Systems Programming

Bachelor in Telecommunication Technology Engineering
Bachelor in Communication System Engineering
Carlos III University of Madrid

Leganés, 21st of March, 2014.
Duration: **75 min.**

Full name: .....

Signature: .....

Instructions

All the problems contribute the same to the final score.

You are only allowed a black or blue pen (or two), a clock or a watch and the DNI or the Student Identification Card. This means: no pencils, phones, calculators....

Ignore this line: .,..,,.,

Problem 1

Write a Java `Counter` application that counts clicks on a button and resets the count by clicking on another button. The visual appearance and behavior of the application is shown in the figure in the next page.

Initial status

After 5 clicks to the Increment button

After one or more clicks to the Clear button

28 clicks after the the clear

You must take into account the following restrictions:

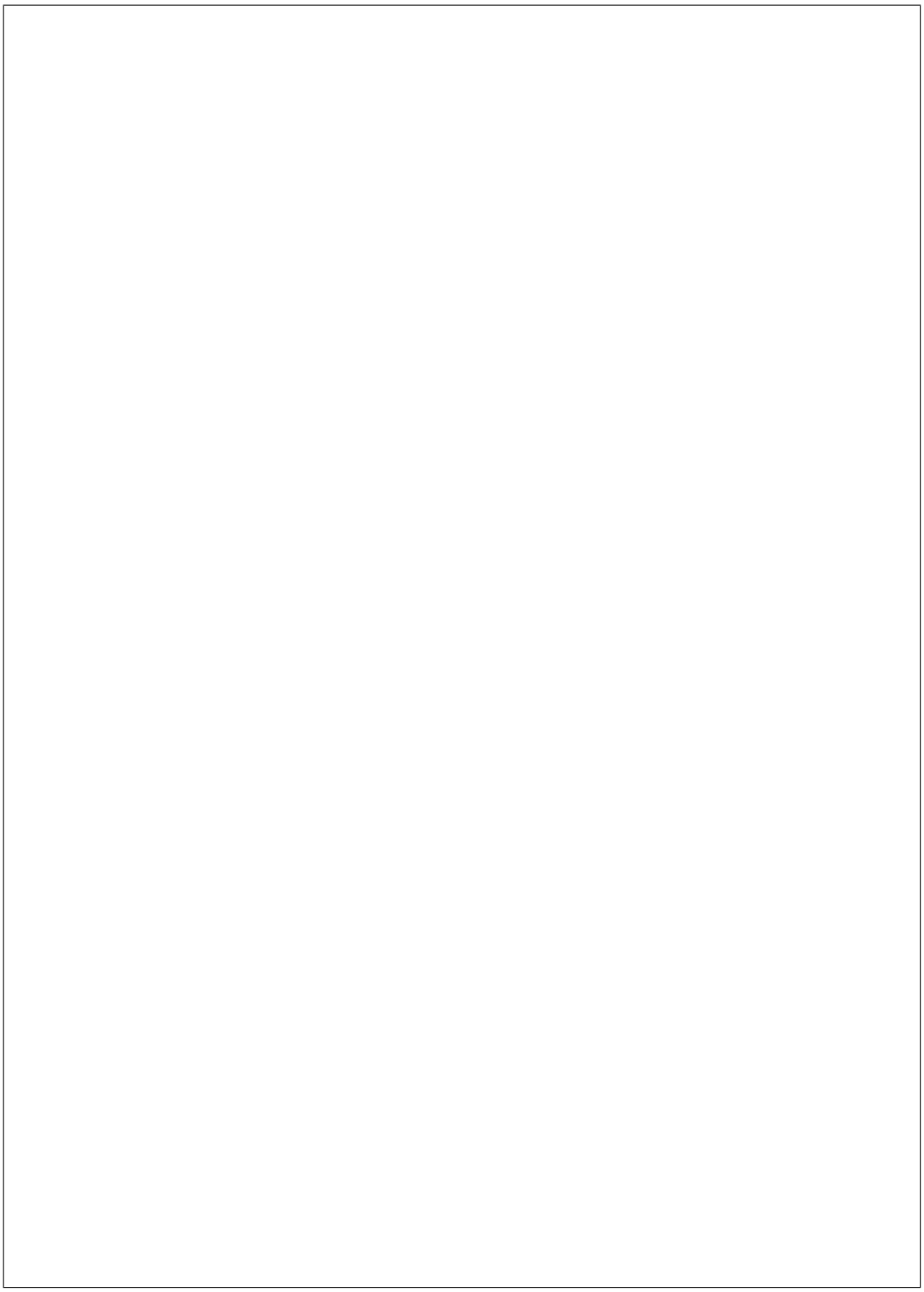
- The visual appearance of your application must be like the one shown in the figure.
- Your solution must compile and run using the following invocations:

```
; javac Counter.java
; java Counter
```

- Your must solve the problem using only one class.
- Such class must inherit from `JFrame` and implement `ActionListener` and you must take maximum advantage of these two properties. This means you must not invoke `new JFrame(...)` on your code and you must not instantiate any new objects as listeners for the buttons.

Points will be awarded for code clarity and readability (20% of the final score).

Answer this problem by filling out the box in the next page.



Problem 2

Consider `PurchaseItem`, a parent class which models customer's purchases. This class must have, at least:

- Two private instance variables `name` (String) and `unitPrice` (double) (i.e. price per unit of the product).
- One constructor to initialize the instance variables.
- A public abstract `getPrice()` method.
- Override `public String toString()` method to return the `name` of the item followed by an '@' symbol, then the `unitPrice`.

Consider `CountedItem`, a subclass with an additional private variable `quantity` of type int.

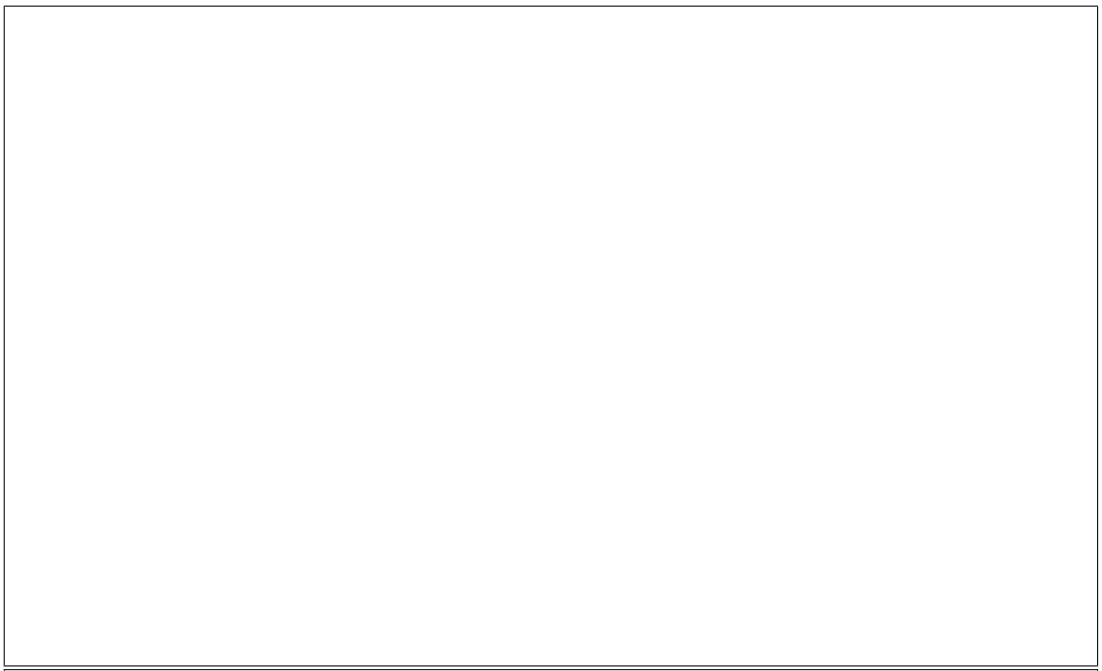
- Write an appropriate constructor for the `CountedItem` class.
- Implement the `getPrice` method of the purchased item based on its unit price and its quantity.
- Override the `toString` method to add the 'x' symbol, the quantity, a '=' symbol and the final price to the original output string.

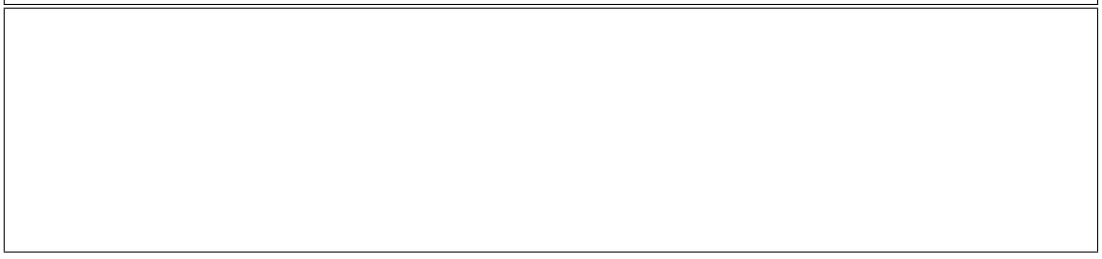
Also write a `Test` class that prints to the standard output a purchase of 2 apples, with a unit cost of 3.25 Euros. The result of executing this test class should be:

```
; java Test
Apple@3.25x2=6.5
```

Code simplicity through reusability will be rewarded, so keep the public API of your classes simple and short.

Answer this problem by filling out the three boxes in the next page (one for each class).





Answer to problem 1

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingConstants;

public class Counter extends JFrame implements ActionListener {

    private JLabel display;
    private int counter;

    public Counter() {

        super("Counter");

        counter = 0;

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton increment = new JButton("Increment");
        increment.addActionListener(this);

        JButton clear = new JButton("Clear");
        clear.addActionListener(this);

        display = new JLabel(Integer.toString(counter), SwingConstants.CENTER);

        add(display, BorderLayout.NORTH);
        add(increment, BorderLayout.CENTER);
        add(clear, BorderLayout.SOUTH);

        pack();
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Increment")) {
            counter++;
        } else {
            counter=0;
        }
        display.setText(Integer.toString(counter));
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    Counter gui = new Counter();
                }
            });
    }
}
```

Evaluation criteria for problem 1

The problem is graded from 0 to 10, being 0 the lowest mark and 10 the highest.

The student mark starts at 0, then the following modifiers are applyed in order:

[C100] (+1) (may be applyed twice) Code clarity and readability. Subjective.

[C102] (+1) The application GUI is visualy correct.

[C103] (+1) Correct class attributes and its initialization (display and counter).

[C104] (+1) Correct inheritance from JFrame and use of super in the constructor.

[C105] (+1) Correct implementation of the ActionListener interface.

[C106] (+1) ActionListener (this) is added to buttons.

[C107] (+1) Correct int to string conversion (counter to label).

[C108] (+1) Correct handling of the two ActionEvent sources (including String comparison).

[C109] (+1) Correct JLabel update on the actionPerformed method.

[C110] (-1) (may be applyed several times) Subjective penalty. Don't consider errors related with the thread safety of SWING objects, as the students don't have enought theorical background on this topic.

Answer to problem 2

```
public abstract class PurchaseItem {
    private String name;
    private double unitPrice;

    public PurchaseItem(String name, double unitPrice) {
        this.name = name;
        this.unitPrice = unitPrice;
    }

    public abstract double getPrice();

    public double getUnitPrice() {
        return unitPrice;
    }

    @Override
    public String toString() {
        return name + '@' + unitPrice;
    }
}
```

```
public class CountedItem extends PurchaseItem {
    private int quantity;

    public CountedItem(String name, double unitPrice, int quantity) {
        super(name, unitPrice);
        this.quantity = quantity;
    }

    @Override
    public double getPrice() {
        return getUnitPrice() * quantity;
    }

    @Override
    public String toString() {
        return super.toString() + 'x' + quantity + '=' + getPrice();
    }
}
```

```
public class Test {
    public static void main(String args[]) {
        System.out.println(new CountedItem("Apple", 3.25, 2));
    }
}
```

Evaluation criteria for problem 2

The problem is graded from 0 to 10, being 0 the lowest mark and 10 the highest.

The student mark starts at 0, then the following modifiers are applyed in order:

[C200] (+1) Correct attributes of the PurchaseItem class

[C201] (+1) Correct constructor of the PurchaseItem class

[C202] (+1) Correct getPrice() of the PurchaseItem class

[C203] (+1) Correct getUnitPrice() of the PurchaseItem class

[C204] (+1) Correct toString() of the PurchaseItem class

[C205] (+1) Correct attributes of the CountedItem class

[C206] (+1) Correct constructor of the CountedItem class

[C207] (+1) Correct getPrice() of the CountedItem class

[C208] (+1) Correct toString() of the CountedItem class

[C209] (+1) Correct Test class

[C210] (-5) CountedItem do not extends from PurchaseItem

[C211] (-1) (may be applyed several times) Subjective penalty. Pay special attention to lack of code reusability or the existence of extra public methods not really needed.

A cascade of errors must count as the sum of all of them, for example, declaring the attributes public and failling to provide a getUnitPrice() will count as [C200] and [C203].