SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

COIMBATORE-35

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

19CST251-OBJECT ORIENTED PROGRAMMING USING C++

Unit-III

1.What is meant by reusability

Reusability is a feature which is supported in object-oriented programming. This allows the reuse of existing classes without redefinition.

2.Define Inheritance

- In object-oriented programming, **inheritance** is a way to form new classes (instances of which are called objects) using classes that have already been defined.
- The former, known as **derived classes**, take over (or **inherit**) attributes and behavior of the latter, which are referred to as **base classes**.
- It is intended to help reuse of existing code with little or no modification.
- **Inheritance** is also called **generalization**, because the **is-a** relationships capture a hierarchal relationship between classes of objects

3.What are the applications of Inheritance

There are many different aspects to inheritance. Different uses focus on different properties, such as the external behavior of objects, internal structure of the object, structure of the inheritance hierarchy, or software engineering properties of inheritance.

(i)Specialization

One common reason to use inheritance is to create specializations of existing classes or objects. This is often called *subtyping* when applied to classes. In specialization, the new class or object has data or behavior aspects that are not part of the inherited class.

Another form of specialization occurs when an inherited class specifies that it has a particular behavior but does not actually implement the behavior. Each non-abstract, concrete class which inherits from that abstract class must provide an implementation of that behavior. This providing of actual behavior by a subclass is sometimes known as *implementation* or *reification*.

(ii)Overriding

Many object-oriented programming languages permit a class or object to replace the implementation of an aspect—typically a behavior—that it has inherited. This process is usually called *overriding*.

(iii)Extension

Another reason to use inheritance is to provide additional data or behavior features. This practice is sometimes called *extension* or *subclassing*.

Extension is often used when incorporating the new features into the inherited class is either not possible or not appropriate.

(iv)Code re-use

One of the earliest motivations for using inheritance was to allow a new class to re-use code which already existed in another class. This practice is usually called *implementation inheritance*.

4.What are the Constraints of inheritance-based design?

- **Singleness**: using single inheritance, a subclass can inherit from only one superclass. Continuing the example given above, Person can be either a Student or an Employee, but not both. Using multiple inheritance partially solves this problem, as a StudentEmployee class can be defined that inherits from both Student and Employee. However, it can still inherit from each superclass only once; this scheme does not support cases in which a student has two jobs or attends two institutions.
- **Staticness**: the inheritance hierarchy of an object is fixed at instantiation when the object's type is selected and does not change with time. For example, the inheritance graph does not allow a Student object to become a Employee object while retaining the state of its Person superclass.
- **Visibility**: whenever client code has access to an object, it generally has access to all the object's superclass data. Even if the superclass is not a public one, the client can still cast the object to its superclass type. For example, there is no way to give a function a pointer to a Student's grade point average and transcript without also giving that function access to all of the personal data stored in the student's Person superclass.

5. What are the types of inheritance?
1. Single inheritance: A derived class with only one base class is called single inheritance
2. Multiple inheritance: A class can inherit properties from more than one class which is known as multiple inheritance
3. Multilevel inheritance: A class can be derived from another derived class which is known as multilevel inheritance.
4. Hierarchical inheritance: When the properties of one class are inherited by more than one class, it is called hierarchical inheritance.

6. How to define derived classes?
A derived class can be defined by specifying its relational ship with the base class in addition to its own details.
The syntax is:

```
class derived-class-name : visibility-mode base-class-name
{
……
…….
};
```

Here the visibility mode is optional and if present, may be either private or public. The default mode is private.
When a base class is privately inherited by a derived class, public members of the base class become private members of the derived class and therefore the public members of the base class can only be accessed by the member functions of the derived class.
When the base class is publicly inherited, public members of the base class become public members of the derived class and therefore they are accessible to the objects of the derived class.

7. What is inherited from the base class?
In principle, a derived class inherits every member of a base class except:

- its constructor and its destructor
- its operator=() members

- its friends

Although the constructors and destructors of the base class are not inherited themselves, its default constructor (i.e., its constructor with no parameters) and its destructor are always called when a new object of a derived class is created or destroyed.

8.Define virtual base class

- A base class that has been qualified as virtual in the inheritance definition.
- In multiple inheritance, a derived class can inherit the members of a base class via two or more inheritance paths.
- If the base class is not virtual, the derived class will inherit more than one copy of the members of the base class.

9. Define virtual function

- It is a function qualified by the virtual keyword. When a virtual function is called via a pointer, the class of the object pointed to determines which function definition will be used.
- Virtual functions implement polymorphism, whereby objects belonging to different classes can respond to the same message in different ways.

10. What is meant by pure virtual function

- A virtual function that is declared in a base class but not defined there. The responsibility for defining the function falls on the derived classes, each of which generally provides a different definitions.
- It is illegal to create instances of a class that declares a pure virtual function. So such a class is necessarily an abstract base class.

11. What is meant by subclass and superclass

Subclass: a class which has link to a more general class

Superclass: a class which has one or more members which are classes themselves.

12. Define abstract class

Abstract class is one that is not used to create objects. An abstract class is designed only to act as a base class. It is a design concept in program development and provides a base upon which other classes may be built.

13.What are the forms of inheritance

<!--[if mso &amp; !supportInlineShapes &amp; supportFields]&gt; SHAPE * MERGEFORMAT <![endif]–> The virtual function must be members of some function.

- They can’t be static members
- They are accessed by using object pointers.
- A virtual function can be a friend of another class
- A virtual function in a base class must be defined, even though it may not be used.
- The prototypes of the base class version of a virtual function and all the derived class versions must be identical. If two functions with the same name have different prototypes, C++ considers them as overloaded functions, and the virtual function mechanism is ignored.
- We cannot have virtual constructors, but we can have virtual destructors.
- While a base pointer can point to any type of the derived object, the reverse is not true.
- When a base pointer to a derived class, incrementing or decrementing it will not make it to point to the next object of the derived class. It is incremented or decremented only relative to its base type. Therefore we should not use this method to move the pointer to the next object.
- If a virtual function is defined in the base class, it need not be necessarily redefined in the derived class. In such cases, calls will invoke the base function.

14. What is meant by this pointer?

- C++ uses the unique keyword called **this** to represent an object that invokes a member function.
- **this** is a pointer that points to the object for which this function was called.
- This unique pointer is automatically passed to a member function when it is called. The pointer this acts as an implicit argument to all the member functions.
- One important application of the **this** pointer this is to return the object it points to.

15. What is meant by pointer and null pointer?

Pointer = pointer is a data type that holds the address of a location in memory.
Null Pointer = is a pointer that does not point to any data object. In C++, the null pointer can be represented by the constant 0.