

# The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, SPRING SEMESTER

## OBJECT-ORIENTED METHODS

Time allowed TWO hours

---

*Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced*

### **Answer Question ONE and TWO others**

*Marks available for sections of questions are shown in brackets in the right-hand margin*

*Only silent, self-contained calculators with a single-line display are permitted in this examination.*

*Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.*

*No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.*

***DO NOT turn examination paper over until instructed to do so***

**1 (Compulsory Question)** Consider the following Java classes and interface:

```
public interface IAnInterface
{
    public abstract int AMethod();
}

public class ClassA implements IAnInterface
{
    public int AMethod()
    {
        return 42;
    }
}

public class ClassB implements IAnInterface
{
    private IAnInterface _iface;

    public ClassB(IAnInterface iface)
    {
        _iface = iface;
    }

    public int AMethod()
    {
        return 10 * _iface.AMethod();
    }
}
```

And the following block of code:

```
IAnInterface i, j;
ClassA a = new ClassA();
int retVal;

i = a;
j = new ClassB(i);
i = j;

retVal = i.AMethod();
```

If the above block of code was executed, using the objects defined as above, the value of `retVal` would be:

- (i) Nothing, `i.AMethod` throws an exception since the method is not defined in the interface.
- (ii) 420
- (iii) 10
- (iv) 42
- (v) None of the above.

(4)

(b) Which of the following statements are true about the Singleton Pattern?

- (i) The Singleton Pattern ensures that one and only one instance of a class is created.
- (ii) The Singleton Pattern declares the constructor to be `private`.
- (iii) Classes implementing Singletons should provide a `static` (class) method to provide access to the single instance.

- (iv) Instantiation of the Singleton tends to be delayed until it is first used.
- (v) All of the above. (4)
- (c) Which of the following are **not** principles of good Object-Oriented design?
- (i) Program to an implementation, not an interface.
- (ii) Identify the aspects of your application that vary and separate them from what stays the same.
- (iii) Favour inheritance over composition.
- (iv) Classes should be open for modification
- (v) Strive for loosely coupled designs between objects that interact. (4)
- (d) Which of the following statements is false:
- (i) The **Observer** pattern provides a mechanism for objects (called *observers*) to find out about changes in the state of another object (called the *subject*).
- (ii) The *observer* objects are dependent on the *subject* informing them of changes in the data.
- (iii) The *subject* being observed contains a list of *observer* objects that need to be informed about changes in the *subject*'s state.
- (iv) The list of *observer* objects is fixed at compile-time.
- (v) The **Observer** pattern is an example of a loosely-coupled design. (4)
- (e) The **Decorator** Pattern is a way for adding or replacing functionality to an existing class. Which of the following statements is false:
- (i) The **Decorator** pattern uses inheritance to gain type compatibility.
- (ii) The **Decorator** pattern wraps another object up.
- (iii) The **Decorator** pattern results in an explosion of subclasses.
- (iv) The **Decorator** pattern encourages black-box reuse of components.
- (v) The **Decorator** pattern can dynamically change an object's behaviour. (4)
- (f) Which of the following statements is true about a *mark-sweep* Garbage Collector:
- (i) it uses reference counting to manage object lifetimes.
- (ii) it deallocates the memory for an object as soon as the object goes out of scope.
- (iii) it divides the memory heap, where objects are allocated, into two halves.
- (iv) it visits every object that the program can still access and marks them as 'in-use'. It then visits every object the system has allocated and destroys those that have not been marked as in-use.
- (v) it keeps free space contiguous. (5)

2

- (a) Some Object-Oriented languages require the programmer to manually destroy objects when they are no longer needed so that the memory can be reused for new objects. Describe some of the problems that a programmer might face when trying to decide when it is safe to destroy an object. (5)
- (b) Reference Counting is a technique that can be used to work out when an object can be destroyed. Describe how this technique works and how it solves the problem of knowing when it is safe to destroy an object. (10)
- (c) One problem with Reference Counting is that of circular references. Describe how this problem arises and what the effect is on the objects involved. (5)
- (d) Reference counting can be grafted onto languages such as C++ to help programmers manage the lifetime of an object. Explain why the programmer still needs to take care to ensure that objects get destroyed. (5)

3 The following is a specification for a computer window system. Using this specification produce a simple object-oriented design for the system, indicating clearly the various kinds of relationship between classes and objects (e.g. aggregation, composition, inheritance etc.). [You should limit the detail you give in your design, as being appropriate for the *time available* to answer one third of this exam paper.]

SWIS is a Simple WIndow System that allows programmers to create, move, resize, destroy and otherwise manipulate various types of rectangular window on a computer screen. Two kinds of windows are possible, text windows, and graphics windows. Each of these has a location and size associated with it, and each may or may not have a border. The current focus for output and keyboard input will have a highlighted border; only one window will be highlighted at any one time (do not worry about how input or output happens, just that there is a single window with focus). Text windows will allow strings of text to be displayed within them, their location being specified to the nearest character position. Graphics windows will allow graphics objects to be displayed, including individual pixels, straight-lines, rectangles and circles, each of which may or may not be filled or shadowed or both. Text can also be displayed in a graphics window, here being located to the nearest pixel.

- (a) List all the classes required for SWIS and give a one sentence description of their purpose. (10)
- (b) Draw a UML diagram to illustrate the relationships between the classes in your system. The diagram should also show the major methods and variables used by the system. (If in doubt about the correct symbols used to illustrate each concept, give a key to the symbols you have used in your diagram.) (15)

4

- (a) Describe where you might use the **Strategy** Pattern in an Object-Oriented program and how it works. (5)
- (b) The University of Nottingham Human Resources (HR) Department uses two Java classes to maintain its list of staff. The `StaffRecord` class stores details about a particular member of staff: Name, Age, Address, department etc. The second class `StaffList` stores these records in an array and provides methods to access various records. It also provides a `Sort()` method which sorts them by surname. Please note only that part of the implementation relevant to the question is shown.

The HR department now wants to be able to sort the list by department and by age as well as by surname. Describe how you could use the **Strategy** pattern to implement these changes. You should outline any new classes and interfaces required including their methods. Indicate also any modifications required to the existing classes. Explain how using your modifications clients of `StaffList` can change the way the list is sorted, and, in particular, give an example of how they can get the `StaffList` to be sorted by age. (20)

```
public class StaffRecord
{
    public StaffRecord(String surname, String forename,
                       int age, String department)
    {
        m_surname = surname;
        m_forename = forename;
        m_department = department;
        m_age = age;
    }

    public String GetSurname() { return m_surname; }
    public String GetForename() { return m_forename; }
    public int GetAge() { return m_age; }
    public String GetDepartment() { return m_department; }

    public void Print()
    {
        System.out.println(m_forename + " " + m_surname +
                           " [" + m_age + "] " + m_department);
    }
    private String m_surname;
    private String m_forename;
    private String m_department;
    private int m_age;
}
```

```

public class StaffList
{
    public StaffList()
    {
        m_staff = new StaffRecord[15];
        m_cStaff = 0;
    }

    public void AddStaff(StaffRecord staff)
    {
        m_staff[m_cStaff++] = staff;
    }

    public void Sort()
    {
        /* Use bubble sort to sort list */
        for(int i = m_cStaff-1; i > 0; --i)
            {
                for(int j=0; j < i; ++j)
                    {
                        /* Compare the surnames here */
                        String firstSurname = m_staff[j].GetSurname();
                        String nextSurname = m_staff[j+1].GetSurname();
                        if(firstSurname.compareTo(nextSurname) > 0)
                            {
                                StaffRecord swapVar = m_staff[j];
                                m_staff[j] = m_staff[j+1];
                                m_staff[j+1] = swapVar;
                            }
                    }
            }
    }

    public void PrintStaff()
    {
        for(int i =0; i< m_cStaff; i++)
            {
                m_staff[i].Print();
            }
    }

    private StaffRecord m_staff[];
    private int m_cStaff;

    public static void main(String args[])
    {
        StaffList sl = new StaffList();

        sl.AddStaff(new StaffRecord("Troughton", "Patrick", 67, "Temporal Physics"));
        sl.AddStaff(new StaffRecord("Ollis", "James", 23, "Comp. Sci"));
        sl.AddStaff(new StaffRecord("Brailsford", "David", 63, "Comp. Sci"));
        sl.AddStaff(new StaffRecord("Pertwee", "Jon", 76, "Temporal Mechanics"));
        sl.AddStaff(new StaffRecord("Picard", "Jean-Luc", 76, "Astrophysics"));
        sl.AddStaff(new StaffRecord("Bagley", "Steven", 29, "Comp. Sci"));
        sl.AddStaff(new StaffRecord("Jarre", "Jean-Michel", 59, "Music"));
        sl.AddStaff(new StaffRecord("Brush", "Basil", 48, "Agriculture"));
        sl.Sort();
        sl.PrintStaff();
    }
}

```