Mock3

1. **[6 marks]** Demonstrate that *named* and *default* arguments can be used with methods. Create a class `Box` that takes two class arguments: a `string` for the `name` and a `double` for `price`.

   Add a method `CostOfTheBox` which has named arguments for `grocery` ,(`bool`), `medication` (`bool`), and `taxRate` (`double`). Default `grocery` and `medication` to `false`, and `taxRate` to `0.10`.

   In this scenario, groceries and medications are not taxable. Return the total cost of the item by calculating the appropriate tax. Your code should satisfy the following:

```
var flour = new Box(name="flour", 4);
Debug.Assert(flour.CostOfTheBox(grocery=true) == 4);
var sunscreen = new Box(name="sunscreen", 3);
Debug.Assert(sunscreen.CostOfTheBox() == 3.3);
var tv = new Box(name="television", 500);
Debug.Assert(tv.CostOfTheBox(rate = 0.06) == 530);
```

5. **[10 marks]** By use of appropriate code examples illustrate how dependency injection can aid the flexibility of your code.

7. **[20 marks]** A refrigerator has a motor, a temperature sensor, a light, and a door. The motor turns on and off primarily as prescribed by the temperature sensor. However, the motor stops when the door is opened. The motor restarts when the door is closed assuming the temperature is too high. The light is turned on when the door is opened and is turned off when the door is closed.

   (a) Identify the classes in the preceding description of a refrigerator, (b) identify the methods for each of your classes, and (c) Provide implementations for each class in `C#`.

8. **[20 marks]** Implement an inheritance hierarchy for the different types of employees at a company. An abstract base class will represent a generic employee. The three derived classes represent the various types of concrete employees: programmers, managers, and interns. Each class has some fields, a constructor, and a `ToString` method.

   i. Code a class to represent an `Employee`. Include two protected fields: a string for the name and a double for the salary.
   ii. Derive a `Programmer` class from `Employee`. A protected double field to store the average overtime worked by the programmer.
   iii. Derive a `Manager` class from `Employee`. Add a protected string field to store the name of the manager's assistant.
   iv. Derive an `Intern` class from `Employee`. Add a protected integer field to store the number of months of the internship. Provide a public constructor that sets the name, salary, and internship length.