BIRKBECK

(University of London)

BSc EXAMINATION FOR INTERNAL STUDENTS

DEPARTMENT OF COMPUTER SCIENCE
AND INFORMATION SYSTEMS

Software and Programming III

BUCI056H6

CREDIT VALUE: 15 credits

DATE OF EXAMINATION: Monday, 10th June 2019
TIME OF EXAMINATION: 09:30am
DURATION OF PAPER: THREE HOURS

1. Candidates should attempt ALL questions in the paper.
2. The number of marks varies from question to question.
3. You are advised to look through the entire examination paper before getting started, in order to plan your strategy.
4. Simplicity and clarity of expression in your answers is important.
5. You may answer questions using only the JAVA programming language unless specified otherwise.
6. You should avoid the use of mutable state or mutable collections in your solutions whenever possible.
7. Start each question on a new page.

Question:	1	2	3	4	5	6	7	8	9	10	Total
Marks:	8	10	10	9	15	16	7	10	9	6	100

Question 1 .... Total: *8 marks*

(a) What is the difference between an *is-a* relationship and a *has-a* relationship? Provide examples to illustrate your answer. 4 marks

(b) Why is it desirable to avoid large inheritance hierarchies? 2 marks

(c) "Familiarity sometimes keeps us from seeing the obvious." 2 marks
In what ways can design patterns help avoid this?

Question 2 .... Total: *10 marks*

(a) State three of the key features of object-oriented programming? 6 marks
Explain the advantages these features provide, illustrating your answer with appropriate examples.

(b) Briefly outline how modularisation has affected Java. You should include appropriate examples to illustrate your answer. 4 marks

Question 3 .... Total: *10 marks*

For each of the following design patterns state its objective illustrating each answer with an appropriate example.

(a) The Façade pattern. 2 marks

(b) The Adapter pattern. 2 marks

(c) The Strategy pattern. 2 marks

(d) The Bridge pattern. 2 marks

(e) The Abstract Factory pattern. 2 marks

Question 4 .... Total: *9 marks*

For each of the following code fragments or descriptions, state the name of the design pattern that is the best match to the description or to the result of the given code; you should justify your answer.

(a) 2 marks

```
CurrencyConverter cc1 = CurrencyConverter.getInstance();
CurrencyConverter cc2 = CurrencyConverter.getInstance();
assert cc1 == cc2; // always passes
```

(b) We have an existing class that counts coins and returns the total amount in GBP (UK pounds). A developer is writing a new class that uses the existing one to do the work but needs answers in US dollars. 2 marks

(c) A method in the cash dispenser software at the airport does currency conversions by forwarding the data to an identical method running on a computer at the bank's data centre where the calculations are done and results returned to the airport computer. 2 marks

(d) 1 mark

```
CurrencyConverter cc = CurrencyConverter.incomingCurrency("USD")
                                        .outGoingCurrency("GBP") // UK pounds
                                        .build();
```

(e) A method in a graphical user interface is called whenever a button on the screen is clicked because the method has previously been registered with the button to be notified whenever a click occurs. 2 marks

Question 5..Total: *15 marks*

Consider the following implementation of a `Tree`:

```
package tree;

public interface Tree {
}

abstract class Node implements Tree {
    public Node(Tree left, Integer elem, Tree right) {
    }
}

abstract class Leaf implements Tree {
}
```

Write a recursive function determining both the *minimum* and *maximum* among the elements of a tree. You should use *de-structuring* techniques in your answer.

The function should accept a `Tree` and return a pair of integers which should respectively be the smallest, and greatest, elements of the tree. The function takes a `Tree` as a parameter to ensure the tree has at least one element.

```
package tree;

public class Compute {
    public static Pair computeMinMax(Tree b) {
        // TODO
    }
}

class Pair {
    private Integer a;
    private Integer b;

    public Pair(Integer first, Integer second) {
        this.a = first;
        this.b = second;
    }
}
```

Note: You should make no assumption about the tree structure, in particular, do not assume it is a binary search tree.

Hint: You might find the following functions useful:

- `Math.min(a, b)`
- `Math.max(a, b)`

Question 6 .... Total: *16 marks*

Suppose we have an application where we consume an `EmailService` to send emails. We could implement this service as:

```
package email;

public class EmailService {
    public void sendEmail(String message, String receiver) {
        System.out.println("Email send to " + receiver + " with message=" + message);
    }
}
```

The `EmailService` class holds the logic to send an email message to the recipient email address. Our application code might be:

```
package email;

public class MyApplication {
    private EmailService email;

    {
        email = new EmailService();
    }

    public MyApplication(MessageService ms){}

    public void processMessages(String msg, String rec) {
        // do some msg validation, manipulation logic, etc.
        email.sendEmail(msg, rec);
    }
}
```

Our client code that will use `MyApplication` to send email messages might be:

```
package email;

public class MyDIClient {
  public static void main(String[] args) {
    MyApplication myApp = new MyApplication(new EmailServiceImpl());
    myApp.processMessages("Hi Fred", "fred@dcs.bbk.ac.uk");
  }
}
```

(a) At first glance there appears to be nothing wrong with the implementation shown above (it compiles and runs!) but the code has certain limitations. Briefly describe each of these limitations. **6 marks**

(b) Use dependency injection to solve the problems with the implementation shown above. Your code will need to adhere to the following requirements: **10 marks**

- Service components should be designed with a base class or an interface.
- Consumers should be written in terms of the service interface.
- Provide an injector class that will initialise the services and then the consumers.

Question 7 .... Total: *7 marks*

(a) What are the key elements in the description of a design pattern? $\boxed{\text{4 marks}}$

(b) State three reasons for studying design patterns? $\boxed{\text{3 marks}}$

Question 8 .... Total: *10 marks*

Briefly explain each of the SOLID principles using appropriate examples to illustrate your answer.

Question 9 .... Total: *9 marks*

The reflection API allows an executing JAVA program to examine or "introspect" upon itself.

Write a method which uses reflection to find out (and output) which methods are defined within a class. You should include the formal parameter and return types in your output together with any checked exceptions the methods may throw.

Question 10 .... Total: *6 marks*

(a) *Test Driven Development* is a strategy where the tests for a module or function are always written before the actual code. State a main reason why this is a useful strategy other than "it ensures that the tests will get written". $\boxed{\text{3 marks}}$

(b) A guideline for testing and debugging is that once a test is found that reproduces a bug, that test must be added to the test suite and retained forever. Why? What value is that test after the bug is fixed? $\boxed{\text{3 marks}}$