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 are worth 30 marks.

The total number of marks is 100.

Examination Questions

Section A - Compulsory

Question A1

a) Answer these questions about the `getInstance` method in the Java implementation of the Singleton pattern.

i) The `getInstance` method can be called even if no Singleton objects have been created. Which language feature of Java makes this possible?

Answer A1.a.i

The `static` keyword applied as a modifier to the `getInstance` method.

2 marks

ii) What benefit is conferred by the keyword `synchronized`?

Answer A1.a.ii

If multiple threads are running and they each call the `getInstance` method, the first invocation of `getInstance` will complete before the second invocation starts, even in the presence of interleaving. This prevents both threads seeing the instance is null and each creating a Singleton object.

4 marks

iii) The `getInstance` method is the only one that can create an instance of the Singleton object. How does the method ensure that it does not create a second Singleton object the second time it is called?

Answer A1.a.iii

A field called instance, initially `null`, is set to contain the Singleton object when it is created the first time. Before creating each instance (and therefore before creating the second instance too) this field is checked to see if it is `null`. If it is not `null`, an instance has clearly already been created.

4 marks

b) A book website allows its users to select books from a catalogue and then pay for them. Each transaction is represented as a collection of objects of class `Order` where `Order` is defined as follows:

```
class Order {
   Book book;
   int quantity;

   public Book getBook() {
      return book;
   }

   public int getQuantity() {
      return quantity;
   }
}
```

However, the billing class has been rewritten and it now expects to process a collection of objects of class `OrderDetails`, which is like `Order` except that there is an extra field `cost` of type `int`, which holds the cost in pennies of the order. Explain how the Adapter pattern could be used to help the billing class to process a collection of objects of class `Order` as well. You may assume the `Book` class has a method `getPrice` which returns the price of the book, again as an `int`, representing pennies. Include a diagram as part of your answer and the necessary code for the pattern.

Answer A1.b

Let `OrderDetails` implement an interface, `OrderDetailsInterface`. Then the billing class should be made to expect a collection of objects of class `OrderDetailsInterface`, a minimal change. A new class `OrderDetailsAdapter` must be created wrapping the `Order` class, as follows:

```
class OrderAdapter implements OrderDetailsInterface {

  private Order order;

  public Book getBook(){
     return order.getBook();
  }

  public int getQuantity(){
     return order.getQuantity();
  }
```

```
    public int getCost(){
       return order.getQuantity() * order.getBook.getPrice();
    }

  }
```

8 marks

c) The Strategy pattern is considered to be an example of the design principle "prefer composition over inheritance". Explain how exactly Strategy exemplifies this principle and outline the advantages of it in this case.

Answer A1.c

The Strategy pattern puts a method with several different implementations into a separate class called `Strategy`, with different subclasses for each implementation. The Strategy object is an instance variable of the object of class `Context`, so composition is being used. The implementation associated with the Context object can therefore change at run-time. If inheritance was used within the `Strategy` class, the implementation would be fixed at compile time.

7 marks

d) The class diagram for the Strategy pattern looks almost identical to that for the State pattern. Explain this in terms of the similar behaviour of these patterns.

Answer A1.d

Both have a Context class, with messages sent to that delegated to a composed object implementing an interface `State` or `Strategy`, as appropriate.

3 marks

e) The intent of the State pattern in the Gang of Four book is as follows: "Allow an object to alter its behaviour when its internal state changes. The object will appear to change its class". Explain how this latter characteristic is achieved in Java implementations.

Answer A1.e

The object of class implementing `State` sends, in each of its methods, a message to the Context class, asking it to change the object it wraps to another object also implementing `State`.

3 marks

f) Suppose a class C has two subclasses C1 and C2. Show, with both diagrams and code, how the Factory Method pattern can be used to let subclasses decide which subclass to instantiate. Include all the code for the factory classes.

Answer A1.f

```
interface Factory {
  C createC();
}
public class C1 extends C {
  public C createC() {
     return new C1();
  }
}

public class C2 extends C {
  public C createC() {
     return new C2();
  }
}
```

The diagram is worth 3 marks and it should show both class hierarchies.

9 marks

Total marks 40

Section B - Answer two questions

Question B1

a) `HashMap` is a generic class. What does this mean?

Answer B1.a

It means that there is a separate `HashMap` class for each possible pair of class names representing the classes of the keys and values.

3 marks

b) What is autoboxing?

Answer B1.b

Autoboxing is the automatic conversion of a primitive type, such as `int`, to its reference equivalent, such as `Integer`.

4 marks

c) Explain how autoboxing and autounboxing could be useful when trying to store and retrieve numbers in an associative array implemented using the generic `HashMap` class. Explain your answer with particular method calls and particular types.

Answer B1.c

If you store an association in a `HashMap<Integer, Integer>` and the two items being linked are `int` primitives then autoboxing will convert them to `Integer` objects, prior to the `put` method inserting them. If you lookup an `int` primitive with the `get` method, then autoboxing will convert it to an `Integer` object, yet again, and autounboxing will convert the `Integer` object returnned to an `int` primitive.

5 marks

d) Recent versions of the Netbeans IDE will often suggest that the line

```
@Override
```

be added to a method you have just written.

i) In what precise circumstances does this happen? Do not assume understanding of the term “override” in your answer.

Answer B1.d.i

It will happen if the method you have just written matches in signature with another method of the same name in a superclass of the class in which you are writing the method.

3 marks

ii) What is this language feature called, and why is it useful?

Answer B1.d.ii

The feature is called annotations and it is useful because an IDE can thereby find out that a method overrides one in a superclass even if the source code for the latter is not available.

4 marks

e) Leaving aside the benefits of autoboxing and autounboxing mentioned above, what other advantages does the generic `HashMap` class have over the non-generic `HashMap` class seen in older versions of Java?

Answer B1.e

Objects returned by `get`, for example, can immediately be used without having to be cast first. And casting can cause an exception if an item of the wrong sort was stored in a non-generic `HashMap`.

4 marks

f) Without using the `toString` method of the `HashMap` class, write the code needed to display all the pairings stored in the `HashMap` you considered in c). Assume `toString` is implemented for both keys and values. Use an enhanced `for` loop to iterate through all the keys, which can be obtained as a Collection using the method `keySet`.

Answer B1.f

```
for (Integer i: h.keySet()) {
   System.out.println(i + "==" + h.get(i));
}
```

7 marks

Total marks 30

Question B2

a) Consider this method declaration:

```
static String EnglishNameOfDay(int day) {
  String dayName;
  switch(day) {
    case 1: dayName= "Monday";
    case 2: dayName= "Tuesday";
    case 3: dayName= "Wednesday";
    case 4: dayName= "Thursday";
    case 5: dayName= "Friday";
    case 6: dayName= "Saturday";
    case 7: dayName= "Sunday";
  }
  return dayName;
}
```

i) Assuming that this is written in Java, explain what happens when this program fragment is executed with the value of the parameter `day` being 5.

Answer B2.a.i

It drops through from case 5 into case 6 , then reaches case 7 and thus returns "`Sunday`".

3 marks

ii) Explain how this could not happen if the language was C#.

Answer B2.a.ii

Answer: Absence of break, or return or goto leads to syntax error in C#.

3 marks

iii) Show how you would correct this fragment if it was written in C#.

Answer B2.a.iii

Add `break;` or `return dayName;` in each case (optional for last).

4 marks

iv) Write a C# method `NumberOfDay` that takes the English name of a day (as in the above example) and which returns the corresponding `int dayNumber` (1 to 7). You may assume that only valid day names will be submitted to your method.

Answer B2.a.iv

```
static int NumberOfDay(String dayName) {
  int dayNum;
  switch(dayName) {
    case "Monday"; dayNum= 1; break;
    case "Tuesday "; dayNum= 2; break;
    case "Wednesday "; dayNum= 3; break;
    case "Thursday "; dayNum= 4; break;
    case "Friday "; dayNum= 5; break;
    case "Saturday "; dayNum= 6; break;
    case "Sunday "; dayNum= 7; break;
  }
  return dayNum;
}
```

5 marks

b) The C# programming language contains a feature called an “indexer”. For example, here is the declaration of an indexer for a file:

```
class File {
  private FileStream s;
  public int this [int offset] {
    get {s.Seek(offset, SeekOrigin.Begin);
         return s.ReadByte();
    }
    set {s.Seek(offset, SeekOrigin,Begin);
         s.WriteByte((byte)value);
    }
  }
}
```

This indexer can be used as follows:

```
  File f= new File();
  int x= f[10];
  f[10]= 'A';
```

i) For the class `File` as above, write the getter and setter methods that you would need to access the `i`th element of the file if you did not use the indexer approach and properties. (Note: This is the same as it would be in Java).

Answer B2.b.i

```
public byte Get(int i) {
  s.Seek(offset, SeekOrigin.Begin);
  return s.ReadByte();
}

public void Set(int i, int value) {
  s.Seek(offset, SeekOrigin,Begin);
  s.WriteByte((byte)value);
}
```

6 marks

ii) Briefly explain the advantages of using indexers in C#.

Answer B2.b.ii

The syntax is arguably "nicer" as it uses a notational convention that all programmers have encountered. Indeed, we can index all sorts of sequences using the familiar square-brackets notation.

4 marks

iii) Write a statement using a C# `foreach` statement that will write out the file by calling `Console.Write` on each `int` element of the file `f` in turn.

Answer B2.b.iii

```
foreach(int i in f) Console.Write(i);
```

5 marks

Total marks 30

Question B3

Given the declarations:

```
public static bool IsDate(int y, int m, int d);
// returns true if and only if y, m, d form a valid date

public static int DaysEarlier(int y1, int m1, int d1,
  int y2, int m2, int d2);
// if y1, m1, d1 and y2, m2, d2 both form dates then
// returns the number of days by which y1, m1, d1 is
// earlier than y2, m2, d2, otherwise returns 9999.
```

a) Given two dates, target and today, whose components are held in `int` variables `targetY`, `targetM`, `targetD` and `todayY`, `todayM`, and `todayD` respectively, write a C# program fragment containing a call to `DaysEarlier`, to find out by how many days today is earlier than target (the result will be negative if target is before today). Pay particular attention to how you would detect invalid date values in either `target` or `today`.

Answer B3.a

```
int days= DaysEarlier(todayY, todayM, todayD, targetY, targetM, ta
if(days == 9999) Console.Write("One or both dates not valid");
```

6 marks

b) The approach of returning a special value in error cases (here 9999) is an example of "defensive" programming. Give two reasons why it is not a good idea in this example.

Answer B3.b

All too easy for two valid dates to be 9999 days apart. Also, it is easy to forget to check that an error value has been returned so that a more correct response can be given.

4 marks

c) By referring to this example or others, explain the principle of "programming by contract" and contrast it with defensive programming.

Answer B3.c

Contract makes explicit obligations on client (requires) and on supplier (ensures), for the date inputs, whereas defensive programmer has both such obligations implicit.

6 marks

d) Show the contract for `DaysEarlier`.

Answer B3.d

```
requires IsDate(y1, m1, d1) && IsDate(y2, m2, d2)
ensures result is the difference in dates between d1/m1/y1 and d2/
```

6 marks

e) Write a program fragment in the style of programming by contract to perform the same function as in part (a) but calling your method. Pay particular attention to how you would now deal with invalid dates.

Answer B3.e

```
if (IsDate(todayY, todayM, todayD)) && IsDate(targetY, targetM, ta
  int days= DaysEarlier(todayY, todayM, todayD, targetY, targetM,
else
  Console.Write("One or both dates not valid");
```

8 marks

Total marks 30

End of Examination Paper

[All marks = 130]