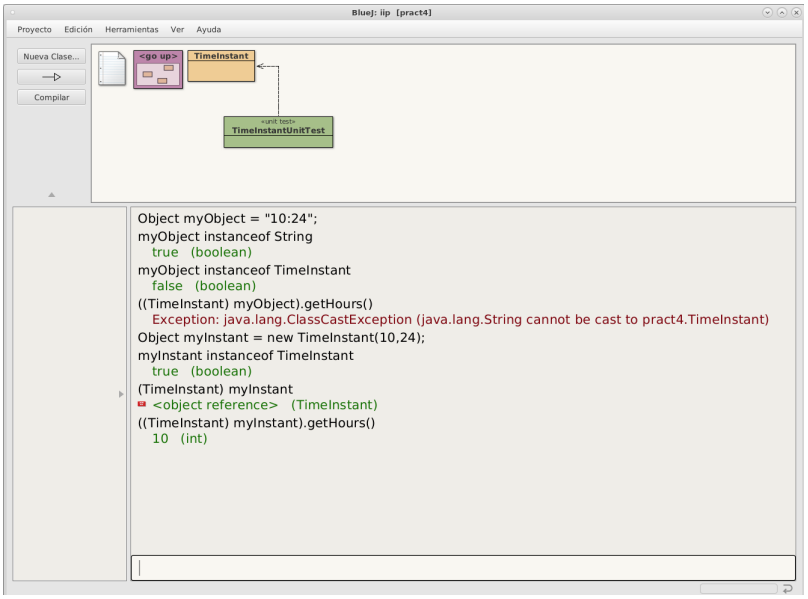

Figura 3: Ejemplo de cómo probar el comportamiento de `instanceof` con ayuda del *Code Pad*.

La clase se debe recompilar y hay que comprobar los métodos añadidos. Por ejemplo, para probar `equals` y `compareTo` se pueden crear tres TimeInstants `timeInstant1`, `timeInstant2` y `timeInstant3`, correspondientes a las 00:00, 12:10, 12:10 respectivamente, y comprobar que:

- `timeInstant2` y `TimeInstant3` son iguales,
- `timeInstant1` es anterior a `timeInstant2`,
- `timeInstant2` es posterior a `timeInstant1`.

Actividad 5: comprobación de las normas de estilo de la clase `TimeInstant`

Comprueba que el código de la clase escrita cumple las normas de estilo usando el `Checkstyle` de *BlueJ*, y corrígelo si no es el caso.

Actividad 6: comprobación del funcionamiento de la clase `TimeInstant`

Para comprobar el correcto funcionamiento de los métodos de la clase `TimeInstant`, se ha proporcionado la clase `TimeInstantUnitTest`. Esta clase es una `TestUnit` que se ejecuta como se puede ver en la Figura 4 . El resultado del test aparece reflejado en la ventana *Probar Resultados* de *BlueJ*. Si los métodos son correctos, aparecerán marcados con el símbolo ✓ (de color verde). Si, por el contrario, algún método no funciona correctamente, aparecerá marcado con el símbolo `X`. Con un clic de ratón, aparecerá un mensaje orientativo sobre la posible causa del error en la parte inferior de la ventana.

Actividad 7: obtención de la documentación de la clase `TimeInstant`

Genera la documentación de la clase pasando del modo *Código Fuente* al modo *Documentación*. Para ello, basta con pulsar el botón correspondiente situado en la parte superior derecha de la ventana, tal y como se muestra en la Figura 5 .

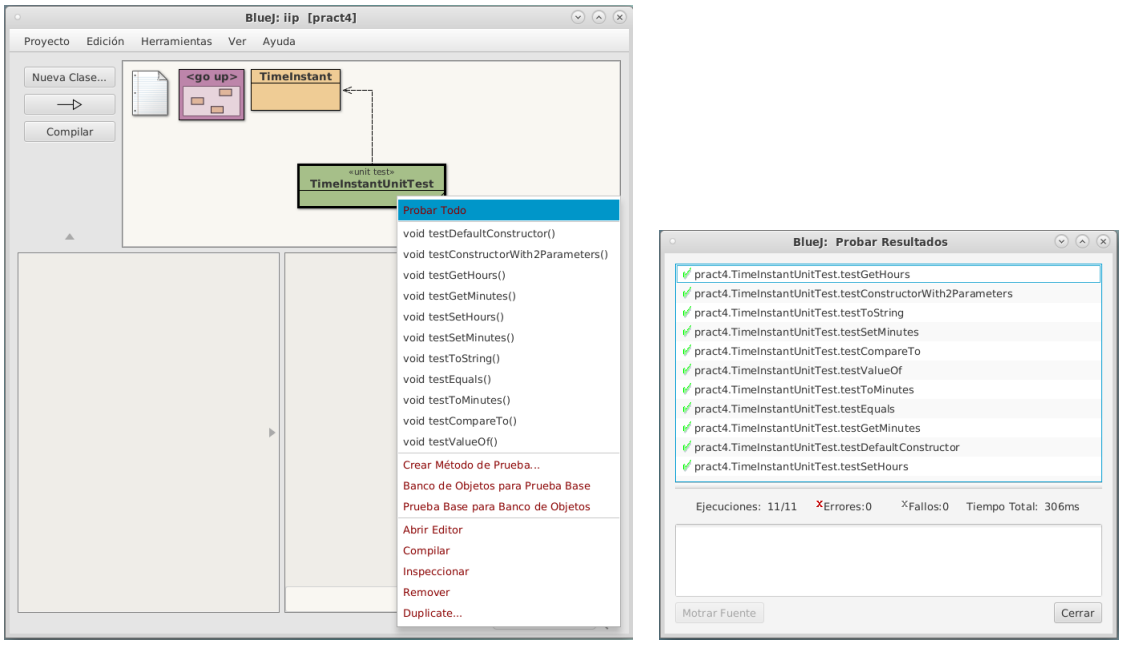

Figura 4: Ejecución de todos los tests incluidos en la *TestUnit* `TimeInstantUnitTest.class`.

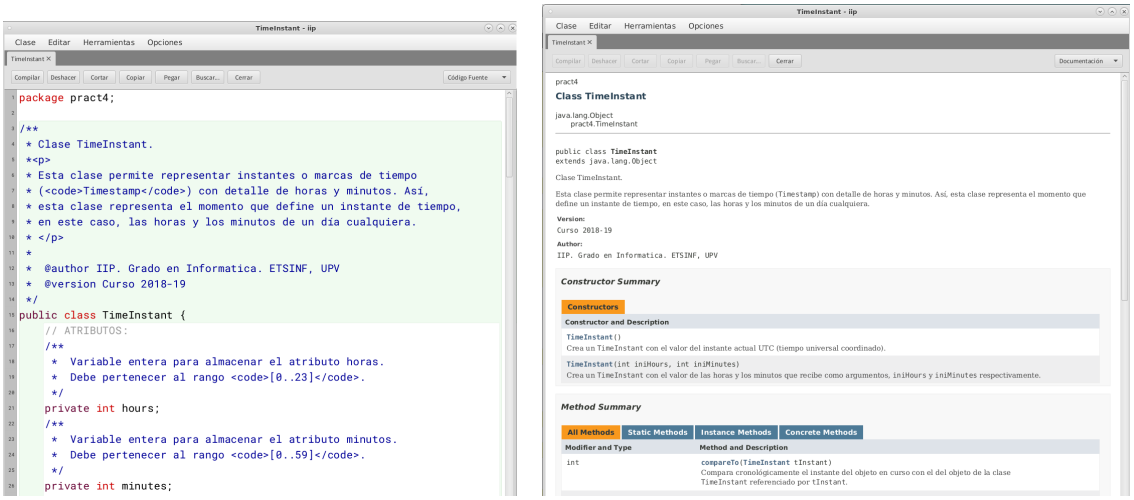

Figura 5: Ejemplo de cómo pasar al modo *Código Fuente* o *Documentación*.

Actividad 8: desarrollo de la clase `Test4`

Añade al paquete `pract4` una nueva clase `Test4` que resuelva el mismo problema que el de la práctica 3, pero esta vez usando la clase `TimeInstant`. Es decir, `Test4` será en buena medida una reescritura del código de la práctica 3, pero tanto la representación temporal que se introduce por teclado, como el tiempo UTC se tienen que gestionar mediante objetos de la clase `TimeInstant`, mostrándose por pantalla en el formato deseado usando el método `toString`.

Para calcular la diferencia en minutos entre ambos instantes de tiempo, se puede resolver de una forma análoga a como se hizo en la clase `Test3`, obteniendo los atributos de cada objeto mediante los métodos consultores correspondientes, o bien haciendo uso de los métodos `toMinutes` o `compareTo` de la clase `TimeInstant`.

Actividad extra: ampliación de la clase `TimeInstant`. 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 `TimeInstant` un método con el perfil siguiente:

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

que dado un instante de tiempo expresado mediante una cadena de texto en el formato `"hh:mm"`, calcule y retorne el objeto de la clase `TimeInstant` correspondiente. Fíjate en que el método `valueOf` es un método *estático*, y por lo tanto, no se podrá invocar con respecto a ningún objeto preexistente, siendo el objeto `textInstant` de la clase `String` que recibe como argumento, su único parámetro de entrada.

El método debe calcular los valores enteros correspondientes al instante representado por `textInstant`, y con ellos, debe crear y retornar el objeto `TimeInstant` correspondiente. Para el cálculo de dichos valores, es conveniente tener en cuenta las siguientes consideraciones:

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

- Aunque existe compatibilidad entre los tipos `char` e `int`, dado que internamente un dato de tipo `char` es un código numérico entero, hay que recordar que los códigos de los caracteres `'0'` a `'9'` no se corresponden con 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 `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, etc., como se observa en los ejemplos del *Code Pad* de la Figura 6 .

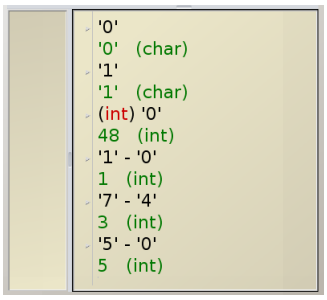

Figura 6: Ejemplo de cómo obtener el valor entero a partir de operaciones con valores de tipo `char`.