Mock Remote Assessment Four

BIRKBECK (University of London)

Software Design and Programming - COIY062H7

Software and Programming III - BUCI056H6

Duration - Three hours

1. Candidates should attempt **ALL** questions on 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 to plan your strategy.
4. Simplicity and clarity of expression in your answers is important.
5. You may answer questions using only the `C#` programming language.
6. You should avoid the use of mutable state or mutable collections in your solutions whenever possible.
7. You should use the solution and project files that are provided.
   Each "text" solution can be provided in a plain text or pdf file within the appropriate question folder.
8. If you have any issues then please raise them in the `Slack` channel.

---

1. **[8 marks]** What are the commonalities and differences of the following:
   - (a) The Adapter and the Proxy design patterns?
   - (b) The Mediator and the Facade design patterns?

   Provide appropriate examples to support your answers.
2. **[6 marks]**
   One way to extend a software system is to find a suitable superclass, and to subclass it. You can achieve a similar effect, without extending the class, by copying some of the operations of one class into a new second class and then delegating calls to an enclosed instance of the first class. Both of these extension techniques, however, require that you know at compile time what behaviour you want to add.
   What are the two techniques that we have just described?
   Indicate clearly “which is which”.
3. **[10 marks]** Discuss what you understand by the following statement:

   > "When designing an object-oriented application, a major tenet of design is *loose coupling*"

What is the role of the `new` operator in this context?
How does *dependency injection* assist with loose coupling?
Provide appropriate examples to support your answer.

4. **[12 marks]**
   Write a method which uses reflection to find out (and output) which methods are defined for a class. You should include the formal parameters and return types in your output, together with any accessibility modifiers.

5. **[12 marks]**
   The function `Evens` accepts an array of integers as its formal parameter type and returns the number of even values in that array. For the following questions you may wish to make use of `LINQ`.

   1. Write a recursive version of `Evens`.
   2. Write a version of `Evens` that uses a higher-order method.
   3. Write a version that uses the equivalent of a filter higher-order method and a regular method.
   4. Write a version that uses the equivalent of a map higher-order method and a regular method.

6. **[12 marks]**
   "The SOLID principles of object-orientation improve the mainatainability of software."

   Discuss this statement with reference to each of the principles and provide appropriate examples to support your answer.

7. **[12 marks]**
   Please note: You should not use any of the examples given in class when answering the following questions.

   1. The Strategy and Command patterns both suggest using objects in place of methods. By the use of appropriate examples explain what the difference is be- tween these two patterns?
   2. The Composite pattern and the Decorator pattern are similar. Describe the differences between the two and give examples of when you would use each.
   3. If you need to add behaviour to an object without changing its class which design patterns might you use?
   4. You are required to connect two parallel class hierarchies by letting subclasses in one hierarchy determine which class to instantiate in the second hierarchy. Which software design pattern should you use and why?

8. **[12 marks]**
   Using the following definition of a `BigInt`, write a `BigInt` method, `Factor`, that returns a `BigInt` that is its smallest factor greater than 1. For example, if `Factor` is applied to a `BigInt` representing `12`, it returns a `BigInt` representing `2`.

   - The method `Factor` can assume that its `BigInt` is not in the error state.
   - Your code does not have to be efficient.

   Please note: you do not (and should not) complete the definitions in `BigInt`.

   ```
   public class BigInt
   {
   ```

```
  private int[] digits; // stores BigInt as an array of Int
  private bool error; // if overflow or underflow occurred, then true else
false

  public BigInt(int x=0) // This class deals only with integers >= 0
  {
    digits = new int[x];
  }

  public BigInt(BigInt x) : this(x.digits.Length)
  {
    // ...
  }

  public BigInt(int[] arr) : this(arr.Length)
  {
    // ...
  }

  // Return a String representation of a BigInteger
  public string OutputBigInteger()
  {
    // ...
  }

  // Add x to this, put return new value
  public BigInt AddBigIntegers(BigInt x)
  {
    // ...
  }

  // Subtract x from this, return new value
  public BigInt SubtractBigIntegers(BigInt x)
  {
    // ...
  }

  // Multiply this by x, return new value
  public BigInt MultiplyBigIntegers(BigInt x)
  {
    // ...
  }

  // Divide this by x, return new value
  public BigInt DivideBigIntegers(BigInt x)
  {
    // ...
  }
  // return the remainder of this when divided by val (this is unchanged)
  public BigInt ModulusBigIntegers(BigInt x)
  {
    // ...
  }

  // true when this == x
  public bool IsEqualTo(BigInt x)
  {
    // ...
  }

  // true when this < x
  public bool IsLessThan(BigInt x)
  {
    // ...
  }
}
```

9. **[16 marks]**
   In this question you will implement a recursive function which evaluates Fibonacci numbers. You will then improve its performance by caching its results using *memoisation*.

   1. Write a recursive method `Fib` which, given an integer *n* computes and returns the *n* − *th* Fibonacci number.

      ```
      Func<int,int> Fib(int n)
      ```

      The first two Fibonacci numbers are one. Every other Fibonacci number is the sum of the previous two Fibonacci numbers. Use recursion in your implementation.

   2. Our Fibonacci method works, but it has a problem — it is very slow for larger numbers! If we take a look at the recursive calls, we can notice that some Fibonacci numbers are computed more than once.
      In fact, the number of recursive calls is exponential in the argument n, and this is due to the fact that we call the method `Fib` multiple times with the same arguments. If we could somehow remember that we are already evaluated a certain tuple, we could use the solution we have computed earlier and avoid computing it twice.

      Implement the method `Memo` which takes a function `f` and returns its memoised version. The memoised function retains internally a mutable `Dictionary` that maps function arguments to return values. Each time a memoised function is applied to some argument it uses the `Dictionary` to check if it was previously applied to that argument. If it was, it returns the return value associated with that argument. Otherwise, it uses the original function `f` to compute the return value and returns it. For example, the following snippet:

```
var mf = Memo(fib);
mf(10);
mf(10);
```

will not recompute all the Fibonacci numbers up to 10 the second time `mf` is called.

END OF PAPER