**OOP/C# Tutorial**

(a)

(i) In C# all object variables are in fact object references. With this in mind, explain the difference between the following two assignment statements (assuming that object variable *myObject* is an object of class *myClass*) :

int aNumber=12;

myClass myObject=new myClass();

(ii) In C#, function arguments are normally passed by value which, in the case of passing object arguments means that object references are passed by value. With this in mind, explain what is wrong with the following piece of code :

public class BankAccount

{

public BankAccount(int b) { balance=b;}

public void transferFunds(int amount, BankAccount myAccount)

{

balance=balance-amount;

int newBalance=myAccount.balance+amount;

myAccount=new BankAccount(newBalance);

}

private int balance;

}

(b) Consider the following C# class which represents a node for insertion into a linked list :

public class Node

{

public Node ptr=null;

private int val;

public Node(int v)

{

val = v;

ptr = null;

}

public void printNode()

{

System.Console.WriteLine("Node val " + val);

}

}

1. In C, this would lead to an infinitely recursive structure. Explain why it is OK in C#.
2. Produce a *LinkedList* class with methods *addNode()* and *printList()* which respectively add a node to the end of the list and print the contents of the list. Make any other reasonable assumptions in your code and add any other methods you feel appropriate.
3. In part (ii), the *LinkedList* class you produced using the *Node* class can obviously only store integers at each node. Describe how, through the use of inheritance and polymorphism, you would produce a reusable *LinkedList* class that was able to store and display the contents of objects of any class at each node.

**OOP/C# Tutorial**

**Solutions**

(a)

(i)

int aNumber=12;

myClass myObject=new myClass();

In the first case, the variable on the left of the assignment contains the value 12 and in the second case the variable on the left of the assignment contains a reference (address) of the object.

(ii)

public class BankAccount

{

public BankAccount(int b) { balance=b;}

public void transferFunds(int amount, BankAccount myAccount)

{

balance=balance-amount;

int newBalance=myAccount.balance+amount;

myAccount=new BankAccount(newBalance);

}

private int balance;

}

This code implies that the state of the argument *myAccount* is changed by the *transferFunds* method but because the argument is passed by value, only a copy of the reference to the original *myAccount* object is passed. The original one remains unchanged.

(b)

public class Node

{

public Node ptr=null;

private int val;

public Node(int v)

{

val = v;

ptr = null;

}

public void printNode()

{

System.Console.WriteLine("Node val " + val);

}

}

1. This doesn’t lead to an infinite recursion because *ptr* is a reference to a *Node* object so the class is not full aggregation.

public class LinkedList

{

private Node head = null;

private Node tail = null;

public void addNode(Node n)

{

if (head == null) // Empty list

{

head = n;

tail = n;

}

else

{

tail.ptr = n;

tail = n;

}

}

public void printList()

{

Node curr = head;

while (curr != null)

{

curr.printNode();

curr = curr.ptr;

}

}

}

1. We redefine the node to be abstract class and then extend this to any node type (eg. one containing strings):

public abstract class Node

{

public Node ptr;

public Node()

{

ptr = null;

}

public abstract void printNode();

}

public class myNode:Node

{

public String val;

public myNode(String s):base()

{

val = s;

}

public override void printNode()

{

System.Console.WriteLine(val);

}

}