ETSINF – IIP First mid term exam – Academic year 2018/2019

November 5th, 2018. Time: 1 hour and 30 minutes.

Notice: The exam is marked up to 10 points, but its specific weight in the final grade of IIP is **2.4 points**

NAME:	GROUP:

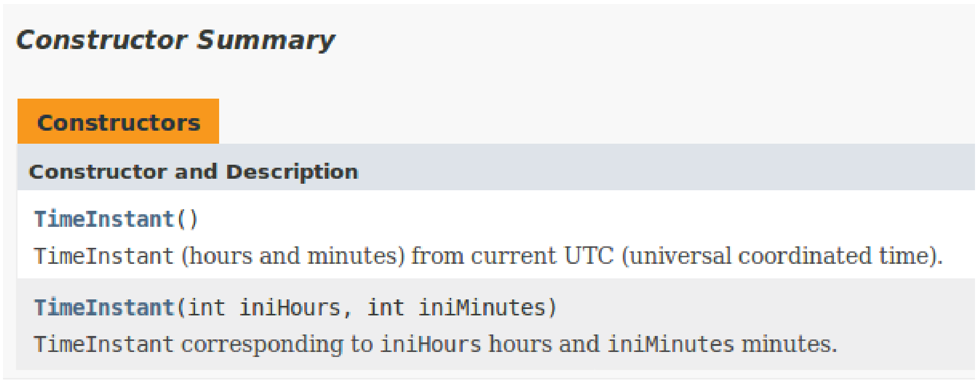
1. $\boxed{\text{6 points}}$ It is needed to design and implement a data type class named `SaveGame` in order to represent games of a given video game that are going to be saved in the memory card of a well known video game console. Each object of the class `SaveGame` has the following attributes:

region	The code of the world region corresponding to the video game. An string of four characters.
identifier	The identifier of the video game within the region, an integer of five digits. Another video game can have the same identifier in another region.
position	The position within the memory card where the saved game is stored. An integer from 1 to 10.
progress	The progress reached during the game at the moment of saving it. A real value expressing the percentage of the achieved progress.

While a game is active there is a virtual clock that only advances when the player performs correct actions, so that each instant of the virtual clock corresponds to a concrete progress level, i.e., a game completed at 100% has a duration of 24 hours. The following table shows several percentages of progress and its corresponding time in hours and minutes:

progress	0%	50%	80%
virtual clock	00 : 00	12 : 00	19 : 12

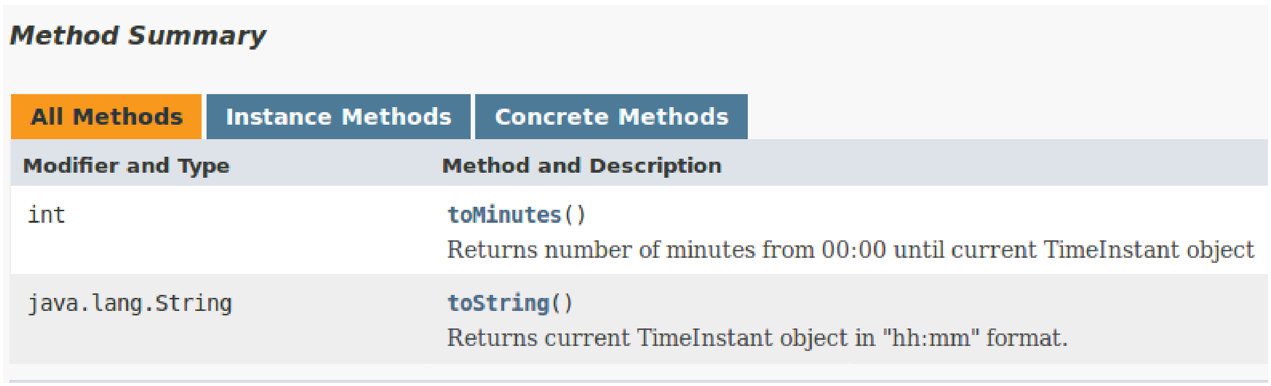
It is available the class `TimeInstant` for representing the virtual clock. Next figure shows the part of the documentation of the class `TimeInstant` needed for this exam.



Constructor Summary

Constructors

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



Method Summary

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

Modifier and Type	Method and Description
`int`	`toMinutes()` Returns number of minutes from 00:00 until current TimeInstant object
`java.lang.String`	`toString()` Returns current TimeInstant object in "hh:mm" format.

You must implement the class `SaveGame` that must be located in the same directory the class `TimeInstant` is, and with the following constants, attributes and methods:

a) (0.25 points) Constants declared as public class variables:

- `MINUTES_PER_DAY`, an integer with value $1440 = 24 * 60$ for representing the minutes in a whole day.

This constant must be used wherever required, for instance in classes `SaveGame` and `GameManager`.

b) (0.5 points) Private attributes (instance variables): `region` (`String`), `identifier` (`int`), `position` (`int`) and `progress` (`float`).

c) (1.25 points) A generic constructor with four parameters: an string with the value of the region, an integer with the identifier of the video game, an integer with the position in the memory card, and an object of the class `TimeInstant` with the virtual duration of the game. This parameter must be used for initialising the attribute `progress` as a percentage. You can assume that all the real parameters used in the call to the constructor are correct.

d) (1 point) A method `toHHMM()` for returning the progress of the saved game as an string with the format `hh:mm`. For instance, if the progress is equal to 59.9% the string returned by `toHHMM()` must be `"14:08"`.

e) (1 point) An `equals` method that must override the `equals` method of the `Object` class. This method must check if two objects of the class `SaveGame` are the same by taking into account the region, the identifier and the progress. The position in the memory card is not relevant for this purpose.

f) (1 point) A `toString()` method that must override the `toString()` method of the `Object` class and return an string as representation of the saved game with the format shown in the following three examples:

```
PAL: SCES_507.60 - 1 - 17.3%
USA: SCUS_971.13 - 2 - 24.3%
JAP: SLPS_204.01 - 3 - 58.9%
```

The structure of the string is the following one:

```
"format: region_quotient - position - progress%"
```

where format can be `PAL` if region is `SCES` or `SLES`, `USA` if region is `SCUS` or `SLUS`, or `JAP` for any other region. Quotient is obtained by dividing the identifier by 100, and progress must be shown in percentage with one decimal digit.

g) (1 point) The criteria to follow for sorting saved games is based on three attributes: region, identifier and progress.

- The first attribute to consider is the region. If regions differ, then the order of both compared objects is determined by the lexicographical order of the region. You can use the method `compareTo()` of the class `String` applied to attribute region.
- If the region of both saved games is the same, then the order of both compared objects is determined but the identifier.
- Finally, if the region and the identifier of both save games are the same, then the order of both compared objects is determined by the progress.

You have to implement the method `compareTo()` that given a parameter named `sg` of the class `SaveGame`, returns a negative integer if the current object (`this`) must go before `sg`, a positive number if the current object (`this`) must go after `sg`, and zero when the order of both objects is the same.

Solution:

```
public class SaveGame
{
    public static final int MINUTES_PER_DAY = 24*60;

    private String          region;
    private int             number;
    private int             position;
    private float           progress;

    public SaveGame( String region, int number, int position, TimeInstant t )
    {
        this.region = region;
        this.number = number;
        this.position = position;
        this.progress = (100 * t.toMinutes()) / (float)MINUTES_PER_DAY;
    }

    public String toHHMM()
    {
        int minutes = (int)Math.floor( this.progress * MINUTES_PER_DAY / 100.0 + 0.5 );

        return String.format( "%02d:%02d", minutes/60, minutes%60 );
    }

    @Override
    public boolean equals( Object o )
    {
        if ( o instanceof SaveGame ) {

            SaveGame other = (SaveGame)o;

            return this.region.equals( other.region )
                && this.number == other.number
                && this.progress == other.progress;

        } else {
            return false;
        }
    }

    @Override
    public String toString()
    {
        String format="JAP";

        switch( region ) {
            case "SCES" :
            case "SLES" : format="PAL"; break;

            case "SCUS" :
            case "SLUS" : format="USA"; break;
        }
        return String.format( "%s: %s_%06.2f - %d - %4.1f%%",
                              format, region, number/100.0,
                              position, progress );
    }

    public int compareTo( SaveGame other )
    {
        int rc = this.region.compareTo( other.region );

        if ( rc == 0 ) {
            rc = this.number - other.number;
            if ( rc == 0 ) {
                if ( this.progress < other.progress ) {
                    rc = -1;
                } else if ( this.progress > other.progress ) {
                    rc = 1;
                }
            }
        }

        return rc;
    }
}
```

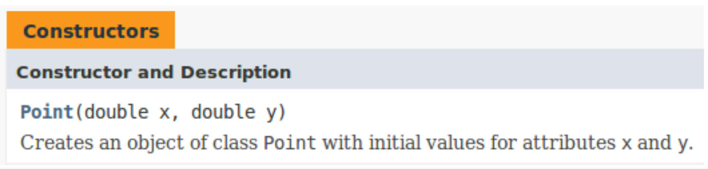
2. 2 points **You must** implement a program class named `GameManager`, in the same directory where class `SaveGame` is, with a `main` method that executes the following actions:

   a) (0.25 points) Create an object `t` of the class `TimeInstant` with the current time in UTC. You can use the default constructor of the class `TimeInstant`.
   b) (0.5 points) Create an object `sg` of the class `SaveGame` with the region `SCES`, the identifier 50760, stored at position 1, and with the progress level corresponding to the hours and minutes of `t`.
   c) (1.25 points) Print the result of executing the method `toHHMM()` with respect to the object `sg`. Print the result of executing the method `toString()` with respect to the object `t`. And, finally, print the result of comparing both strings `sg.toHHMM()` and `t.toString()`.

Solution:

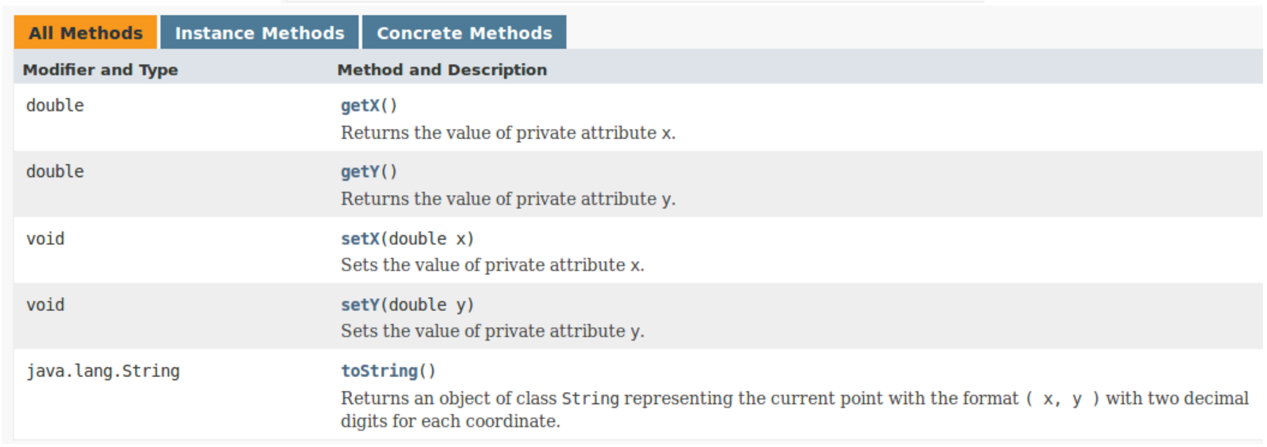
```
public class GameManager
{
    public static void main( String [] args )
    {
        TimeInstant t = new TimeInstant();

        SaveGame sg = new SaveGame( "SCES", 50760, 1, t );

        System.out.println( "The virtually elapsed time is: " + sg.toHHMM() );
        System.out.println( "That must match with.........: " + t );

        System.out.println( "Are both strings the same? " + sg.toHHMM().equals( t.toString() ) );
    }
}
```

3. 2 points It is available the class `Point` that defines a point in the two-dimensional space with two attributes (abscissa and ordinate). Part of the functionality of this class is shown in the following figures:



Constructors

Constructor and Description
`Point(double x, double y)` Creates an object of class `Point` with initial values for attributes `x` and `y`.



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

Modifier and Type	Method and Description
`double`	`getX()` Returns the value of private attribute x.
`double`	`getY()` Returns the value of private attribute y.
`void`	`setX(double x)` Sets the value of private attribute x.
`void`	`setY(double y)` Sets the value of private attribute y.
`java.lang.String`	`toString()` Returns an object of class `String` representing the current point with the format `( x, y )` with two decimal digits for each coordinate.

Given the following Java program:

```
public class Exercise3
{
    public static void main( String[] args )
    {
        Point p = new Point( 1.0, -1.0 );

        double x = p.getX();
        double y = p.getY();

        System.out.print( "Before calling changeCoords(): " );
        System.out.println( "x = " + x + ", y = " + y + ", p = " + p.toString() );

        changeCoords( x, y, p );
        System.out.print( "After the first call to changeCoords(): " );
        System.out.println( "x = " + x + ", y = " + y + ", p = " + p.toString() );

        x = p.getY();
        y = p.getX();
        changeCoords( x, y, p );
        System.out.print( "After the second call to changeCoords(): " );
        System.out.println( "x = " + x + ", y = " + y + ", p = " + p.toString() );
    }

    public static void changeCoords( double x, double y, Point p )
    {
        double z = x; x = y; y = z;

        p.setX(x);
        p.setY(y);
    }
}
```

You have to complete the gaps:

`Before calling changeCoords(): x =` ___1.0___ `, y =` ___-1.0___ `, p = (`___1.0___ `,` ___-1.0___ `)`

`After the first call to changeCoords(): x =` ___1.0___ `, y =` ___-1.0___ `, p = (`___-1.0___ `,` ___1.0___ `)`

`After the second call to changeCoords(): x =` ___1.0___ `, y =` ___-1.0___ `, p = (`___-1.0___ `,` ___1.0___ `)`