OLLSCOIL NA hÉIREANN
THE NATIONAL UNIVERSITY OF IRELAND, CORK

COLÁISTE NA hOLLSCOILE, CORCAIGH
UNIVERSITY COLLEGE, CORK

Autumn Exam 2010

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.

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 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)*
Explain the notions of a *class* and that of an *object*.

Question 2.b. *(5 marks)*
What are *object references*?

Question 2.c. *(10 marks)*
Explain the purpose of *visibility modifiers*. As part of your explanation you should provide an example.

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

Question 3.a. *(10 marks)*
What is *inheritance*? Your answer should use and explain the the notions of 'being more specific' and 'being more general'.

Question 3.b. *(5 marks)*
State three advantages of inheritance.

Question 3.c. *(5 marks)*
What is *overriding*?

Question 3.d. *(5 marks)*
What is *overloading*?

Question 4: 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` (`Feline Animal`), and a `Wolf` (`Canine Animal`). *However, they are* expecting *a* `Dog`, *a* `Tiger`, *a* `Cat`, *and many more* `Feline` *and* `Canine Animals`.

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**. 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 5: Event Handlers. *(25/225 marks)*

Provide a class `ExamButton`. The class should have a `main( )` method which displays a window with a button on in. The initial size of the window should be 200×300 pixels. The button should display the text '`click me`'. When the user clicks the button for the ith time, the background of the button should change to yellow if i is odd and to green if i is even.

Table 1 on Page 7 lists some methods and constants which you may use to answer this question. The names of the colours are not listed in the table: they are defined as class attributes of the `Color` class and have names which obey the `Java` code conventions. Notice that the table is not exhaustive and lists some red herrings.

Question 6: Abstract Classes and Interfaces. *(25/225 marks)*

Question 6.a. *(5 marks)*

What is an *abstract class*?

Question 6.b. *(5 marks)*

What is an *interface*?

Question 6.c. *(5 marks)*

Why are interfaces needed in `Java`? *Hint: this has got to do with the diamond problem.*

Question 6.d. *(10 marks)*

Many interfaces in the `Java` Collections are also implemented in the form of an abstract class. For example, the `Collection` interface is also provided as an `AbstractCollection` abstract class. The same is true for `Set` and `AbstractSet`, for `List` and `AbstractList`, and for `Map` and `AbstractMap`. Provide an explanation why this is useful.

```
public class PartialComparableClass /* FILL IN #1 */ {
    private String thing;

    public PartialComparableClass( String thing ) {
        this.thing = thing;
    }

    /* FILL IN #2 */
}
```

Figure 1: Contrived partial class.

Question 7: Iterators and Generics. *(25/225 marks)*

Question 7.a. *(10 marks)*

Figure 1 depicts a contrived partial class which implements the `Comparable` interface. The only purpose of this contrived class is comparing its instances with a given `String`.

This class can be completed by providing code for the '`/* FILL IN #1 */`' and the '`/* FILL IN #2 */`' in Figure 1. To answer this question you are asked to complete this class. Make sure you clearly indicate which code completes '`/* FILL IN #1 */`' and which code completes '`/* FILL IN #2 */`'.

Question 7.b. *(15 marks)*

Provide a generic class `Pair` for representing pairs of things. The class should provide a constructor, a method for getting the first member of the pair, a method for getting the second member of the pair, a method for setting the first member of the pair, and a method for setting the second member of the pair. The class should be parameterised over two types: one for the first member and one for the second member of the pair.

Question 8: Exceptions. *(25/225 marks)*

Joe is an avid scientist. In his spare time he conducts experiments with his computer. Some of his experiments involve Joe's own `JoeThermometerConnection` class, the purpose of which is to get temperature readings from his outdoor thermometer. The class provides the following constructor and instance methods.

`JoeThermometerConnection( )`: Make a connection with the thermometer.

`double getTemperature( )`: Get the current temperature in degrees Celcius.

`void close( )`: Close the `JoeThermometerConnection` connection.

Joe's thermometer is faulty, which causes errors.

Using proper exception handling, demonstrate how to use Joe's class to (1) make a connection with the thermometer, (2) read and print the current temperature, and (3) close the connection with the thermometer. The exception handling should provide as much details as possible about the cause of failure, and print the stack trace which led to the failure.

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

Using `Java enums`, implement a class for representing drinks and the cost of drinks. The constants in the class correspond to the drinks 'Pepsi Cola', 'Coca Cola', 'Guinness', 'Beamish', 'Strawberry Cordial', and 'Blackcurrant Cordial'. The price of a drink is determined by its type: cola, stout, or cordial. The price of the cola drinks is €2.0, the price of the stouts is €4.0, and the price of the cordials is €0.5.

The class should provide an instance method `double price( )` which returns the price of the drink. Make sure your class is maintainable: it should be possible to add and remove drinks without breaking existing code. *Hint: implement your class using the* strategy enum *pattern.*

Class/Interface	Methods and Constants
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 setBackground(Color bg)
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.