OLLSCOIL NA hÉIREANN
THE NATIONAL UNIVERSITY OF IRELAND, CORK

COLÁISTE NA hOLLSCOILE, CORCAIGH
UNIVERSITY COLLEGE, CORK

Summer Exam 2011

Second Year Computer Science

CS2500: Software Development

Dr Carron Shankland
Professor J.A. Bowen
Dr M.R.C. van Dongen

INSTRUCTIONS: Answer **all 9** questions for full marks (225). All questions carry equal marks (25). There is **no need** to write whole classes *except where this is explicitly stated*. There is **no need** to write `import` statements. Use meaningful identifier names. Pay attention to the layout of your code and make sure your coding style adheres to the Java coding conventions. There is **no** need to add comments. Table 1 on Page 5 lists some methods, constructors, and constants which may be useful when answering the questions. Note that the table is not exhaustive and may contains errors.

3 hours

Question 1: Basics. *(25/225 marks)*

Question 1.a. *(10 marks)*
Provide a `main( )` method that uses the enhanced for loop to print its arguments to standard output.

Question 1.b. *(15 marks)*
Provide a *class* method that creates a `Scanner` to read from standard input. The method should use the `Scanner` to read all `int`s on standard input and print their sum to standard output.

Question 2: Classes. *(25/225 marks)*

Question 2.a. *(10 marks)*
What is a class and how does it relate to an object?

Question 2.b. *(15 marks)*
Implement a class that generates random numbers. The class should use a `Random` instance variable which is used to compute the next random number. The class should provide an instance method `int nextRandomInt( )` which computes the next random number and a class method `int totalRandomNumbers( )` which returns the total number of random numbers computed by the class. Do not rely on the Java convention that attributes get default values. Instead you should explicitly initialise all variables.

Make sure your class is implemented using good object-oriented and software engineering standards.

Question 3: Inheritance. *(25/225 marks)*

Question 3.a. *(10 marks)*
Explain the notion of inheritance, and state three advantages of inheritance.

Question 3.b. *(10 marks)*
Provide a concrete example of how you may implement inheritance in Java. Explain your example in terms of the notions of 'being more specific' and 'being more general,' and relate these notions to the notions of 'is-a,' 'extension,' 'subclass,' and 'superclass.' *There is no need to provide Java code.*

Question 3.c. *(5 marks)*
Provide an example of overloading and an example of overriding. Explain the difference between overloading and overriding.

Question 4: I/O and Exceptions. *(25/225 marks)*

Implement a method called `copy( )` which copies a source file to a destination file. The names of the source and destination files are provided as arguments to the method. The method `copy( )` should be robust and properly deal with possible errors.

Question 5: Class Design. *(25/225 marks)*

Fota Wildlife Park has several `Animals`. Each `Animal` has *eating*, *roaming*, and *noise* behaviour. An `Animal` eats either grass or meat. Currently Fota has a `Hippo`, a `Lion` (a `Feline Animal`), and a `Wolf` (a `Canine Animal`). *However, they are* expecting *more `Animals`, includeing `Felines` and `Canines`.*

The `Hippo` eats grass and roams alone. The `Feline Animals` roam alone and eat meat. The `Canines` roam in packs and also eat meat.

Model the behaviour of Fota's `Animals` **using abstract classes in the higher levels and concrete classes at the final level of the class hierarchy**. There is no need to implement the `Feline` class. The noise behaviour should be implemented by printing the name of the `Animal` that makes the noise. **Write each class and abstract class on a separate sheet in your answerbook.**

Keep in mind that your class design should support the addition of new `Animals` with as little coding effort as possible.

Question 6: Event Handlers. *(25/225 marks)*

Provide a class `Exam`. The class should have a `main( )` method, which displays a window with two buttons and a label on it. The buttons should be in the top and the bottom area of the frame; the label should be in the centre of the frame. The initial size of the window should be 200×300 pixels. The text of a button should be the number of times the button has been clicked. The text of the label should be the total number of times the buttons have been clicked. For example, if the user clicks the top button once and then clicks the bottom button twice, then the top button should have the text '1', the bottom button should have the text '2', and the label should have the text '3'. *Hint: use an inner class for the buttons.*

Table 1 on Page 5 lists some methods and constants which you may use to answer this question. Notice that the table is not exhaustive and lists some red herrings.

Question 7: Recursion. *(25/225 marks)*

Question 7.a. *(10 marks)*

Explain the notion of recursion.

Question 7.b. *(15 marks)*

Implement a recursive method that multiplies two non-negative `ints`. The method is not allowed to use multiplication. *Hint: use repeated addition. For example,* $2 \times 3 = 3 + 3$ *and* $3 \times 4 = 4 + 4 + 4$. **Note:** No marks are awarded if you provide an iterative solution or multiplication.

Question 8: Generic Classes. *(25/225 marks)*

Implement a generic class that implements a limited form of linked lists. The class should provide a constructor, a method for adding an object to the front of an existing list, and a method that computes the length of a given linked list.

Question 9: Enumerated Types. *(25/225 marks)*

Question 9.a. *(10 marks)*

Provide a critical comparison of `int` enums and Java enums. State advantages and disadvantages.

Question 9.b. *(15 marks)*

Using Java enums, implement a class for payroll computation. The constants in the class correspond to the **normal** days of the week: Monday, Tuesday, ..., Friday, **weekend** days: Saturday and Sunday, and a Bank Holiday, which is a **special** day of the week.

The class should provide an instance method `double pay( double time, double payrate )` which returns the total pay of an employee who has worked on the current day. The rules for computation are as follows:

- The **pay** for the given day given by

$$\text{pay} = \text{base pay} + \text{overtime pay of that day}.$$

- Here **base pay** is given by

$$\text{base pay} = \text{pay rate} \times \text{hours worked}.$$

- The **overtime pay of that day** is given by

$$\text{overtime pay} = \text{pay rate} \times \text{overtime hours}/2.$$

- The **overtime hours** depend on the kind of day.

  Normal weekday: For a normal week day the overtime hours are the hours worked on that day in excess of 8 hours.

  Weekend: For weekend days, the overtime hours are the hours worked on that day.

  Bank holiday: For a bank holiday, the overtime hours are 1.5 times the hours worked on that day.

Make sure your class is maintainable: it should be possible to add and remove days without breaking existing code. *Hint: implement your class using the* strategy enum *pattern.*

Class/Interface	Methods and Constants
InputStreamReader	int read( ) throws IOException
OutputStreamWriter	void write( ) throws IOException
FileReader	FileReader(File file) throws FileNotFoundException
FileWriter	FileWriter(File file) throws IOException
JButton	void addActionListener(ActionListener listener)
JButton	void addChangeListener(ChangeListener listener)
JButton	void doClick( )
JButton	void doClick(int pressTime)
JButton	void fireActionPerformed(ActionEvent event)
JButton	void fireItemStateChanged(ItemEvent event)
JButton	void fireStateChanged( )
JButton	void setText(String text)
JFrame	static int EXIT_ON_CLOSE
JFrame	Container getContentPane( )
JFrame	void setDefaultCloseOperation(int operation)
JFrame	void setSize(int x, int y)
JFrame	void setVisible(boolean visibility)
JFrame	void update(Graphics g)
Component	void add(Component comp)
Component	void addContainerListener(ContainerListener listener)
Component	void addPropertyChangeListener(PropertyChangeListener listener)
Component	void addPropertyChangeListener(String propertyName, PropertyChangeListener listener)
ActionListener	void actionPerformed(ActionEvent event)
ChangeListener	void stateChanged(ChangeEvent event)
ItemListener	void itemStateChanged(ItemEvent event)

Table 1: Useful methods and constants. Some of the JButton methods are inherited from AbstractButton. Some of these methods are not needed: they're red herrings.