Rewarding Learning

AS LEVEL

FACT FILES

Software Systems Development

AS1 Introduction to Object Oriented Development

Part ii – DEFINING DATA

For first teaching from September 2013

For first AS award in Summer 2014

For first A Level award in Summer 2015

software systems development

FACT FILE

software systems development

Introduction to Object Oriented Development

Part ii – DEFINING DATA

Learning Outcomes

Students should be able to:

- explain and apply
  - Primitive types
  - Reference (object) types
- identify and name objects;
- explain that an object is an occurrence of a class;
- identify and name classes (abstract, base and inherited classes);
- examine and apply the functions of classes;
- explain the use of a static variable within a class;
- examine and apply attributes and types to a class;
- identify that attributes represent the properties of a class; and
- identify the use of interfaces in relation to specific classes.

Course Content

An object-oriented program can be thought of as a collection of interacting objects.

Classes

A class is a set of related fields and methods detailing how to construct an object, much like a plan for building a house. An object is an instance of a class. Many objects can be instantiated from a class. A class will have one or more constructors which enable an object to be created. An application can be made up of many objects from different classes.

A simple example of a class is a student with the data: student number; name and date of birth. A method for this class could be to calculate the student age on the 1st of September. An example of an object for this class could be: 111; Stevenson; 10/12/1995.

Note that the data can be a mixture of primitive and reference types.

Types

When we plan to use a piece of data in a software program, we must also give some thought as to how that data will be held in the computer's memory. This allows us to declare the required memory size and format, or type, of the data unit. Is the data number or text? If it is a number, is it an integer or a decimal-point number? Once a variable has been declared as a certain type in the program, it cannot change type.

Primitive Types

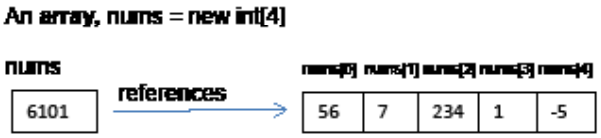
Most programming languages have certain primitive data types built in for use. For example the primitive type, int, lets you declare integer (whole number) variables in the range -2,147,483,648 to +2,147,483,647.

Type	Range of possible values	
byte	0	255
short	-32,768	32,767
int	-2,147,483,648	+2,147,483,647
long	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
double	-5 x 10324	1.7 x 10308
char	This is any unicode character eg; 'd'	
bool	0	1

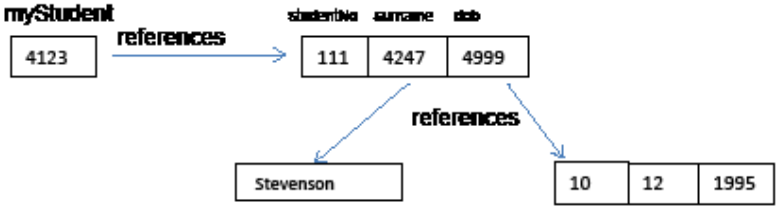
Reference Types

Commonly defined reference types are **strings and arrays.** Creating user-defined types is at the heart of object oriented programming. An **object** is a reference type. Reference types store the address of their data rather than the data itself. They reference where the data is held in memory.

For example, a **string**, 'abc ' will reference contiguous locations in memory capable of holding the string of characters.



An object, myStudent, instantiated from the class Student, made up of the data:
int studentNo; string surname; SimpleDate dob



Categories of Classes

A basic class can be referred to as **base, super or parent**. This class can be built upon to give a refined class referred to as **derived, sub or child**. The derived classes may have access to some or all, of the base attributes depending on the visibility assigned. This is known as inheritance and languages such as JAVA and C# support single inheritance only. Interfaces allow a sense of multi-inheritance for these languages.

Base Class

In practice, classes are often evolved from other classes. The top level class is the **base class**, or parent class. An example of a base class would be ...

```
public class Boat
{
    // data declaration
    private  int  boatLength;

   // constructor  - passes in a value
   public Boat (int len)
   {
          BoatLength   =   len;   // uses
properties to set the value
   }

   // property
   public int  BoatLength
   {
        get { return boatLength; }
        set  {
         // validate value if necessary
              boatLength  = value;
              }
      }
      // insert methods here
}
```

The base class is Boat. Note that the constructor has a variable passed in, len. This is the length of an object instantiated from this class and is stored in the class variable boatLength. All boats have length so this variable will be required by any further derivation of this class.

Inheritance

Inheritance is the ability of object oriented languages to define a new class which inherits the behaviours of an existing class. The new class is called a derived, sub or child

```
public  class  Yacht  :  Boat
{
     // data declaration
     private  int noOfSails;

    // constructor
    public Yacht( int len, int  sails)
            : base(len)      // len passed back to base class Boat
   {
      NoOfSails   =  sails;

   }
   // property  - accessor  / modifier
   public int  NoOfSails
   {
        get { return noOfSails; }
        set  {
         // validate value if necessary
              noOfSails  = value;
              }
    }
    // insert  Yacht methods here
}
```

A new class Yacht has been defined which inherits behaviour from its base class Boat. It has a variable, noOfSails, which defines the number of sails on a yacht. The length of the yacht is passed back to the base class Boat for processing. Another class MotorBoat could be derived from the Boat class but would have a variable engineSize. Again, the length of the motor boat is passed back to the base class Boat.

An important principle behind inheritance is code reuse – not having to reinvent the wheel.

Static Variables

A static variable is a variable that has had a memory location allocated for the entire lifetime of the run of the program. This is in contrast to a local variable in a class which is allocated when the class is first called. Furthermore, static variables can be allocated and used without declaring an instance of the class.
Static methods use the same principle.

Interfaces

Designers may wish to impose a structure on subsequent derived sub classes by defining an **interface** consisting of method signatures.

```
interface IboatMove
{
      void Method_To_Make_Boat_Move( object obj);
}
```

In the example the designer does not define the process of how the boat moves. The Boat class can include the detail of the process – using currents and oars. The Yacht class can include the detail of the process - using wind speed and sails. The Motorboat can include the detail of the process – using currents and engine size.

```
public class Yacht : Boat : IboatMove
{
        public void Method_To_Make_Boat_Move( object obj)
        {
         // insert code to process boat movement using wind speed and sails
        }

}
```

A very common interface is the **IComparable** used in comparing objects and implemented in Sorting.

Activities

QUESTIONS

Q1 Which of the following statements best describe the relationship between an object and a class?

a. An object and a class are the same.
b. A class is an instance of an object.
c. An object is a template for a class.
d. An object is an instance of a class.

Q2 How many objects of a given class can be created in a program?

a. One per defined class
b. One per constructor definition.
c. As many as the program needs.
d. One per main() method.

Q3 Place the following object pairs into the correct category, base or inherited.

	base	inherited
Clothing; Coat	Clothing	Coat
Water; Liquid		
Vehicle; Car		
Rose; Flower		
Book; Publication		

Q4 Which data type would you use to hold a variable containing the weight in Kg of newly born puppies?

a. Integer
b. Short
c. Boolean
d. Float

software systems development

COUNCIL FOR THE CURRICULUM EXAMINATIONS AND ASSESSMENT 29 Clarendon Road, Clarendon Dock, Belfast BT1 3BG
✆ +44 (0)28 9026 1200 +44 (0)28 9026 1234 +44 (0)28 9024 2063 info@ccea.org.uk www.ccea.org.uk
Rewarding Learning