CHAPTER - 5
General OOP Concepts
TYPE A : VERY SHORT ANSWER QUESTIONS

1.	**What are the two major types of programming languages?**
Ans.	The two major types of programming languages are Low Level Languages and High Level Languages.
2.	**Which two programming languages are low level languages?**
Ans.	Machine language and Assembly language are low level languages.
3.	**How are programs written in (i) machine language, (ii) assembly language?**
Ans.	(i) machine language: In machine language instructions are written in binary code (using 0 and 1). (ii) assembly language: In assembly language instructions are written using symbolic names for machine operations (e.g., READ, ADD, STORE etc.) and operands.
4.	**Why are low level languages considered close to the machine?**
Ans.	Low level languages are considered close to the machine because it provide a vehicle for the programmers to specify actions to be executed, so that all important aspects of a machine are handled simply and efficiently in a way that is reasonably obvious to the programmer.
5.	**Why is it easier to program with high level languages? Why are high level languages considered close to the programmer?**
Ans.	High level languages offer English like keywords, constructs for sequence, selection and iteration and use of variables and constants. Thus, is it easier to program with high level languages. High level languages are considered close to the programmer because it provides a set of concepts for the programmer to use when thinking about what can be done.
6.	**Which two related purposes should be served by a programming language?**
Ans.	A programming language should serve two related purposes as following: (i) it should provide a vehicle for the programmers to specify actions to be executed and (ii) it should a set of concepts for the programmer to use when thinking about what can be done.
7.	**Why 'C++' is called 'middle level language'?**
Ans.	'C++' is both "close to the machine" and "close to the programmer", so it is called 'middle level language'.
8.	**What do you understand by programming paradigm? Name various programming paradigms.**
Ans.	A Programming Paradigm defines the methodology of designing and implementing programs using the key features and building blocks of a programming language. Following are the different programming paradigms: (i) Procedural Programming (ii) Object Based Programming (iii) Object Oriented Programming
9.	**What are the characteristics of procedural paradigm?**
Ans.	✓ Do not model real world very well. ✓ More emphasis is on doing things. ✓ It employs top-down approach in program design. ✓ It is based on the problem at hand. Sequence or procedure or functionality is paramount.
10.	**What is a module? What is modular programming paradigm? What are its characteristics?**
Ans.	A set of related procedure with the data they manipulate is called a Module. Modular programming paradigm combines related procedures in a module and hides data under modules. The data are dependent on the functions. The arrangement of data can't be changed without modifying all the functions that access it.
11.	**What is an object? What is a class? How is an object different from a class?**
Ans.	An object is an identifiable entity with some characteristics and behavior. It represents an entity that can store data and its associated functions. A class is a group of objects that share common properties and relationship. It represents a group of similar objects.
12.	**What is object oriented programming paradigm? Name the four basic concepts of OOP.**
Ans.	We can define the object oriented programming (OOP) paradigm as following: Decide which classes and objects are needed; provide a full set of operations for each class. Following are the basic OOP concepts:

	1. Data Abstraction 2. Data Encapsulation 3. Modularity 4. Inheritance 5. Polymorphism
13.	**What is meant by Abstraction?**
Ans.	Abstraction is the act of representing essential features without including the background details.
14.	**The real world concept gets simplified using concept of abstraction. True or False?**
Ans.	True.
15.	**Give an example to illustrate the concept of abstraction.**
Ans.	In switch board we only press certain switches according to our requirement. What is happening inside, how it is happening etc. we need not know. So, this is abstraction, we know only the essential things to operate on switch board without knowing the background details of switchboard.
16.	**What is encapsulation? Why data is considered safe if encapsulated?**
Ans.	Encapsulation is the way of combining both data and the functions that operate on the data under a single unit. In encapsulation, data can't access directly. The only way to access the data is provided by the functions. The data is hidden, so data is considered safe if encapsulated
17.	**How are the terms abstraction and encapsulation related?**
Ans.	Encapsulation is the way of implementing abstraction. Thus, the terms abstraction and encapsulation are related because while implementing encapsulation, abstraction is also implemented.
18.	**Why classes are called Abstract Data Type?**
Ans.	Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT).
19.	**What is a base class? What is a subclass? What is the relationship between a base class and subclass?**
Ans.	A subclass is a class that inherits properties from some other class. A base class is a class whose properties are inherited by subclass. A subclass has nearly all the properties of base class but the reverse of it is not true.
20.	**What is modularity? What benefits does it offer?**
Ans.	Modularity is the property of decomposing a system into a set of cohesive and loosely coupled modules. It offers following benefits: (i) It reduces its complexity to some degree. (ii) It creates a number of well-defined, documented boundaries within the program.
21.	**Inheritance ensures closeness with the real-world models. True or False?**
Ans.	True.
22.	**How does inheritance support 'reusability'?**
Ans.	Inheritance supports 'reusability' by allowing the addition of additional features to a existing class without modifying it.
23.	**What do you mean by the transitive nature of inheritance?**
Ans.	The transitive nature of inheritance states that if a class A inherits properties from its base class B then all its subclasses will also be inheriting the properties of base class of A i.e., B.
24.	**A subclass defines all the features of base class and its additional features. True or False?**
Ans.	False
25.	**A subclass inherits all the properties of its base class and vice versa. True or False?**
Ans.	False
26.	**What are object-based languages? Give an example of object-based language.**
Ans.	Languages that support programming with objects are said to be object-based programming languages. They do not support inheritance and dynamic binding. Ada is a typical object-based programming language.
27.	**What is polymorphism? Give an example illustrating polymorphism.**
Ans.	Polymorphism is the ability for a message or data to be processed in more than one form. For example, consider an addition operation. It shows different behavior in different types of data. For two numbers, it will generate a sum. The numbers may integers or float. Thus the addition for integer is different from the addition to floats.

TYPE B : SHORT ANSWER QUESTIONS

1.	**Briefly explain the two major types of programming languages.**
Ans.	Low Level language: These are machine-oriented and require extensive knowledge of computer circuitry. In machine language, the instructions are given in binary codes whereas in assembly language, instructions are

	given in symbolic names. High Level language: High level languages like BASIC, PASCAL etc. offer English like keywords, programming constructs for selection and iteration and allow use of variables and constants. Programmers feel much at ease while working in HLLs as compared to low level languages.
2.	**Write a short note on programming in two major types of languages.**
Ans.	Same as above Q. No. 1
3.	**What are programming paradigms? Give names of some popular programming paradigms.**
Ans.	Programming Paradigm: A Programming Paradigm defines the methodology of designing and implementing programs using the key features and building blocks of a programming language. Following are the some popular programming paradigms: (i) Procedural Programming (ii) Modular Programming (iii) Object Oriented Programming
4.	**Write a short note on procedural programming.**
Ans.	The programming approach that focuses on the solution of a problem is known as procedural programming paradigm. This approach emphasis on the 'doing' rather than the 'data'. Languages support this paradigm by providing facilities for passing argument to functions and return values from functions.
5.	**Write a short note on modular programming.**
Ans.	In modular programming, a set of related procedures with the data they manipulate is combined under a uit called module. In modular programming, since many modules access the same data, the way the data is stored becomes critical. The arrangement of the data can't be changed without modifying all the functions that access it.
6.	**What are the shortcomings of procedural and modular programming approaches?**
Ans.	✓ In procedural Programming more emphasis is on doing things. ✓ Procedural Programming is susceptible to design changes. ✓ Procedural Programming leads to increased time and cost overheads during design changes. ✓ In modular programming, the arrangement of the data can't be changed without modifying all the functions that access it. ✓ The problem associated with procedural and modular programming is that their chief components – functions etc. do not model the real world very well.
7.	**Write a short note on OO programming.**
Ans.	OOP stands for Object Oriented Programming. In, Object-Oriented Programming (OOP), the program is organized around the data being operated upon rather than the operations performed. The basic idea behind OOP is to combine both, data and its functions that operate on the data into a single unit called object. Following are the basic OOP concepts: 1. Data Abstraction 2. Data Encapsulation 3. Modularity 4. Inheritance 5. Polymorphism
8.	**How does OOP overcome the shortcomings of traditional programming approaches?**
Ans.	OOP provides the following advantages to overcome the shortcomings of traditional programming approaches: ✓ OOPs is closer to real world model. ✓ Hierarchical relationship among objects can be well-represented through inheritance. ✓ Data can be made hidden or public as per the need. Only the necessary data is exposed enhancing the data security. ✓ Increased modularity adds ease to program development. ✓ Private data is accessible only through designed interface in a way suited to the program.
9.	**What is the difference between object and class? Explain with examples.**
Ans.	An object represents an entity that can store data and its associated functions whereas a class represents a group of similar objects. 'Object' is an instance of 'class'. For example, we can say 'bird' is a class but 'parrot' is an object.
10.	**Explain briefly the concept of data abstraction with the help of an example.**
Ans.	Abstraction refers to the act of representing essential features without including the background details or

	explanations. Abstraction is implemented through public members of a class, as the outside world is given only the essential and necessary information through public members, rest of the things remain hidden. For example, while drive a car, we only knows the essential features to drive a car without including the background details or explanations. This is abstraction.
11.	**Explain briefly the concept of encapsulation with the help of an example.**
Ans.	The wrapping up of data and operations/functions (that operate o the data) into a single unit (called class) is known as Encapsulation. Encapsulation is implemented with the help of a class as a class binds together data and its associated function under one unit. For example, cars and owners. All the functions of cars are encapsulated with the owners, no one else can access it.
12.	**How the data is hidden and safe if encapsulation is implemented? Explain with example.**
Ans.	In encapsulation, data cannot be accessed directly. The only way to access the data is provided by the functions. To read the item in an object, call a member function in the object. It will read the item and return the value. The data is hidden, so it is safe from accidental alteration. The best example of encapsulation is a CLASS because a class hides class variables/functions from outside the class.
13.	**Simulate a daily life example (other than the one mentioned in the chapter) that explains encapsulation.**
Ans.	In daily life, mobile phone explains encapsulation. Mobile phone has some interface which helps us to interact with cell phone and we can uses the services of mobile phone. But the actually working in cell phone is hidden. We don't know how it works internally.
14.	**Write a short note on inheritance.**
Ans.	Inheritance enables us to create new classes that reuse, extend, and modify the behavior that is defined in other classes. The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class. A derived class can have only one direct base class. Inheritance is transitive. When we define a class to derive from another class, the derived class implicitly gains all the members of the base class, except for its constructors and destructors.
15.	**What are the advantages offered by inheritance?**
Ans.	✓ Inheritance ensures the closeness with the real-world models. ✓ Allows the code to be reused as many times as needed. The base class once defined and once it is compiled, it need not be reworked. ✓ We can extend the already made classes by adding some new features. ✓ Inheritance is capable of simulating the transitive nature of real-world's inheritance, which in turn saves on modification time and efforts, if required.
16.	**How is reusability supported by inheritance? Explain with example.**
Ans.	Inheritance allows the addition of additional features to an existing class without modifying it. One can derive a new class from an existing one and add new features to it. For example, we have a class 'Student', and we need to add a new class called 'GraduateStudent'. We derive the new class 'GraduateStudent' from the existing class 'Student' and then all we really need to add the extra features to 'GraduateStudent' that describe the difference between students and graduate students.
17.	**What is the benefit of transitive nature of inheritance? Explain with example.**
Ans.	Suppose we inherit class B from existing class A. the classes C and D inherit from class B. later we find that class A has a bug that must be corrected. After correcting the bug in A, it automatically will be reflected across all classes that inherit from A, if the class A has been inherited without changes. The benefit of being transitive is reduction in amount of efforts that one would have done if each class inherited from A was to be modified separately.
18.	**What is polymorphism? Explain with example.**
Ans.	Polymorphism is the ability for a message or data to be processed in more than one form. It is a property by which the several different objects respond in a different way to the same message. Suppose we have three different classes called rectangle, circle and triangle. Each class contains a draw() method to draw the relevant shape on the screen. When you call a draw method through a function call, the required draw() will be executed depending on the class, i.e., rectangle, circle or triangle.
19.	**Do you think OOP is more closer to real world problems? Why? How?**
Ans.	Yes, OOP is more closer to real world problems because object oriented programming implement inheritance which has capability to express the inheritance relationship which makes it ensure the closeness with the real-

	world models. Object oriented programming implement inheritance by allowing one class to inherit from another. By implementing inheritance, real-world relations among objects can be represented programmatically.

TYPE C : LONG ANSWER QUESTIONS

1.	**Write a note on software evolution.**
Ans.	A program serves the purpose of commanding the computer. The two major types of programming languages are as follows: (a) Low Level Languages: These are machine-oriented languages and require extensive knowledge of computer circuitry. Machine language and Assembly language are low level languages. The low level languages provide a vehicle for the programmer to specify actions to be executed, so they are closer to machine. (b) High Level Languages: Offers English like keywords constructs for sequence, selection and iteration and use of variables and constants. High level languages are considered close to the programmer because it provides a set of concepts for the programmer to use when thinking about what can be done. 'C++' is both "close to the machine" and "close to the programmer", so it is called 'middle level language'. Programming Paradigm: Paradigm means organizing principle of program. It is an approach to programming. Following are there different programming paradigms: (i) Procedural Programming: Decide which procedure you want; use the best algorithm you can find. (ii) Modular Programming: Decide which module you want; partition the program so that data is hidden in modules. (iii) Object Oriented Programming: Decide which classes and objects are needed; provide a full set of operations for each class.
2.	**Explain different programming paradigms, their shortcomings etc. with appropriate examples.**
Ans.	Following are the different programming paradigms: (i) Procedural Programming: Procedural programming paradigm lays more emphasis on procedure or the algorithm. Data is considered secondary. Data is loosely available to many functions. The problem associated with procedural and modular programming is that their chief components – functions etc. do not model the real world very well. For instance, a procedural program for library maintenance aims at the operations Issue, Return etc. whereas real world entities are Books. (ii) Modular Programming: In modular programming, a set of related procedures with the data they manipulate is combined under a unit called module. In this paradigm, since many modules access the same data, the way the data is stored becomes critical. The arrangement of the data can't be changed without modifying all the functions that access it. Another drawback is same as above in procedural programming. (iii) Object Oriented Programming: The object oriented approach views a problem in terms of objects involved rather than procedure for doing it. OOP implements inheritance, so that real-world relations among objects can be represented programmatically. The problem associated with OOP is that OOP programs' design is tricky and to program with OOP, programmer need proper skills such as design skills, programming skills, thinking in terms of objects etc.
3.	**Explain the basic concepts of OOP with examples.**
Ans.	Following are the general OOP concepts: **1. Data Abstraction:** Data abstraction means, providing only essential information to the outside word and hiding their background details i.e. to represent the needed information in program without presenting the details. For example, while drive a car, we only know the essential things to drive a car without including the background details or explanations. This is abstraction. **2. Data Encapsulation:** The wrapping up of data and operations/functions (that operate o the data) into a single unit (called class) is known as Encapsulation. For example, in a big company, there are so many departments. Suppose an employee in the production dept. wants to know about purchased material. The production dept employee will have to issue a memo to the 'purchase' requesting for the required information. Then some employee of the 'purchase' dept. will send the replay with the asked information. Here, we can say that

	'Department data and department employees are encapsulated into a single entity, the department'. **3. Modularity:** Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules. For example, the music system comprises of speakers, cassette-player, record-player, cd-player etc. Now, these parts are complete units in themselves, yet they are a subpart of the music system. This is modularity. **4. Inheritance:** Inheritance is the capability of one class of things to inherit capabilities or properties from another class. For example, humans inherit from the class 'Humans' certain properties, such as ability to speak, eat, drink etc. The class 'Human' inherits these properties from the class 'Mummal' which again inherits some of its properties from another class 'Animal'. **5. Polymorphism:** Polymorphism is the ability for a message or data to be processed in more than one form. For example, 5 + 7 results into 12, the sum of 5 and 7. And 'A' + 'B' results into 'AB', the concatenated strings. The same operation symbol '+' is able to distinguish between the two operations depending upon the data type it is working on.