IIP (E.T.S. de Ingeniería Informática)
Year 2017-2018

Lab activity 4 - Java Classes Development and Reuse

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Contents

1 Objectives and previous work

The main objective of this lab activity is to design a datatype class. More specifically, these concepts of Units 2 and 4 will be developed:

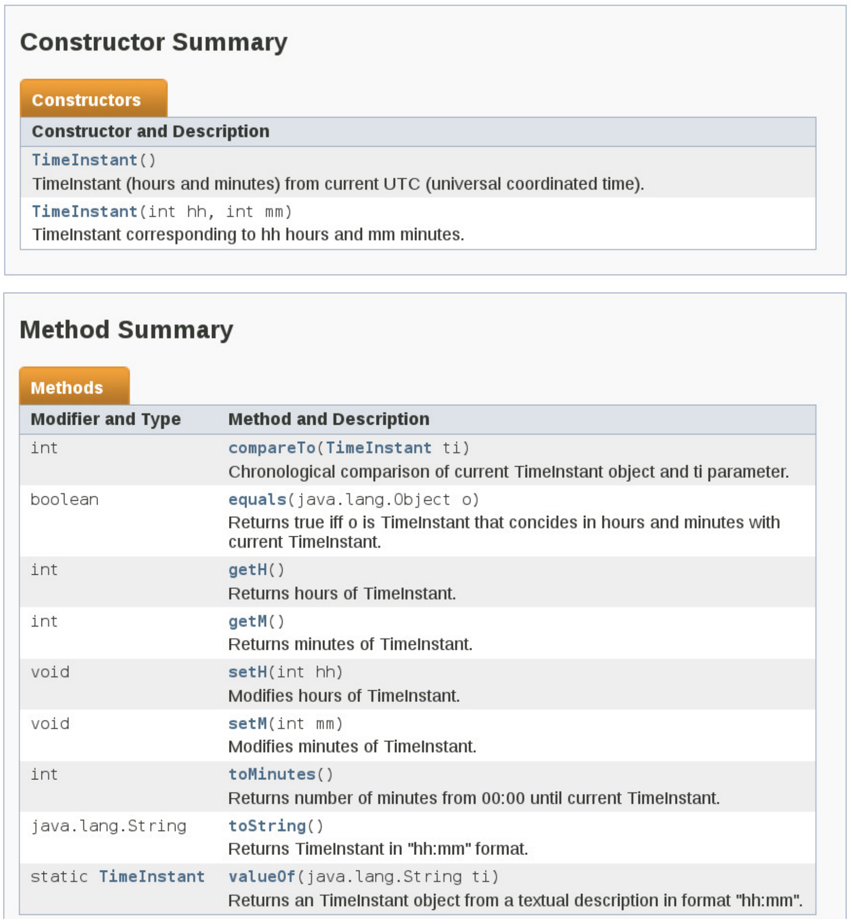
- Implementation of a class (as a structure for objects)
- Implementation of constructors, consultors (`get`), modifiers (`set`), and other methods
- Use of the implemented datatype class: reference vars declaration, object creation, and object use (via methods)

2 Problem description

In this lab activity, the implementation of a class `TimeInstant` is required. Class `TimeInstant` must provide the functionality described in the except of its documentation shown in Figure 1.

After developing the `TimeInstant` class, a program class `LabAct4` must be developed to obtain the same functionality than the program class of the previous lab activity (`LabAct3`), but by using objects and methods of the `TimeInstant` class.

The name of the class, `TimeInstant`, reflects what is represented: a time stamp. Thus, this class represents the instant which defines an hour, in this case by the hours and minutes of that hour. Anyway, a time stamp usually includes more details (year, month, day, hours, minutes, seconds, and mili/microseconds, according its implementation), but in this lab activity it is simplified in the `TimeInstant` class by using only two attributes (for hours and minutes). Notice that *timestamp* is the usual term employed in databases for managing data items that corresponds to times.



Constructor Summary

Constructors

Constructor and Description
`TimeInstant()` TimeInstant (hours and minutes) from current UTC (universal coordinated time).
`TimeInstant(int hh, int mm)` TimeInstant corresponding to hh hours and mm minutes.

Method Summary

Methods

Modifier and Type	Method and Description
`int`	`compareTo(TimeInstant ti)` Chronological comparison of current TimeInstant object and ti parameter.
`boolean`	`equals(java.lang.Object o)` Returns true iff o is TimeInstant that concides in hours and minutes with current TimeInstant.
`int`	`getH()` Returns hours of TimeInstant.
`int`	`getM()` Returns minutes of TimeInstant.
`void`	`setH(int hh)` Modifies hours of TimeInstant.
`void`	`setM(int mm)` Modifies minutes of TimeInstant.
`int`	`toMinutes()` Returns number of minutes from 00:00 until current TimeInstant.
`java.lang.String`	`toString()` Returns TimeInstant in "hh:mm" format.
`static TimeInstant`	`valueOf(java.lang.String ti)` Returns an TimeInstant object from a textual description in format "hh:mm".

Figure 1: `TimeInstant` class API documentation

3 Lab activities

Activity 1: creation of the BlueJ package `labact4`

1. Download the files `TimeInstant.java` and `Lab4TestUnit.class` availables in the folder of the lab activity in PoliformaT.

   The `TimeInstant.java` file contains the skeleton of the class to be developed, including the comments that must precede each of the methods. The file `Lab4TestUnit.class` would be used to check that you made a correct implementation of the `TimeInstant` class (see Activity 6)

2. Open the *BlueJ* project for the subject (`iip`)

3. Create the new package `labact4` by using Edition - New package; open it by double-cliking on it

4. Add to the package `labact4` the class `TimeInstant.java` (Edition - Add class from file). Check that the first line of code includes the line that tells the compiler that the

class pertains to the right package (`package labact4;`), and then compile it

5. Copy on the folder `iip/labact4` the `Lab4TestUnit.class` file
6. Close the `iip` project in *BlueJ* and re-open it to make the previous step to take effect

Activity 2: development and test of the `TimeInstant` class: attributes and constructors

As said previously, each object of the datatype `TimeInstant` keeps the information of the hours and minutes that define a time stamp. Thus, the attributes are:

```
    private int h;
    private int m;
```

The class must include a first constructor method with header:

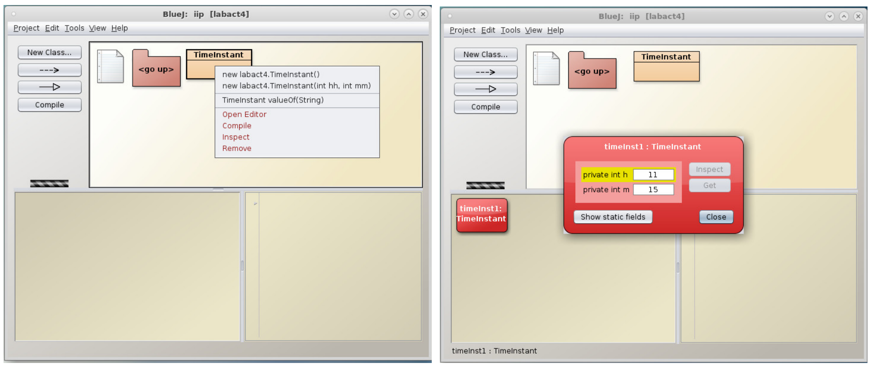
```
    /** <code>TimeInstant</code> corresponding to <code>hh</code>
     *  hours and <code>mm</code> minutes.
     *  <p> Precondition: <code>0<=hh<24, 0<=mm<60</code> </p>
     */
public TimeInstant(int hh, int mm)
```

Notice that comments may include some `HTML` tags (such as `<code>` or `<p>`) to allow a better view in the browser.

Apart from this, a default constructor with no parameters must be implemented, which will initialise the attributes to current UTC time. Thus, this method encapsulates the calculations that in the previous lab activity allowed to calculate current UTC hour:

```
    /**
     *  <code>TimeInstant</code> (hours and minutes) from current
     *  UTC (universal coordinated time).
     */
public TimeInstant()
```

Once this class is edited and correctly compiled, test by using the Object Bench the generation and correction of `TimeInstant` objects, such like in the example of Figure 2.



private int h 11
private int m 15

Figure 2: Correct example for object `TimeInstant` creation and inspection.

Activity 3: development and test of the `TimeInstant` class: consultors and modifiers

Add to the `TimeInstant` class the following constructors and modifiers:

```
/** Returns hour of current TimeInstant object */
public int getH()

/** Returns minutes of current TimeInstant object */
public int getM()

/** Modifies hour of current TimeInstant object */
public void setH(int hh)

/** Modifies minutes of current TimeInstant object */
public void setM(int mm)
```

Before adding more methods, recompile the class and check that all methods are correct. For that, you must create objects (in *BlueJ Object Bench* or *Code Pad*) and check the methods results.

Activity 4: development and test of the `TimeInstant` class: methods `toString`, `equals`, `toMinutes`, and `compareTo`

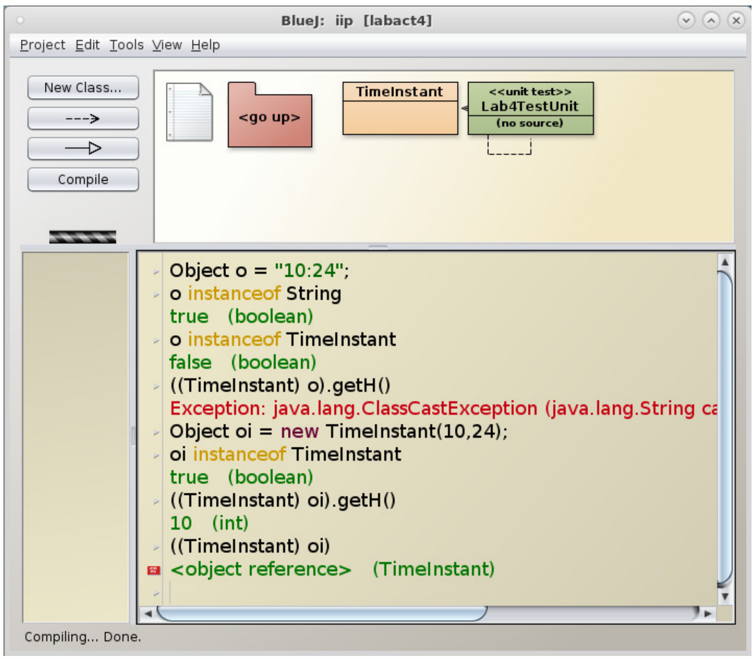
Add to the class the methods whose headers are described below:

```
/** Returns current TimeInstant object in "hh:mm" format
 */
public String toString()

/** Returns true only if o is a TimeInstant that coincides in
 *  hours and minutes with current TimeInstant
 */
public boolean equals(Object o)

/** Returns amount of minutes from
 *  00:00 until current TimeInstant object
 */
public int toMinutes()

/** Compares chronologically current TimeInstant object and ti
 *  parameter. Result is the difference between the conversion
 *  obained with <code>toMinutes</code> for current and parameter
 *  object, giving:
 *      - negative when current TimeInstant is previous to
 *        <code>ti</code>
 *      - zero if they are equal
 *      - positive when current TimeInstant is posterior to
 *        <code>ti</code>
 */
public int compareTo(TimeInstant ti)
```



```
Object o = "10:24";
o instanceof String
true   (boolean)
o instanceof TimeInstant
false   (boolean)
((TimeInstant) o).getH()
Exception: java.lang.ClassCastException (java.lang.String ca
Object oi = new TimeInstant(10,24);
oi instanceof TimeInstant
true   (boolean)
((TimeInstant) oi).getH()
10   (int)
((TimeInstant) oi)
<object reference>   (TimeInstant)
```

```
Compiling... Done.
```

Figure 3: Example on the use of `instanceof` in the Code Pad

When implementing `equals`, remember that the use of the shortcut AND (`&&`) makes important the order of the operands of the comparison between the parameter `o` and current `TimeInstant` object: first, it is checked if `o` is a `TimeInstant`, and then if the different attributes of `o` have the same value than the current `TimeInstant`:

```
o instanceof TimeInstant
&&  this.h == ((TimeInstant) o).h
&&  this.m == ((TimeInstant) o).m
```

Using this form, second and third operands will be evaluated only when parameter `o` is effectively a `TimeInstant` object. In that case, casting can be applied on object `o` and attributes can be properly accessed.

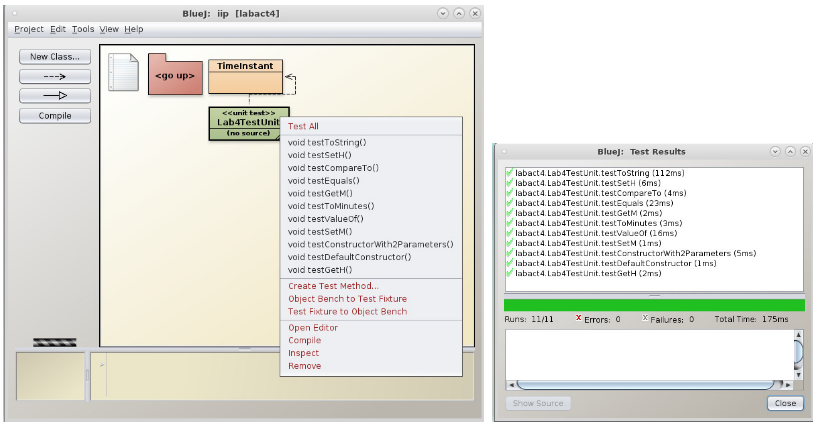
The `instanceof` operand can be tested in the *Code Pad* by using tests such like those in Figure 3.

Recompile class and test the new methods. For example, for `equals` and `compareTo` you can create three objects `ti1`, `ti2`, and `ti3` for hours 00:00, 12:10, and 12:10, respectively, and check that:

- `ti2` and `ti3` are equal
- `ti1` is previous to `ti2` (negative result of `compareTo`)
- `ti2` is posterior to `ti1` (positive result of `compareTo`)

Activity 5: checking style for the `TimeInstant` class

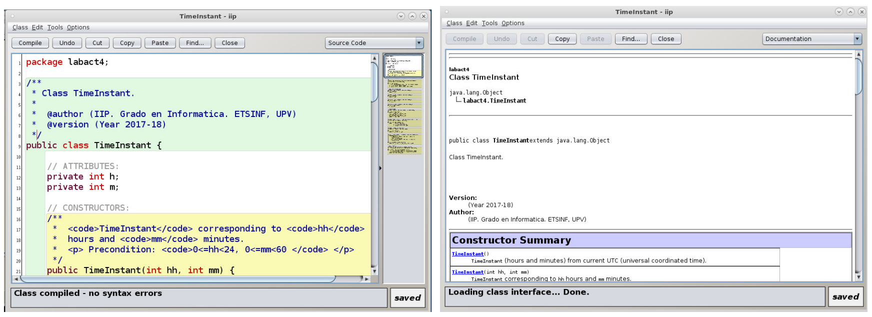
Check that the implementation of `TimeInstant` fulfils the coding style directives by using the Checkstyle of *BlueJ*, and correct the different warnings.



```
✓ labact4.Lab4TestUnit.testToString (112ms)
✓ labact4.Lab4TestUnit.testSetH (6ms)
✓ labact4.Lab4TestUnit.testCompareTo (4ms)
✓ labact4.Lab4TestUnit.testEquals (23ms)
✓ labact4.Lab4TestUnit.testGetM (2ms)
✓ labact4.Lab4TestUnit.testToMinutes (3ms)
✓ labact4.Lab4TestUnit.testValueOf (16ms)
✓ labact4.Lab4TestUnit.testSetM (1ms)
✓ labact4.Lab4TestUnit.testConstructorWith2Parameters (5ms)
✓ labact4.Lab4TestUnit.testDefaultConstructor (1ms)
✓ labact4.Lab4TestUnit.testGetH (2ms)

Runs: 11/11   Errors: 0   Failures: 0   Total Time: 175ms
```

Figure 4: Execution of all the test in the TestUnit (`Lab4TestUnit.class`



```java
package labact4;

/**
 * Class TimeInstant.
 *
 *  @author (IIP. Grado en Informatica. ETSINF, UPV)
 *  @version (Year 2017-18)
 */
public class TimeInstant {

    // ATTRIBUTES:
    private int h;
    private int m;

    // CONSTRUCTORS:
    /**
     *  <code>TimeInstant</code> corresponding to <code>hh</code>
     *  hours and <code>mm</code> minutes.
     *  <p> Precondition: <code>0<=hh<24, 0<=mm<60 </code> </p>
     */
    public TimeInstant(int hh, int mm) {
```

```
Class compiled - no syntax errors
```

labact4
Class TimeInstant

```
java.lang.Object
  └ labact4.TimeInstant
```

```
public class TimeInstantextends java.lang.Object
```

Class TimeInstant.

Version:
(Year 2017-18)
Author:
(IIP. Grado en Informatica. ETSINF, UPV)

Constructor Summary

`TimeInstant()`
`TimeInstant` (hours and minutes) from current UTC (universal coordinated time).

`TimeInstant(int hh, int mm)`
`TimeInstant` corresponding to `hh` hours and `mm` minutes.

```
Loading class interface... Done.
```

Figure 5: Example of how to pass from edition to interface mode for class `TimeInstant`.

Activity 6: checking correctness for the `TimeInstant` class

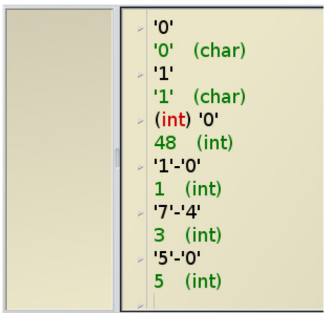
To check the correct implementation of the `TimeInstant` class, the `Lab4TestUnit` class is provided. This class is a `TestUnit` which is executed as shown in Figure 4. The test result appears in the *Test Results* window of *BlueJ*. If the methods are correct, they will be shown with ✓ (in green) in the test results window of *BlueJ*. However, when a method is not correctly implemented it will be shown with X. When selecting the incorrect test, a message will be shown on the bottom of the window with an explanation of the possible error.

Activity 7: generating documentation for `TimeInstant` class

Generate class documentation by passing from the edition (implementation) mode to the interface mode as shown in Figure 5.

Activity 8: implementation of class `LabActivity4`

Add to the package `labact4` a new program class `LabActivity4` that solves the same problem than `LabActivity3` but by using `TimeInstant` objects. That is, `LabActivity4` will be a transliteration of `LabActivity3` where:



```
'0'
'0'  (char)
'1'
'1'  (char)
(int) '0'
48  (int)
'1'-'0'
1  (int)
'7'-'4'
3  (int)
'5'-'0'
5  (int)
```

Figure 6: Transforming char digits into its numerical value.

- All time instants (that are inputed via keyboard or the current UTC time) must be stored into `TimeInstant` objects
- The time instants must be printed on the screen in the format `"hh:mm"` by using the `toString()` method of the `TimeInstant` objects
- Difference in minutes between time instants must be calculated similarly, but by employing the consultor methods, the `toMinutes` method, or the `compareTo` method of the `TimeInstant` objects

Extra activity: expansion of class `TimeInstant`: method `valueOf`

This activity is optional and can be developed in the lab is there is enough time. This activity allows to reinforce concepts on the `char` and `String` datatypes.

It is proposed to add to the `TimeInstant` class the following method:

```
/** Returns a TimeInstant from its textual description
 *  in a <code>String</code> with format "<code>hh:mm</code>".
 */
 public static TimeInstant valueOf(String hhmm)
```

This method, given a `String` which represents a time instant in format `"hh:mm"`, calculates and returns the corresponding `TimeInstant` object. It is a `static` method (is not applied to any object) that only works with the given `String`.

The method must calculate the integer values that are stored into parameter `hhmm` and then create the `TimeInstant` object that corresponds to that time. For calculating these values, you must take into account that:

- Characters in position 0 and 1 (`hhmm.charAt(0)` and `hhmm.charAt(1)`) correspond to tens and units of the hour, while those in positions 3 and 4 correspond to tens and units of the minute
- Although `char` and `int` are compatible (`char` are numbers in the range [0,65535]), codes for characters between `'0'` and `'9'` do not have the numerical values between 0 and 9; but since they are consecutive, the expression `d - '0'` gives the numerical value for `char d` if it stores a character that represents a digit (i.e., 0 for char `'0'`, 1 for `'1'`, etc.), as you can check on the examples seen in Figure 6