| SHADED logo | INSTITUTE OF TECHNOLOGY TRALEE  SUMMER EXAMINATIONS AY 2012-2013  **Object Oriented Programming 1**  **PROG61002**  **CRN 43830** |
| --- | --- |

**External Examiner**:

**Internal Examiner**: Mr. J. Brosnan

**Duration**: 2 Hours

**Instructions to Candidates:** You may attempt *any* 3 questions. All questions carry equal marks. Show all workings clearly as you may lose marks otherwise.

**Q1.**

**(a)**

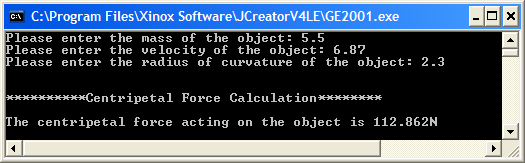
An object undergoing circular motion is said to have a *centripetal force* acting on it which keeps the body moving in a circular fashion. The centripetal force acting on the object can be determined using the formula

where m is the mass of the object, is the velocity of the object and r is the radius of the circle in which the object is moving (its radius of curvature)

Write a Java program that prompts for and reads in the values for the mass of the object, its velocity and the radius of curvature and uses these to determine the centripetal force acting on the object. You can take it here that all values entered could be **fractional** in nature.

Once the centripetal force has been determined, it should be displayed correct to **3 decimal places**, as indicated in the sample screenshot below.

Note that your program should run **exactly** as indicated in the sample screenshot with all formatting and newlines, and the **program code should be complete** with comments (single and multiline), class definition header and main().



**(13.33 marks)**

**(b)**

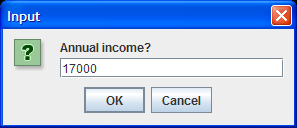
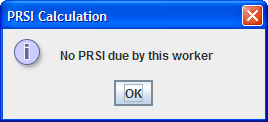
In Ireland most workers are obliged to pay a Pay Related Social Insurance (PRSI) contribution on their earnings. However, any workers earning less than €18304 per year are exempt from paying any PRSI.

Should they earn at least €18304, self-employed persons must pay PRSI at the rate of 3% on all income up to €100100 and at a rate of 2.5% thereafter. So, for example, if a self-employed person earns €120000 per annum then they must pay 3% PRSI on the first €100100 of their income and 2.5% on anything above the €100100. Public-sector employees earning at least €18304 pay PRSI at the rate of 0.9% on all earnings.

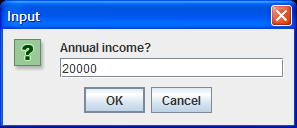
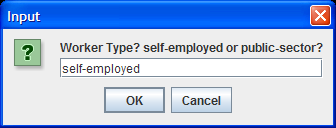
Write a Java program that first of all reads in the annual income of a worker. Should the income be below the €18304 threshold then the program should simply give a message indicating that there is no PRSI due by the worker. If the income is at least €18304, then the program should ask whether the worker is self-employed or public-sector. Then it should use the information outlined earlier to determine how much PRSI the worker needs to pay in a given year. The amount of PRSI due should be displayed to the **nearest whole Euro**.

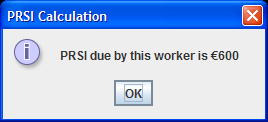
Note that, for brevity, the screenshots below only show some of the possible runs of this program:

**Run 1 – the worker has an annual income of €17000**

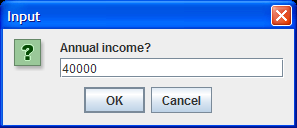
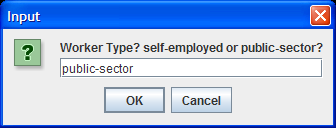
 

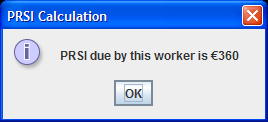
**Run 2 – the worker has an annual income of €20000, and is self-employed**



**Run 3 – the worker has an annual income of €40000, and works in the public-sector**



**(20 marks)**

**Q2.**

The details of an arbitrary number of computers are to be processed by a Java program. The details to be entered are the computers *serial number* (which can contain letters), its *hard disk space* (a **whole number** of GB), its *processor* type and its *price*. When all the computer details have been entered, the user of the program should then enter the empty string (by hitting return on the keyboard) for the serial number to indicate the end of input.

The program should keep track of the total number of computer records entered and use this to display the **average hard disk space** available on the computers to the **nearest whole number**.

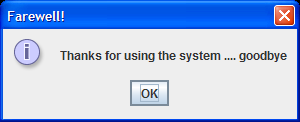
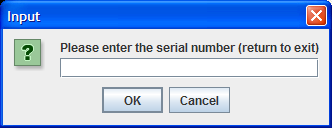
Your program should ensure that the **hard disk space** of the computer is **validated** perfectly, giving the user as many goes as necessary to enter a valid value and not moving on until such a value has been supplied. You can take it here that any hard disk space value *greater than* 50 and *less than* 5000 is valid and that the program should reject any values containing non-digits as indicated in the screenshots below. You may find the **charAt**() and **length**() methods useful in your validation algorithm here.

Also, if the user should enter the empty string for the serial number immediately when the program runs, the program should just terminate at that point and not display any output results at all, issuing a farewell message instead.

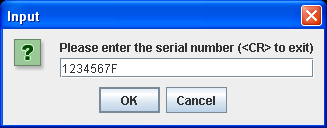
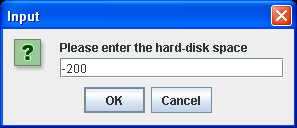
Your program should make use of a **user-defined method** called **averageDiskSpace**() in order to determine and return the average hard disk space value as a float. This method will take two arguments, these being the total disk space for all the computers entered and the number of computers processed.

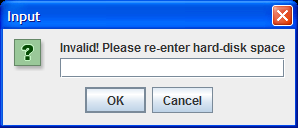
**Sample Screenshots**

**Run 1 – In this case the user hits return immediately when asked for the first serial number, a farewell message is issued and the program terminates when the user hits the OK button**

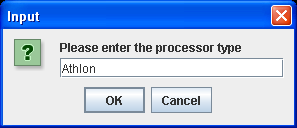
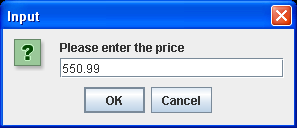


**Run 2 – The user begins by entering a serial number and then several invalid hard-disk space values. These are all rejected in turn until a valid one (3000) is supplied and the user is then prompted for further details.**

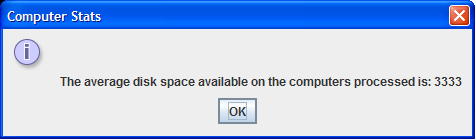
** **

**** 

** **

** **

**Several rounds of input later …… finally followed by hitting return for the serial number gives**



**(33.33 marks)**

**Q3.**

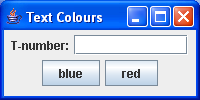
Write a Java program that contains a JFrame window. This window should have a label, a text-field and a 2 buttons. It should have dimensions of 200 x 100 pixels. It should use a flow-layout style and the application should terminate when the close button is hit on its title bar. The label should simply prompt the user for their t-number. The user can then type in their t-number in the text-field. The buttons should be labeled “blue” and “red” respectively. If the user presses the “blue” button then the t-number that has been typed should turn blue in colour and if the user presses the “red” button then the t-number should turn red.

Note that there should be a little **validation** in the application. For example, if the user clicks either button but there is no value in the text-field, a suitable error message should appear to the user.

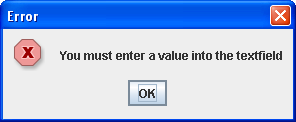
You should make full use of the **appendix** of method definitions at the end of the script when answering this question.

**Sample Screenshots**

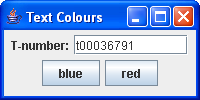
**After launching the application it appears as follows:**



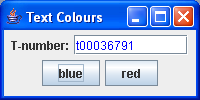
**If the user clicks either button at this point the following message dialog appears:**



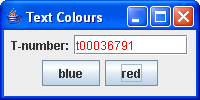
**If the user now types in a tnumber (any text will do really) it appears in the default (black) colour:**



**If the user now presses the blue button the text appears in blue:**



**If the user now presses the red button the text appears in red:**



**(33.33 marks)**

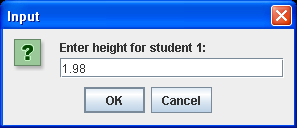
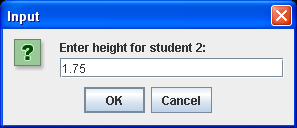
**Q4.**

Write a Java program which makes use of an array of double called heights of size 10, to store the heights of 10 students. Apart from main(), this program should have four **user-defined methods**. The names and purpose of these methods, respectively, are as follows:

1. readInHeights() - prompt for and read in the heights of the 10 students
2. averageHeight() - calculate and return the **average** **height** of the 10 students
3. tallestHeight() - calculate and return the height of the **tallest** student
4. displayOver1\_7m() - display a list of the heights that were above 1.7 metres

When averageHeight() has been called in main(), the value returned by it should be displayed correct to **2 decimal places**.

The program should run as indicated in the following sample screenshots:

**…… 8 more input values follow and then the following message dialogs appear in turn**

**(33.33 marks)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**APPENDIX A**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Feel free to use any of the following methods as necessary in your exam, in particular the GUI program in Q3. All methods are instance methods unless otherwise stated.

**JFrame(String txt)** – creates a JFrame window and places the information described by txt into its title bar

**JTextField(int size)** – creates a text-field component whose size is indicated. The number given is approximately the value of the number of capital “W” characters that can be seen in the text field e.g. 30 indicates that 30 “W” characters could be seen in the component.

**JPasswordField(int size)** – creates a password-field component whose size is indicated. When the user types in the field, only asterisks appear in the field.

**JLabel(String txt)** – creates a label containing the text txt.

**JLabel()** – creates a blank label, often used as placeholding or dummy labels.

**JButton(String txt)** – creates a button containing the text txt.

**JPanel()** – creates a panel which is a sub-container into which GUI components (including other JPanels can be placed)

**JPanel(LayoutManager lm)** – creates a panel sub-container and associates the particular layout manager as indicated by the lm argument with it (FlowLayout, BorderLayout or GridLayout).

**JRadioButton(String txt, boolean selected)** – creates a radio-button and associates the text txt with it. If the button is to appear selected upon creation the value of selected will be true.

**JCheckBox(String txt, boolean selected) -** creates a check-box and associates the text txt with it. If the check-box is to appear selected upon creation the value of selected will be true.

**JComboBox(String[] names) -** creates a combo-box and associates the names contained in the array names with the combo-box so that they appear as the various options available when the component is interacted with.

**void setMaximumRowCount(int rows) –** determines exactly how many rows will appear when the combo-box is interacted with.

**int getSelectedIndex() –** returns an integer which indicates the specific combo-box row (option) that the user selected

**void setIcon(Icon icon) –** allows an icon (including image icon) to be displayed on a component such as a label.

**Font(String type,int style,int point) –** creates a font and gives it an associated type e.g. “TimesRoman”, a particular style e.g. Font.BOLD and a particular size (point).

**ImageIcon(String name) -** creates an image icon and associates the file which stores the actual image, which is referenced by the argument name, with the icon.

**void setSize(int width, int height)** – sets the size of a component, giving it a certain number of pixels in width and height.

**void setVisible(boolean b)** – shows the component on the screen if the argument b is true. Hides it otherwise.

**void addWindowListener(WindowListener winlstner)** – adds the specified window listener to “listen” for events that may be generated by this window

**void addActionListener(ActionListener actlstner)** – adds the specified action listener to “listen” for events that may be generated by this button or text-field

**void addItemListener(ItemListener itlstner)** - adds the specified item listener to “listen” for events that may be generated by this radio-button, checkbox or combo-box.

**void addMouseListener(MouseListener mlstner)** - adds the specified mouse listener to “listen” for events that may be generated by the mouse interacting with a particular component.

**void windowClosing(WindowEvent e)** – method of the class WindowAdapter called when a window is in the process of being closed.

**void actionPerformed(ActionEvent e)** – method of the interface ActionListener called when a button has been pressed or the user hits return on a text-field

**void itemStateChanged(ItemEvent e)** – method of the interface ItemListener called when a radio-button, check-box or combo-box has been selected or deselected.

**int getStateChange() –** method called on an ItemEvent object and returns an integer indicating whether the object triggering the event was selected or deselected. The value can be compared with the constant ItemEvent.SELECTED to determine which.

**void mouseClicked(MouseEvent e)** – method of the interface MouseListener called when the mouse has been clicked on a component

**void mousePressed(MouseEvent e)** – method of the interface MouseListener called when the mouse has been pressed on a component

**void mouseReleased(MouseEvent e)** – method of the interface MouseListener called when the mouse has been released on a component

**void mouseEntered(MouseEvent e)** – method of the interface MouseListener called when the mouse has entered onto the borders of a component

**void mouseExited(MouseEvent e)** – method of the interface MouseListener called when the mouse has exited from the borders of a component

**void setLocation(int x, int y)** – places the component in the location on the screen indicated by the x (horizontal) and y (vertical) coordinates. The default is (0,0) to indicate the top-left of a window.

**Container getContentPane()** – returns a reference to the window pane’s “container” so that other components may be added to it.

**void setLayout(LayoutManager lm)** – takes an object of a class that implements interface LayoutManager as an argument e.g. FlowLayout, BorderLayout or GridLayout and sets the layout style for the container concerned to this value.

**GridLayout(int r,int c)** – creates a GridLayout layout manager object and indicates that it will have r rows and c columns associated with it.

**BorderLayout()** – creates a BorderLayout layout manager object.

**FlowLayout()** - creates a FlowLayout layout manager object.

**void add(Component comp)** – adds the component comp to a particular container.

**void add(Component comp, int pos)** – adds the component comp to a particular container using the BorderLayout layout manager. The argument pos decides the region in which the component will be placed e.g. **BorderLayout.NORTH**

**ButtonGroup()** – creates a ButtonGroup object to which radio-buttons can be added and thereby logically-linked so that if one of the radio-buttons becomes selected, all the others in the group become automatically deselected.

**void add(AbstractButton button)** – a method associated with the ButtonGroup class which allows objects of classes that subclass AbstractButton (such as JRadioButton and JCheckBox) to be added to a ButtonGroup.

**void setAlignment(int alignment)** – called on a FlowLayout object, this will ensure that the default alignment (centred) is overridden via the alignment argument which can be either FlowLayout.LEFT or FlowLayout.RIGHT

**void setLayout(LayoutManager lm)** – called on a Container object, this will set the layout manager associated with the container to that indicated the subclass of LayoutManager passed as an argument e.g. FlowLayout.

**String getText()** – this returns the text associated with a particular component.

**void setText(String txt)** – this sets up the text associated with a particular component with the value contained in txt.

**double random()** – this method returns a pseudo-random value between 0 and up to, but not including, 1. It is a static method from the Math class.

**double pow(double x, double y)** – this method returns the value of x raised to the power of b. It is a static method from the Math class.

**Object getSource()** – this method returns the GUI object which caused an event to be triggered.

**double parseDouble(String num)** – this method returns a double whose value has been parsed from the String value num. It is a static method of the Double class.

**float parseFloat(String num)** – this method returns a float whose value has been parsed from the String value num. It is a static method of the Float class.

**int parseInt(String num)** – this method returns an int whose value has been parsed from the String value num. It is a static method of the Integer class.

**StringBuffer(int sz)** – creates a StringBuffer capable of holding 20 characters.

**int length()** – method that can be called on a StringBuffer or String object to determine the number of characters it currently contains.

**void requestFocus()** – requests that this component gets the input focus.

**char charAt(int p)** – method that can be called on a String object to determine the value of a character at a particular position p within the String, beginning at position 0 for the first character.

**void append(Object obj)** – adds the characters associated with the object obj to the StringBuffer.

**void setBackground(Color c)** – method that can be called on any component to set its background colour to the value of the colour object reference c e.g. **Color.RED** would set the components background colour to red.

**void setForeground(Color c)** – method that can be called on any component to set its foreground colour to the value of the colour object reference c e.g. **Color.RED** would set the components foreground colour to red.

**Color getBackground()** – method that can be called on any component to determine its current background colour.

**char[] getPassword()** – method that can be called on a JPasswordField component to retrieve the text the user typed into it. This can be in turn be converted to a String object by calling the constructor String(char[] password)

**void requestFocus()** – method that can be called on any component to make it the currently active component

**String showInputDialog(Object message)** – a static method of the class JOptionPane that allows a small dialog box to appear which has a text-box in it allowing input to be supplied. You can take it that the argument will be the *message* you wish to issue to the user.

**void showMessageDialog(Component parentComponent, Object message, String title, int messagetype)** – a static method of the class JOptionPane that allows a small dialog box to appear. You can take it that the first argument will be “**null**”, the second one will be the *message* you wish to issue, the third will give the dialog box a *title* in its titlebar and the last will indicate the *messagetype*, which may be one of **JOptionPane.INFORMATION\_MESSAGE, JOptionPane.PLAIN\_MESSAGE,JOptionPane.WARNING\_MESSAGE, JOptionPane.ERROR\_MESSAGE**

**void setDefaultCloseOperation(int action)** – a method of the class JFrame that determines what will happen when the exit button is pressed in the title bar of the window. You can assume here that the action argument will always be **JFrame.EXIT\_ON\_CLOSE**