# SIT232 Object-Oriented Development Exam Solution – Trimester 1, 2016

*This solution covers Section B (short answer) of the examination paper only. Section A (multiple choice) is drawn from a large pool of questions which are marked automatically by Scanning within DSA. To maintain the value of the pool of multiple choice questions, no reproduction of those questions or answers is made available to students.*

*Note that all answers provided are only for background, it is not expected that a student will have the exact same answer, or necessarily the same level of detail.*

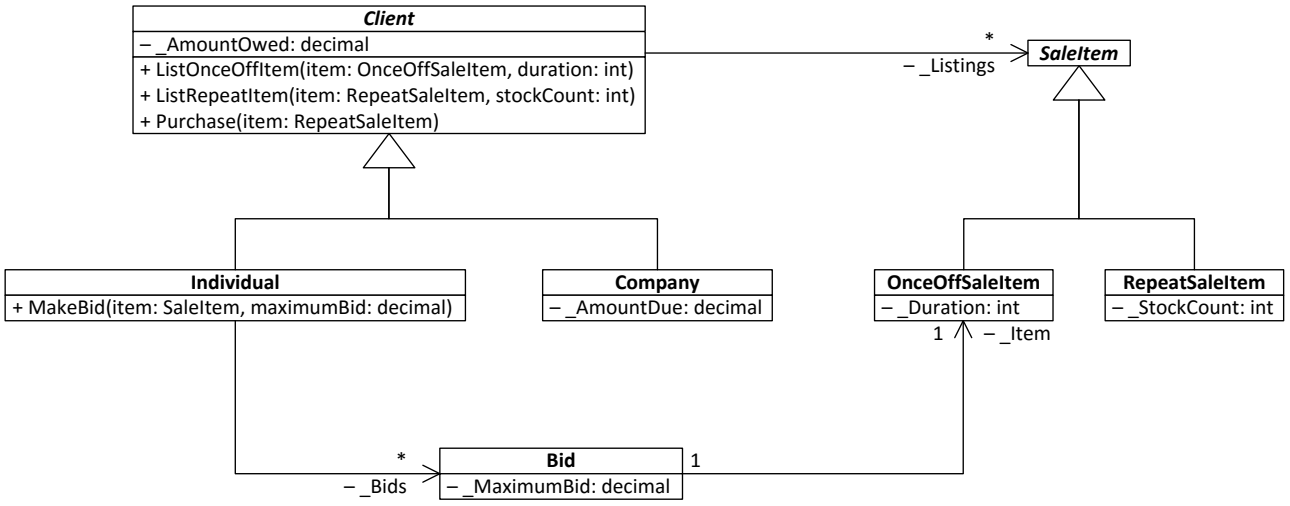
**Question a.**

a) Consider the following problem statement:

*You have been hired by a major online company to develop the underlying software system for a new web site which will compete with other combined auction and retail sites. The software system needs record clients who will be either individuals or companies. Both can sell items on the web site, either individual items for a once-off sale, e.g., a second-hand/used item being sold, or can be an item for repeat sales, e.g., for an individual/company selling Blu-Ray titles as part of an ongoing retailing business. Once-off items will also have a duration recorded, specifying how long the auction will remain open for (either 4, 7, or 10 days), after which time they will no longer appear on the web site. Repeat sales items will have a record of how many items are in stock and once all items are sold they will no longer appear on the web site. Individuals can also place bids for both once-off items, for which they will specify their maximum bid amount, and can purchase repeat sales items. The software system will also track how much money individuals owe for purchasing items and how much money is owed to both individuals and companies for items sold.*

Your task is to prepare a UML class diagram illustrating the concepts in the problem statement above. Your class diagram must demonstrate the following elements:

- Attributes, properties, and methods;
- Collections;
- Abstract classes;
- Inheritance;
- Associations (any type); and
- Cardinality.




---

*Note that the above UML is only one example solution, many others will be acceptable.*
*Marking scheme (15 marks):*

- *7.0 marks (1.0 mark each) – the following elements are applied appropriately in the UML: attributes, properties, methods, collections/cardinality, abstract classes, inheritance, and associations.*
- *6.0 marks (2.0 marks each) – the following elements are represented appropriately in the UML: clients, sale items, and bids.*
- *2.0 marks – the notation in the UML is otherwise correct*

---

**Question b.**

b) For this task you are required to write C# code satisfying the following requirements:

- An abstract class named Purchase which has:
  - A string attribute for the description of the purchase;
  - A decimal attribute for the amount owed;
  - Read-only properties encapsulating the above attributes;
  - A method MakePayment which allows only derived classes to specify a payment against the amount owed, returning true if the total amount owed has been paid or false if there is still an amount owing after the payment;
  - A custom constructor which takes a parameter for description and price, initialising the attributes above (the price represents the initial amount owed); and
  - A ToString() method which returns a string containing the asset description and price in the format:

    *description*: *price*
- A class named CashPurchase which inherits from the Purchase class and has:
  - A decimal attribute for amount paid;
  - A string attribute for change owed;
  - Read-only properties encapsulating the above attributes;
  - A custom constructor which takes description and price, passing this information to the base class constructor and initialising the above attributes as required;
  - A MakePayment() method (returns void), which accepts a decimal parameter indicating the amount paid, increases the amount paid attribute by the parameter value, and updates the change owed attribute to a string formatted as follows:

    *note_value* dollars in notes and *coin_value* in coin.

    *For example, paying $50 for a price of $8.50 would result in a string "40 dollars in notes and 1.5 in coin" (currency format could be used to improve this, but is not required).*

*Hint: notes come in $5, $10, $20, $50, and $100, so the amount in notes will always be a multiple of $5, and coin value will always be less than $5.*

** See over page for sample solution.*

---

*Marking scheme (15 marks):*
*Purchase class:*

*1.0 mark – class is declared with abstract keyword*
*2.0 marks – description and amount owed attributes and read-only properties correct*
*1.0 marks – constructor accepts parameters and initialises attributes correctly*
*2.0 marks – MakePayment method has protected accessibility and is otherwise correct*
*1.0 marks – ToString returns correctly formatted string and is otherwise correct*

*CashPurchase class:*

*1.0 mark – class inherits from Purchase class*
*1.0 mark – amount paid attribute and read-only property correct*
*2.0 marks – constructor accepts and initialises amount paid, passing all parameters to base*
*0.5 mark – MakePayment method adds amount parameter to amount paid*
*1.0 mark – MakePayment method correctly calculates change owed (numerical calculation)*
*0.5 mark – MakePayment modifies change owed attribute if amount paid exceeds amount owed*
*2.0 marks – MakePayment correctly updates change owed attribute with specified format (only if correct)*

*Penalise up to 2.0 marks is there are clear problems with the syntax.*

```
abstract class Purchase
{
    private string _Description;
    public string Description
    {
        get { return _Description; }
    }

    private decimal _AmountOwed;
    public decimal AmountOwed
    {
        get { return _AmountOwed; }
    }

    protected bool MakePayment(decimal amount)
    {
        bool result = false;

        _AmountOwed -= amount;
        if (_AmountOwed < 0)
            result = true;

        return result;
    }

    public Purchase(string description, decimal price)
    {
        _Description = description;
        _AmountOwed = price;
    }

    public override string ToString()
    {
        return string.Format("{0}: {1:c}", _Description, _AmountOwed); // note currency format optional
    }
}

class CashPurchase : Purchase
{
    private decimal _AmountPaid;
    public decimal AmountPaid
    {
        get { return _AmountPaid; }
    }

    private string _ChangeOwed;
    public string ChangeOwed
    {
        get { return _ChangeOwed; }
    }

    public CashPurchase(string description, decimal price)
        : base(description, price)
    {
        _AmountPaid = 0.0M;
    }

    public void MakePayment(decimal amount)
    {
        _AmountPaid += amount;
        decimal difference = _AmountPaid - base.AmountOwed;
        if(difference > 0)
        {
            // Alternatively can be solved with while loop (for $100 notes) and if statements (other notes)
            int dollarValue = (int)difference - ((int)difference % 5);
            decimal coinValue = difference - dollarValue;
            _ChangeOwed = string.Format("{0} dollars in notes and {1} in coin", dollarValue, coinValue);
        }
        else                                                                    // optional / not required
        {                                                                       // optional / not required
            _ChangeOwed = string.Format("You still owe {0:c}", -difference);    // optional / not required
        }                                                                       // optional / not required
    }
}
```