COMP102

Part 1

Data Types and Structures

Lecture 1

Background and Review

COMP101 introduced the basic elements of the Java programming language.

This part of COMP102 is concerned with the topic of

Data Structures

- the abstract properties, organisation, and operations on these.
- their specific realisations in Java.

This description may raise a few questions:

1. Why consider more involved mechanisms for storing and operating upon data, when it possible to realise solutions to programming design problems using only Primitive Data Types, Strings, and simple Arrays?
2. Even if there is some point to this, why be concerned with anything other than how to implement these in Java? In other words, why consider these in a *language independent* context?

Answers:

1a. The initial design of a solution to an application problem is frequently expressed in terms of processing data that is assumed to be stored within some highly structured form.

This may happen because the problem suggests such representations in a 'natural' way,

1b. The choice of data structure can have a significant influence on algortihmic efficiency.

2. Java is just a *single* example of a specific of programming languages paradigm.

Understanding the abstract organisation of complex data structures provides the basis to implement these in any language

Over the next lectures the following structures will be examined:

Multi-dimensional Arrays.
Linked Lists.
Stacks and Queues.
Trees.

In this lecture, we examine some general aspects of data structures.

Representation and Manipulation

It may appear that the only issue of interest is how to implement a structure using of basic language types.

How to implement different operations and what operations to provide is then directed by the implementation.

Consider the String class.

This provides:

- Mechanisms for creating new Strings.
- Mechanisms for performing operations on Strings,

There are two points to notice:

1. Both are defined within a **single** package.
2. How the representation is done does not have to be known by programs using Strings.

Java provides an environment for implementing compound data types (such as String) whereby the representation can *only* be manipulated by the operations that are defined to do so.

In many programming languages there is no 'clean' mechanism for enforcing this.

By combining the definitions of

- The implementation of a data structure.
- The operations that act upon the structure.

implementation details can be 'hidden' from programs employing it.

This means that:

1. A designer, X, using data structures defined by Y, only needs to know the operations (i.e. the methods) and their definitions as provided by Y: X does not need to know how these are realised by Y.
2. Run-time errors are less likely to arise by reason of inconsistencies caused when the internal form of a data structure can be manipulated directly.
3. A new implementation of a structure could be substituted without programs using it having to be rewritten.

The organisation of **Abstract Data Types** (ADTs), as a collection of data and the operations acting upon the data, is called

Encapsulation

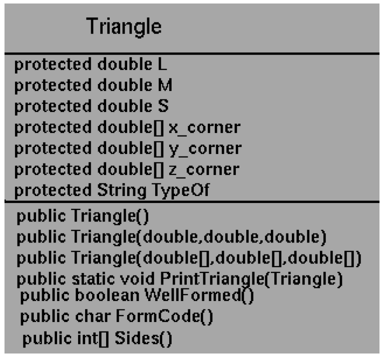
This is an important element of Object-Oriented Programming approaches.

Arrays of Objects

Java does not limit the use of Array structures to the basic types - **int**, **double**, **char**, **boolean**, etc, but also allows arrays of Object Instances,

This is particularly useful in the context of defining tables of **records**, of the type that would occur in a typical relational database.

As an example, consider a **Triangle** Object, whose Class Diagram is shown below:



To define an **array** of some number, LEN, of **Triangle** Objects:

Triangle [] TriArray = **new** *Triangle*[LEN];

The declaration will not instantiate any of the individual components.

But this could be done by:

for (i=0; i<LEN; i++)
  TriArray[i] = **new** Triangle();

A Simple Example Program

Given an integer bound on the length of any side of a triangle: 10 say,

1. Determine the number of **distinct** valid triangles with integer side lengths at most 10.
2. Output the number triangles in each category.
3. For each category of valid triangle, list all of the valid side lengths.

Analysis

There is no user supplied input.

1. How to enumerate all possible distinct cases.
2. How to find the total numbers in triangle category.
3. How to print the results.

Three lengths (S,M,L) (with $S \leq M \leq L$) define a triangle if $S + M > L$.


```
final static int LEN_UPB = 10;
static int MAX_POSS = 0;
//
// Calculate number of triangles with
// side length at most LEN_UPB.
//
private static void Compute_Max ()
  {
  for (int i=1; i<=LEN_UPB; i++)
    for (int j=i; j<=LEN_UPB; j++)
      for (int k=j; k<=LEN_UPB; k++)
        if (i+j>k) MAX_POSS++;
  }


static Triangle[] TriArray;
    ....
public static void main( String[] args )
  {
  Compute_Max();
  TriArray = new Triangle[MAX_POSS];
     ...
  }
```


There are **three** types of triangle - equilateral, isosceles, and scalene - three integers count these:

static int equi_tot=0;
static int iso_tot=0;
static int scal_tot=0;

```
for (int i=1; i<=LEN_UPB; i++)
  for (int j=i; j<=LEN_UPB; j++)
    for (int k=j; k<=LEN_UPB; k++)
      if (i+j>k)
        {
        TriArray[IsTriangle] = new Triangle(i,j,k)
        //
        // Determine counter to increment
        //
        switch(TriArray[IsTriangle].FormCode())
          {
          case 'E': equi_tot++; break;
          case 'I': iso_tot++; break;
          case 'S': scal_tot++;
          };
        IsTriangle++;           // Increase count
        };
```

When executed, in a within a suitable output harness,


```
Triangles with integer side lengths at most 10
################################################
Well formed     | 125
Equilateral     | 10
Isosceles       | 65
Scalene         | 50
################################################

Equilateral Triangles with Integer Side Length at most 10
##########################################################
(1,1,1) ; (2,2,2) ; (3,3,3) ; (4,4,4) ; (5,5,5) ;
(6,6,6) ; (7,7,7) ; (8,8,8) ; (9,9,9) ; (10,10,10) ;

Isosceles Triangles with Integer Side Length at most 10
##########################################################
(1,2,2) ; (1,3,3) ; (1,4,4) ; (1,5,5) ; (1,6,6) ;
(1,7,7) ; (1,8,8) ; (1,9,9) ; (1,10,10) ; (2,2,3) ;
(2,3,3) ; (2,4,4) ; (2,5,5) ; (2,6,6) ; (2,7,7) ;
(2,8,8) ; (2,9,9) ; (2,10,10) ; (3,3,4) ; (3,3,5) ;
(3,4,4) ; (3,5,5) ; (3,6,6) ; (3,7,7) ; (3,8,8) ;
(3,9,9) ; (3,10,10) ; (4,4,5) ; (4,4,6) ; (4,4,7) ;
(4,5,5) ; (4,6,6) ; (4,7,7) ; (4,8,8) ; (4,9,9) ;
(4,10,10) ; (5,5,6) ; (5,5,7) ; (5,5,8) ; (5,5,9) ;
(5,6,6) ; (5,7,7) ; (5,8,8) ; (5,9,9) ; (5,10,10) ;
(6,6,7) ; (6,6,8) ; (6,6,9) ; (6,6,10) ; (6,7,7) ;
(6,8,8) ; (6,9,9) ; (6,10,10) ; (7,7,8) ; (7,7,9) ;
(7,7,10) ; (7,8,8) ; (7,9,9) ; (7,10,10) ; (8,8,9) ;
(8,8,10) ; (8,9,9) ; (8,10,10) ; (9,9,10) ; (9,10,10) ;

Scalene Triangles with Integer Side Length at most 10
##########################################################
(2,3,4) ; (2,4,5) ; (2,5,6) ; (2,6,7) ; (2,7,8) ;
(2,8,9) ; (2,9,10) ; (3,4,5) ; (3,4,6) ; (3,5,6) ;
(3,5,7) ; (3,6,7) ; (3,6,8) ; (3,7,8) ; (3,7,9) ;
(3,8,9) ; (3,8,10) ; (3,9,10) ; (4,5,6) ; (4,5,7) ;
(4,5,8) ; (4,6,7) ; (4,6,8) ; (4,6,9) ; (4,7,8) ;
(4,7,9) ; (4,7,10) ; (4,8,9) ; (4,8,10) ; (4,9,10) ;
(5,6,7) ; (5,6,8) ; (5,6,9) ; (5,6,10) ; (5,7,8) ;
(5,7,9) ; (5,7,10) ; (5,8,9) ; (5,8,10) ; (5,9,10) ;
(6,7,8) ; (6,7,9) ; (6,7,10) ; (6,8,9) ; (6,8,10) ;
(6,9,10) ; (7,8,9) ; (7,8,10) ; (7,9,10) ; (8,9,10) ;
```

Summary

1. An **Abstract Data Type** (ADT) is defined by a collection of data and a set of operations manipulating it. The principle of **Encapsulation** states that Data and Operations on the data should be defined together.
2. In Java ADTs can be realised as new Objects using Class definitions.
   a) Instance Methods provide a 'safe' mechanism for accessing (instance) fields of a class
3. Encapsulation and Data Hiding provide a number of advantages in the development of applications programs:
   a) The organisation of data within an ADT can be changed without programs using the ADT having to be aware of it (and thence re-written).
   b) The risk of errors in **data integrity** is greatly reduced, since only permissible methods can alter data.
   c) Documentation can be organised in more tightly structured fashion
4. To support the ideas of Data Hiding, data fields in a class definition are normally declared as **private** or **protected**.
5. ADTs are a fundamental study within software and algorithm design.