

Class CyclicCounter

1/3

```
/*
 * CyclicCounter
 *
 * @author Konstantinos Filios (drcypher@edrasemfe.gr)
 * @version 1.0
 */
public class CyclicCounter
{
    private int limit;
    private int count;

    /**
     * Returns the counter's value as a string with the
     * same number of digits as the limit, filling those
     * missing with zeros.
     *
     * @return      String version of internal counter
     */
    public String printValue()
    {
        String limitString = String.valueOf(limit); // Alternative: ""+li
mit
        String countString = String.valueOf(count); // Alternative: ""+co
unt
        for (int i = limitString.length() - countString.length(); i > 0;
i--)
            countString = "0" + countString;
        return countString;
        // Alternative (only for two digits, thus not correct ::):
        // if (count < 10)
        //     return " " + count;
        // else
        //     return "0" + count;
    }

    /**
     * Increments the counter value by one or resets it
     * the limit is reached.
     */
    public void increment()
    {
        if (count == limit - 1)
            this.reset();
        else
            count++;

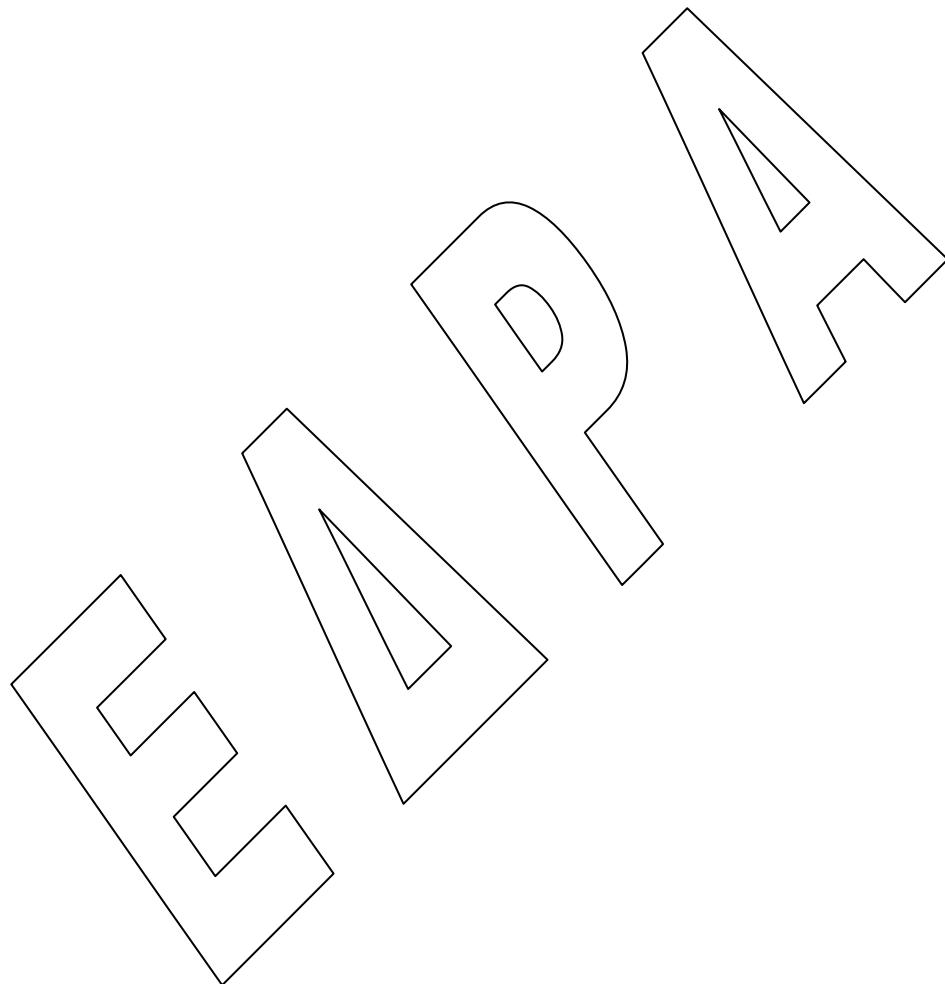
        // Alternative:
        // count = (count + 1) % limit;
    }
}
```

```
/**  
 * Returns the current counter value as an integer.  
 *  
 * @return Integer value of counter  
 */  
public int getCount()  
{  
    return count;  
}  
  
/**  
 * Sets a new value for the counter, as long as this  
 * new value is within the limit.  
 *  
 * @param newCount Counter's new value  
 */  
public void setCount(int newCount)  
{  
    if ((0 <= newCount) && (newCount < limit))  
        count = newCount;  
}  
  
/**  
 * Returns the current limit value as an integer.  
 *  
 * @return Integer value of limit  
 */  
public int getLimit()  
{  
    return limit;  
}  
  
/**  
 * Sets a new value for the limit and resets the counter.  
 *  
 * @param newLimit New limit value  
 */  
public void setLimit(int newLimit)  
{  
    limit = newLimit;  
    this.reset();  
}  
  
/**  
 * Resets the counter to 0.  
 */  
public void reset()  
{  
    count = 0;  
}  
/**
```

Class CyclicCounter (continued)

3/3

```
* Class constructor. Initializes counter's limit
* and resets it (through this.setLimit()).
*/
public CyclicCounter(int initialLimit)
{
    this.setLimit(initialLimit);
}
}
```



Class DisplayClock

1/3

```
/*
 * DisplayClock
 *
 * @author Konstantinos Filios (drcypher@edrasemfe.gr)
 * @version 1.0
 */
public class DisplayClock
{
    private CyclicCounter hour = new CyclicCounter(24);
    private CyclicCounter minute = new CyclicCounter(60);
    private boolean isFullDay;
    private boolean isAM;
    private String display;

    /**
     * Class constructor. Initializes variable isFullDay and
     * the time to "00:00 (AM)"
     *
     * @param newIsFullDay True for 24-hour clock, false for 12-hour
     */
    public DisplayClock(boolean newIsFullDay)
    {
        isFullDay = newIsFullDay;

        if (isFullDay)
            this.setTime(0, 0);
        else
            this.setTime(0, 0, true);
    }

    /**
     * Sets the time in 24-hour format.
     *
     * @param h Hours (0 - 23)
     * @param m Minutes (0 - 59)
     */
    public void setTime(int h, int m)
    {
        isFullDay = true;

        // Set hour limit and new hour/minute values
        hour.setLimit(24);
        hour.setCount(h);

        minute.setCount(m);

        // Update display
        display = hour.printValue() + ":" + minute.printValue();
    }

    /**
     * Sets the time in 24-hour format.
     */
```

```

*
 * @param h          Hours (0 - 11)
 * @param m          Minutes (0 - 59)
 * @param newIsAM    True for AM, false for PM
 */
public void setTime(int h, int m, boolean newIsAM)
{
    isFullDay = false;
    isAM = newIsAM;

    // Set hour limit and new hour/minute values
    hour.setLimit(12);
    hour.setCount(h);

    minute.setCount(m);

    // Update basic display
    display = hour.printValue() + ":" + minute.printValue();

    // Concatenate appropriate AM/PM postfix
    if (isAM)
        display = display + " AM";
    else
        display = display + " PM";
}

/**
 * Ticks the clock, i.e. advances time by one minute.
 */
public void tick()
{
    // Increment minutes
    minute.increment();

    // If minutes were reset, hours should also increment
    if (minute.getCount() == 0)
        hour.increment();

    // Update basic display
    display = hour.printValue () + ":" + minute.printValue();

    // If it's not full day, take care of AM/PM
    if (!isFullDay)
    {
        // If time was reset to 00:00, invert AM/PM and PM/AM
        if ((hour.getCount() == 0) && (minute.getCount() == 0))
            isAM = !isAM;

        // Concatenate appropriate AM/PM postfix
        if (isAM)
            display = display + " AM";
        else
            display = display + " PM";
    }
}

```

```
    }  
}  
}
```

