MARKING SCHEME

U08186 Advanced Object-Oriented Programming

Examination Rubric

Examination length: **2 hours.**

Answer **three** questions.

Section A is compulsory and contains one question, worth 40 marks.

Two further questions must be answered from Section B. Each is worth 30 marks.

The total number of marks is 100.

Examination Questions

Section A - Compulsory

Question A1

a) Draw the diagram of the Template Method pattern, including the use of a note to indicate the implementation of an operation where necessary. You do not need to remember the official Gang of Four names for the classes and operations you write but any alternative names you choose should reflect the purpose of these roles and classes. Indicate abstract classes and operations clearly either by using italics or by using the property list {abstract}.

Remember that you can use the forward slash notation thus: /abstractSomething/ to indicate abstract classes and operations.

Answer A1.a

One mark for class containing Template Method. One mark for its definition. One mark for auxiliary methods abstract in superclass. One mark for auxiliary methods abstract in superclass and concrete in subclass.

4 marks

b) What is the Hollywood Principle and where is it being used in this pattern?

Answer A1.b

"Don't call us. We'll call you." In other words, parent classes call their children. (2 marks) Here, `templateMethod` in `AbstractClass` calls `primitiveOperation1` in `ConcreteClass`, for example. (2 marks)

4 marks

c) The Facade pattern reduces coupling between client and subsystem classes. Explain what this means, how it happens, and why it is useful.

Answer A1.c

The client classes depend on very little knowledge (2 marks) of the subsytem classes because all communication is through the Facade class (2 marks). This is useful because the subsystem classes can change without the client classes needing to change. (2 marks)

6 marks

d) The State pattern appears to cause the object that represents the state to change its class. Explain how it does this.

Answer A1.d

There is a Context class that holds a reference to the state object. The state object holds a reference to the Context class as well. It can call a set method to change the state object to which the Context class holds a reference.

4 marks

e) The rest of this question is about the Decorator design pattern.

i) Draw the diagram for the Decorator design pattern, naming the classes Component, Decorator, ConcreteDecoratorA, ConcreteDecoratorB and ConcreteComponent.

Answer A1.e.i

Decorator is a kind of / inherits from Component. (2 marks, 1 mark for just inheritance) Decorator has a / wraps a Component. (2 marks, 1 mark for just aggregation)

4 marks

ii) Imagine that a pizza company serves many different kinds of pizza base, each of which can have one or more kinds of topping. Example pizzas include Pepperoni Deep Pan and Pineapple Thin 'n' Crispy with Extra Cheese. The choice of base and the choice of topping(s) both affect the prize that the customer must pay. Imagine you are asked to write a program that calculates the prize of a pizza using the Decorator pattern. Give the bindings you would use to instantiate the classes and operations of the Decorator pattern to write the program.

Answer A1.e.ii

Bind `operation` to `getPrice` . Bind `Component` to `Pizza` . Generalise `Leaf` to `Leafs` and then bind to `DeepPanPizza` and `ThinNCrispyPizza`. Bind `Decorator` to `Topping` . Bind `Decorators` to `PepperoniTopping` and `ExtraCheeseTopping` and `PineappleTopping` . One mark for each binding statement. One mark for pointing out the generalisation if one binding statement has been missed.

5 marks

iii) Assuming the existence of appropriate constructors, give the code necessary to create a Pineapple Thin 'n' Crispy with Extra Cheese and ask for its price.

Answer A1.e.iii

```
Pizza pizza = new ExtraCheese (new Pineapple (new ThinNCrispyPizza()));
int price = pizza.getPrice();
```

One mark for each use of a constructor and a mark for the call to `getPrice` up to a maximum of 3 marks.

3 marks

iv) The use of polymorphism in the Decorator pattern illustrates the Open-Closed Principle. Explain what this principle means with reference to the Decorator pattern.

Answer A1.e.iv

If you want to create extra pizzas and extra toppings you can do so without modifying the existing code. (2 marks) Just create an extra pizza subclass or an extra topping subclass. (2 marks)

4 marks

v) It is possible, though not advisible, to implement pizza pricing in a procedural manner with just one class, ie not using the Decorator pattern and not using polymorphism either. What would the attributes of this class be and how would the price of the pizza be calculated from it?

Answer A1.e.v

One attribute (eg integer) for the pizza type and one attribute (eg an array of integers) for the toppings. (2 marks) Use an if statement to set the initial price of pizza (2 marks) and a for loop to go through each topping, modifying the price accordingly.

6 marks

[Total 40 marks]

Section B - Answer two questions

Question B1

a) Write an definition in Java of an enumerated type that can be used to represent the four different states of a set of traffic lights.

Answer B1.a

```
enum LightState {red, redamber, green, amber};
```

4 marks

b) Write code that can be used to simulate enumerated types.

Answer B1.b

```
class LightStateWithoutEnum{
    public static final int red=0;
    public static final int redamber=1;
    public static final int green=2;
    public static final int amber=3;
}
```

4 marks

c) Write the code for declaring a light and initialising it to red, for each of the two different implementations.

Answer B1.c

```
LightState l1;
l1 = LightState.red;

int l2;
l2= LightStateWithoutEnum.red;
```

4 marks

d) Imagine that you wish to pass an Array List as a parameter to a method `doSomething` but you want `doSomething` to be flexible enough to accept Array Lists of Strings but also Array Lists of Integers. Assuming that `doSomething` does not return a value and does not take another parameter, give its declaration and say what aspect of generics you are using.

Answer B1.d

```
void doSomething(ArrayList<?> list);
```

Wildcards are being used here (1 mark)

4 marks

e) What feature of Java 5.0 would be used to access an Array List of Integers as though it contained ints?

Answer B1.e

Autounboxing of each Integer to make an int

2 marks

f) Give the code for `doSomething` assuming that its purpose is to print out all the elements of the Array List. Give three different forms of the code as follows:

i) one using the enhanced for loop,

Answer B1.f.i

```
for (Object o: list)
        System.out.println(o);
```

3 marks

ii) one using the `Iterator` interface explicitly

Answer B1.f.ii

```
for (int i=0; i < list.size(); i++)
    System.out.println(list.get(i));
```

3 marks

iii) one indexing the ArrayList with an integer.

Answer B1.f.iii

```
Iterator<?> iter = list.iterator();
while (iter.hasNext())
     System.out.println(iter.next());
```

4 marks

g) What is the relationship between these three different solutions?

Answer B1.g

The third is the behind-the-scenes implementation of the first (1 mark) and the second is unrelated, applicable only to Collections that implement the List interface (1 mark).

2 marks

[Total 30 marks]

Question B2

Consider the implementation of a stack as the class `Stack` in C# supplied in the appendix.

a) Imagine you have asked Microsoft Visual Studio, or some other IDE, to produce unit tests and the following method is produced.

```
public void popTest()
{
    Stack_Accessor target = new Stack_Accessor();
    // TODO: Initialize to an appropriate value
    int expected = 0; // TODO: Initialize to an appropriate value
    int actual;
    actual = target.pop();
    Assert.AreEqual(expected, actual);
    Assert.Inconclusive
      ("Verify the correctness of this test method.");
}
```

Explain what this code does and why the IDE produces it.

Answer B2.a

It gets access to a stack object (1 mark) calls the `pop` method (1 mark) and sees whether the correct result has been returned (1 mark), most likely causing an assertion error (1 mark) and then a second assertion error with the stated error message (1 mark).

5 marks

b) Explain why the IDE produced the second assertion.

Answer B2.b

It reminds the programmer that expected is simply a default value (1 mark)

and even if the test has passed (1 mark), the method has not been tested properly (1 mark).

3 marks

c) Similarly, imagine that the following code has also been produced.

```
public void pushTest()
{
    Stack_Accessor target = new Stack_Accessor();
    // TODO: Initialize to an appropriate value
    int elem = 0; // TODO: Initialize to an appropriate value
    target.push(elem);
    Assert.Inconclusive
      ("A method that does not return a value cannot be verified.");
}
```

Given the message of the second assertion, how should the method `push` be tested?

Answer B2.c

By examining (1 mark) the stack after the `push` operation, either with a pop (1 mark) operation, checking that what was popped is the same as what has just been pushed (1 mark), or better still by examining every element and comparing it to what it should be. (1 mark)

4 marks

d) What are the advantages of unit testing compared to manual testing?

Answer B2.d

Unit testing can be performed in just a fraction of a second (1 mark) in contrast to manual testing and can be repeated (1 mark) when the slightest change is made to a program (1 mark).

3 marks

e) What are the advantages of using a language like Spec# in place of or in conjunction with unit testing?

Answer B2.e

Documentation of intended behaviour of each operation which enhances readability (1 mark). Static ie before execution (1 mark) checking (1 mark) of the proclaimed properties (1 mark). (Maximum of 3 marks.)

3 marks

f) Give four examples of where Spec# annotations could be useful for this class. Include accompanying reasons and syntax where applicable.

Answer B2.f

Any four from the following:

i) The field `elems` could be given a non-nullable type (1 mark) thus: `int[]!` (1 mark)

ii) The methods `pop` and `push` could be given preconditions (1 mark) using the `requires` keyword (1 mark) to ensure that no popping is done from an empty stack or pushing onto a full stack (1 mark)

iii) Array accesses could be checked (1 marks) with keyword `forall` (1 mark) to ensure that out of bounds accesses happen at run time (1 mark)

iv) Using the keyword `invariant` (1 mark), an object invariant (1 mark) could record that `size<SIZE`, to ensure that the unused part of the stack is not used (1 mark)

v) The methods `pop` and `push` could be given post-conditions (1 mark) using the `ensures` keyword (1 mark) that state that the part of the stack beneath the top is untouched or the size has increased/decreased by one (1 mark)

12 marks

[Total 30 marks]

Question B3

Answer the following questions about C#.

a) While maintaining a C# program, you encounter the following code:

```
struct Date
{
    public int day, month, year;
    public Date(int day, int month, int year) {
        this.day = day;
        this.month = month;
        this.year = year;
    }
}
```

What is the difference between making `Date` be a `struct` and making it be a `class`. Explain the advantage of using a `struct` in this case.

Answer B3.a

A `struct` is allocated on the stack (1 mark), whereas a `class` is allocated on the heap (1 mark). This is good because memory does not need to be claimed for the object (1 mark) thereby saving time (1 mark), especially useful for domain concepts like Dates that are likely to have many instances. (1 mark) Can also mention for a mark that assignment between structs is value assignment rather than pointer assignment.

5 marks

b) Explain what properties are and how they are used.

Answer B3.b

Properties are getters (1 mark) and setters (1 mark) of attributes that are used with the dot notation (1 mark) but the given definitions (1 mark) are run instead of the usual record access. (1 mark)

5 marks

c) Define properties for one of the attributes of `Date`.

Answer B3.c

```
public int day { set { day = value; } get { return day; } }
```

2 marks for setters, 2 marks for getters, 1 mark for rest

5 marks

d) Redefine these as read-only auto-implemented properties.

Answer B3.d

```
public int day {get; private set; }
```

Marks as for previous part.

5 marks

e) Consider the following code.

```
int [,] arrs;
arrs = {{1,2},{3,4,5}};
```

i) This code uses rectangular arrays. Explain how you know this and what it means

Answer B3.e.i

The `[,]` notation. (1 mark) Arrays must consist of rows of equal length. (1 marks)

2 marks

ii) Why does the code not compile?

Answer B3.e.ii

Interpreting the inner arrays as rows, one row has length 2 (1 mark) and another row has length 3 (1 mark)

2 marks

iii) Change the code so that it compiles without changing what is to be stored in the array.

Answer B3.e.iii

```
int[][] arrs = new int[2][];
arrs[0] = new int[] {1,2};
arrs[1] = new int[] {3, 4, 5};
```

2 marks for each line.

6 marks

[Total 30 marks]

End of Examination Paper

[All marks = 130]

Appendix A Implementation of a Stack in C#

```
class Stack
{
    int[] elems;
    int size;
    static int SIZE = 10;

    Stack()
    {
        elems = new int[SIZE];
        size = 0;
    }

    int pop()
    {
        size--;
        return elems[size+1];
    }

    void push(int elem)
    {
        elems[size] = elem;
        size++;
    }
}
```