> Where constructing algorithms is required, show code of example algorithms. If tracing is required, trace example algorithms.

D Object-oriented programming

D.1 Objects as a programming concept

D.1.1 Outline the general nature of an object

An object is an abstract entity in computer science. It represents an abstraction of a complex concept and thus allows programmers to deal with a large amount of interlinked data without having to deal with all of its detail at once. A real life example is a car engine. You don't need to be able to know exactly how an engine works to be able to operate it, you only need to know how its abstraction - the control elements in the cockpit - works. This also allows you to deal with many different types of engines because although they work differently but their abstractions are largely similar.

D.1.2 Distinguish between an object (definition, template or class) and instatiation

An class is the coded description of an object. It provides methods for the object to allow it to carry out actions, and describes its properties. An instantiation of an object is the physical storage space reserved for an object where its properties and current state can be stored. There can be only one class describing an object, but many instantiations of an object can exist within a program.

D.1.3 Construct unified modelling language diagrams to represent object designs

D.1.4 Interpret UML diagrams

D.1.5 Describe the process of decomposition into several related objects

D.1.6 Describe the relationships between objects for a given problem

D.1.7 Outline the need to reduce dependencies between objects in a given problem

If many things are interdependent in a program, changing one part can be very difficult as all dependencies must be taken into account when making changes. This increases difficulty of maintenance. Therefore, dependencies should be kept to a minimum or standard calling procedures should be implemented.

D.1.8 Construct related objects for a given problem

Scenario: a school students&staff database has to be developed with personal details of every people in the school.

Because students and teachers both have common properties, a master class is useful from which both groups will inherit some of their properties.

Object people: This object stores the general details of the people in the school, like gender, age, phone number and home address.

Object students: This objects inherits all the properties of the *people* class and adds some more properties specific to students, like grade, marks for subjects and days missed.

Object teachers: This object inherits all the properties of the *people* class and addso some more properties specific to teachers, like homeroom, days sick, salary and subjects teached.

In this scenario, only objects of teachers and students would be instantiated, but because they inherit from the people class, they will possess properties of this class. *people* is now a superclass while teachers and students are a subclass.

D.1.9 Explain the need for different data types to represent data items

Data is stored as a combination of binary values in the computer. Data types are used to store different kinds of data, like text, numbers, floats or other values. They are needed because they specify to the computer how to interpret the binary values in the storage.

D.1.10 Describe how data items can be passed to and from actions as parameters

In many programming languages, data can be passed to methods to calculate with. This is done by giving them to the method while calling it as parameters. In Java and C# parameters are given in brackets after the calling name of a method:

```
int s = addInt(3, 2); //<- The values to work with are given as parameters

public int addInt(int parameter1, int parameter2)
{
    int result = parameter1 + parameter2; //<- using the values passed as
parameters
    return result; //<- return the result of the operation to the calling
procedure
}
```

D.2 Features of OOP

D.2.1 Define the term encapsulation

Encapsulation is enclosing the properties and methods of an object so that they can easily be dealt with and are secured against invalid changes. This usually means making their object variables inaccessible from outside their class and using methods with security check mechanisms to change those variables.

D.2.2 Define the term inheritance

Inheritance is when an object inherits or adapts the properties and methods of an other object and uses them as if they were its own. An object of the other class does not need to be instantiated for this.

> There is an example for inheritance in section D.1.8
> There, the teachers and students inherit properties of the object *people*

D.2.3 Define the term polymorphism

Polymorphism is the method of defining multiple methods with same names but with different parameters in order to deal with different parameter configurations. When such a method is called the program automatically selects the method whose parameter configuration matches the parameters given by the calling method.

D.2.4 Explain the advantages of encapsulation

- Direct access to the variables of an object can be restricted. This reduces the risk of accidentally setting invalid values as properties.
- Security checks can be implemented to avoid setting invalid values as properties.
- If methods are set to access properties, objects inheriting properties can also use them → less complexity in the program

```
public class people
{
    private int age;    //<- private operator allows access to variable only
from within the class

    public void setAge(int x)
    {
        if (age < 1)
        {
            throw exception; //<- throw exception for an invalid value -
negative or 0 is not a valid age
        }
        else
        {
```

```
            age = x;
        }
    }
}
```

D.2.5 Define the advantages of inheritance

It is possible that many objects share a set of common properties and functions that they have inherited from a common object. The advantage is that it is easy to add functionality to the objects. If for example, in the school database both the teachers and students would need a property for storing an email address, this could be implemented in the *people* class. This way, the properties and methods would have to be implemented only once, which would make development easier and less prone to errors.

D.2.6 Define the advantages of polymorphism

A method can accept various sets of parameters. For example, the method in the *people* class that is used to set a phone number could be implemented in a way to accept a string as a phone number and in a way to accept an integer as phone number. This way, phone numbers could be entered using both ways.

D.2.7 Desribe the advantages of libraries of objects

Sorting and other complex algorithms and processes do not have to be re-invented.

D.2.8 Describe the disadvantages of OOP

- Increased complexity for small problems
- Unsuited for some kinds of problems

D.2.9 Discuss the use of programming teams

Compared to working alone, programming teams can have many advantages:

- People can work on many parts of a program at the same time, reducing time needed
- Code can be cross-checked to avoid mistakes
- Because people don't see other's information, module dependencies are reduced due to information hiding
- Expertise can be concentrated in a narrow field

D.2.10 Explain the advantages of modularity in program development

- Modules can be tested and debugged separately
- Time reduced as work can be done on many modules simultaneously

- Modules can be easily changed, so easier maintenance
- Internal structure of moduls can be changed without having to worry about the other modules → easier code optimisation

D.3 Program development

D.3.1 Define the terms: class, identifier, primitive, instance variable, parameter variable, local variable

class: An abstract model for objects. It defines the properties and methods of an object. It is like a data type for an object. For example, in Java strings are not fundamental variables like integers but objects.

identifier: A pointer that explicitly identifies an object. Generally, it is the variable name.

primitive: primitive data types are the ones involving the most basic entities in informatics: numbers. Other data types like arrays are abstract because they combine many primitive data types.

instance variable: A variable in a class from which every instantiated object gets its own copy. If a class called *people* would have an instance variable called *age*, every object of this calss would have its own variable called age.

parameter variable: A variable that is passed along to a function that is called to perform operations with. In Java it is the argument passed in the brackets with the function caller.

```
int a = aSimpleFunction(int b); //<- see the parameter variable in the brackets? Magnificent!
```

local variable: A variable declared inside a function. This variable is only known and accessible by the function it is declared inside.

D.3.2 Define the terms: method, accessor, mutator, constructor, signature, return value

method: A method is a set of rules defining operations that are executed.

accessor: A method used to return values of a private variable of an object.

mutator: A mutator is a method used to control changes to the private variables of an object. Setter and accessor methods are mutator methods.

constructor: The method called when an object is instantiated.

signature: Defines the inputs and outputs of a method. In some programming languages it even defines the errors thrown by the function.

return value: Specifies the data type that a function returnes to its caller.

D.3.3 Define the terms: private, protected, public, extends, static

private: A constructor that hides variables and methods from access from outside the class.

protected: A member access modifier. A protected is only accessible within its own or derived class instances.

public: Member can be accessed by any method that references its calss.

extends: This keyword is used to state from which class the current class inherits its methods and properties.

static: Declares a static member of a class that will be the same for all members.

D.3.4 Describe the uses of the primitive data types and the reference class string

In examination questions the primitive types will be limited to int, long, double, char and Boolean.

Primitive data types are most commonly used to store simple values. They are also used as the building blocks of the more complex abstract data types.

D.3.5 Construct code examples to implement assessment statements D.3.1-D.3.4

Let us design a class. It should be used to model people in a system:

```
public class People    //The begin of the declaration of the class
{
    private int age;    //Note the private keyword restrict access to
variable only within this class
    private string name;
    private int phone;

    new People()    //The constructor method is used to initialise some
values when an object is instantiated.
    {
        age = 0;
        name = "";
        phone = 0;
    }

    public void setAge(int x)    //The setter method is used to modify
values of variables.
    {
        age = x;
    }
```

```
    public int getAge()    //The accessor method (part of the mutator
methods) is used to access values of variables.
    {
        return age;
    }

    public void setPhone(int x)    //The next two methods are overloaded,
the parameter variables decide the function to be called.
    {
        phone = x;
    }

    public void setPhone(string x)
    {
        phone = Transform(x);
    }

    private int Transform(string number)    //Private keyword restricts use
of method for only within the class.
    {
        int numerical = Convert.toInt(number);    //local variable
        return numerical;    //Return specifies what value the method
returns to its caller.
    }
}
```

D.3.6 construct code examples related to selection statements

```
int a = 3;
int b = 2;

if (a < b) //selection statement
{
    output("b is larger");
}
else
{
    output("a is larger");
}
```

D.3.7 Construct code examples related to repetition statements

```
/*
This code adds up all the numbers from 0 to 10.
Notice that instead of typing 0+1+2+3.....+10
a loop is used to add up the numbers
*/
int count = 0;
```

```
int total = 0;

while (count <= 10)
{
    total = total + count;
    count = count + 1;
}
```

D.3.8 Construct code examples related to static arrays

```
/*
Following example illustrates the use of static arrays.
Sequential search is used to find a number in a given array.
*/
public void searchNumber(int[] numbers, int desired)
{
    int count = 0; //set up counter for loop
    while (count < numbers.count) //loop until end of array is reached
    {
        if (numbers[count] == desired) //if desired number equals to number
at position count, return count
        {
            return count;
        }
    }
}
```

D.3.9 Discuss the features of modern programming languages that enable internationalisation

- Use of common character sets among many platforms and languages, like UNICODE
- Platform independent high level languages enable code to run on many platforms

D.3.10 Discuss the ethical and moral obligations of programmers

- Adequate testing of products to prevent possibilities of commercial or other damage

The 2008 financial crisis was coused by an error in the programs on the stock market

- Acknowledging the work of other programmers (plagiarism)
- Open Source movement

HL Extension

D.4 Advanced program development

D.4.1 Define the term recursion

D.4.2 Describe the application of recursive algorithms

D.4.3 Construct algorithms that use recursion

D.4.4 Trace recursive algorithms

D.4.5 Define the term object reference

D.4.6 Construct algorithms that use reference mechanisms

D.4.7 Identify the features of the abstract data type (ADT) list

D.4.8 Describe application of lists

D.4.9 Construct algorithms using a static implementation of a list

D.4.10 Construct list algorithms using object references

D.4.11 Construct algorithms using the standard library collections included in JETS

D.4.12 Trace algorithms using the implementations described in assessment statements D.4.9-D.4.11

D.4.13 Explain the advantages of usign library collections

D.4.14 Outline the features of ADT's stack, queue and binary tree

D.4.15 Explain the importance of style and naming conventions in code