Class Diagrams

Candidates offering Information Technology need to be able to read and create class diagrams. Both Paper 1 and Paper 2 will make use of class diagrams in varying ways. Standardisation is important because, as with an algorithm, these diagrams should be able to be used by programmers as the basis for coding structures in any language.

The main components of a class diagram are:

- The name of the class
- The fields of the class
- The methods of the class

If a candidate is being asked to draw up a class diagram from a specification or from some supplied code, it is important that these three components (name, fields and methods) are named exactly as defined in the specification or code. Similarly, parameters to methods should be named and typed as per the specification.

Fields

Fields should be defined in the same order as they are supplied, with the same name. They will additionally have an access modifier and a type.

Methods

Methods should be defined in the same order as they are supplied, with the same name. They will additionally have an access modifier, possibly a parameter list and a type. Method names are always followed by a set of parentheses, whether there are parameters or not.

Access modifiers

Candidates are expected to be familiar with the following three access modifiers and understand their function:

```
+    →    public
-    →    private
#    →    protected
```

Public : directly accessible in any class in the same package
Private : directly accessible only in the class in which defined
Protected : directly accessible in the class in which defined as well as any class which inherits from it and by any class in the same package

Types

Field, method and parameter types are generic in a class diagram so as to be used in any programming language. The following conventions will apply:

- `string`
- `integer`
- `real`
- `char`
- `boolean`
- `[ ] (array)`

Imperative to the understanding of class diagrams and their contents are the concepts of static and non-static, typed and void.

- Static fields and methods belong to a CLASS.
- Non-static fields and methods belong to an OBJECT

- Void methods do something.
- Typed methods do something AND return a value of the same type as the method.

Field definitions

The naming convention which applies to fields is as follows: names start in lower case. A field name which is a combination of two words still starts in lower case, but the second word starts in upper case. Types are always shown in lower case, not abbreviated, and follow the field name.

Example of field definitions

```
- description : string
- qty : integer
- costPrice : real
- sellingPrice : real
```

Constructor method

A constructor method is used to instantiate objects and either assign the fields default values or specific values via a set of parameters. As a standard, the constructor method is named "Constructor", not the name of the class, which will happen during coding. Note the use of upper case in the name – this is because the method name will be the same as the class, and class names always have the first letter in upper case. The parameters to a constructor method must be in the same order supplied and named and typed accordingly.

Example of a constructor method

```
+ Constructor (d : string, q : integer, c: real)
```

toString method

The toString method is used to concatenate the fields of an object into a single string. A toString method can also be used to do the same with an array of objects via a loop. This detail would not be shown in a class diagram, but would be coded according to the specification.

Example of a toString method

```
+ toString() : string
```

Methods which operate on non-static fields

Generally, candidates will need to add accessor and mutator methods to a class diagram. These methods are important to allow the manipulation of, particularly, private fields of an object. Accessor methods are always prefixed with "get" and mutator methods are always prefixed with "set". Accessor methods are always typed – this stands to reason as they are going to access, or get, a value of a field and return this. Mutator methods are always void methods, but will always accept a parameter which is passed to the method.

Example of mutator and accessor methods

```
+ getDescription() : string
+ getQty() : integer

+ setPrice(p:real)
+ setSellingPrice(sp:real)
```

Void methods other than accessor methods

Various methods may be called for in a class diagram which allow for, for example, a calculation. These methods are defined in the same way as a normal void method.

Example of a void method

```
+ sell (q : integer)
```

Static fields and constants

Static constants and static fields need to be distinguishable from static fields. There are conventions which are applied as follows:

- Static fields are named and typed in a similar fashion to non-static fields. However, to distinguish between the two, they are underlined in a class diagram.
- Constants, whose value by definition cannot change, are distinguished twofold: they are underlined but the name of the field is always shown totally in upper case.

Example of static fields and static constants

```
- totalQty : integer          (static)
- totalSales : real           (static)
+ MARKUP = 75 : integer       (static constant)
```

Methods which operate on static fields and constants

These methods are defined and operate in a similar manner to methods which operate on non-static fields. Once again, to distinguish them, they are shown in a class diagram underlined.

Example of methods which operate on static fields

```
+ getTotalQty() : integer
+ getTotalCost() : real
```

Full example of a class diagram incorporating all of the above:

Fruit
- description : string - qty : integer - costPrice : real - sellingPrice : real - totalQty : integer - totalCost : real - totalSales : real + MARKUP = 75 : integer
+ Constructor (d : string, q : integer, c: real) + getDescription() : string + getQty() : integer + getPrice() : real + getSellingPrice() : real + setDescription(d : string) + setPrice(p : real) + sell(q : integer) + getTotalQty() : integer + getTotalCost() : real + getTotalSales() : real + toString() : string

Inheritance

There are three things to bear in mind when reading or creating class diagrams in an inheritance scenario:

- The protected access modifier
- The constructor method in the child class
- The layout of the diagram showing the link between the child and parent class

Access modifiers

Fields and methods can be set as public, private or protected in an inheritance scenario. These have been detailed earlier in this document. Candidates need to be able to understand which modifier is required for which field or method by reading the question carefully. Remember # is used for protected.

Constructor method

The constructor method of a child class is a special case in terms of the parameters. Some designs show only the fields which relate to the child object as parameters. The standard to be adopted is that all fields must be shown in the constructer, ie the fields of the parent object as well as the additional fields which relate to the child object.

Example of constructors

Assume a parent object has the following three fields:

```
# description : string
# qty : integer
# costPrice : real
```

Assume that a child object has the following two fields:

```
- markUp : integer
- sellingPrice : real
```

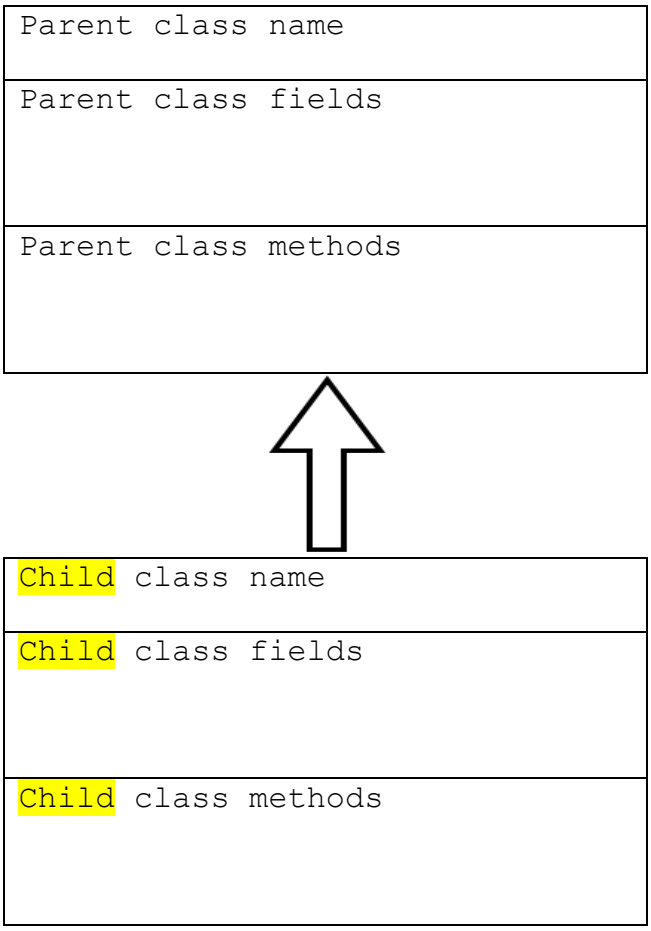
An example constructor for a PARENT object would look like:

```
+ Constructor(d : string, q : integer, c : real)
```

An example constructor for a CHILD object would look like:

```
+ Constructor(d : string, q : integer, c : real, m : integer, s :
real)
```

Class diagram layout



NB: The arrowhead is shown as clear, not filled.