

ACHIEVER'S EDUCATION CONSULTANT
08141710097
SUMMARY OF CIT383 WITH PAST QUESTIONS

Introduction

The course presents background concepts of objects and describes a widely used programming methodology in object-oriented design . In OOD, the first step in the problem solving process is to identify the components called objects, which form the basis of the solutions and determine how these objects interact with one another. This course is divided into three modules. Another concept discussed in the module is the static behaviour of classes.

Inheritance and polymorphism are addressed in details with examples and possible output. In module 3, units 1 and 2 are discussed using Java while with C sharp is used in units 3-5. This is done because Java does not support operator overloading. Overloading is discussed in details.

Others include overloading true and false, logical operator overloading indexers. This course is aimed at enhancing your former knowledge in programming by introducing objects to improve your knowledge in writing program through possible use of objects and classes. Some of the concepts treated in the course require you to have some basic background on some topics in computer science, especially programming course. There is the need to have a fore-knowledge of programming syntax.

Therefore, you are advised to read through further reading to enhance the knowledge you will acquire from this course material.

Course Aims

Introduce the basic concepts of object-oriented programming. Discuss in details overloading focussing on the different type of overloading. Analyse static methods and fields associated with an entire class rather than specific instances of the class. Learn how to create a class that inherits fields and methods from another class.

Working through This Course

This course is designed to cover approximately 16 weeks, and it will require a thorough understanding of the concepts. You should do the exercises in the Tutor-Marked Assignments and submit to your tutors.

Declarative Programming in a Prototype-Instance System

Ken Arnold & James Gosling .

Assignment File

The self-assessment exercises will enable you monitor your performance by yourself, while the Tutor-Marked Assignment is a supervised assignment. The assignments take a certain percentage of your total score in this course. The Tutor-Marked Assignments will be assessed by your tutor within a specified period. At the end of the course, the examination will test your understanding as regards the concept and principle of the subject matter.

This course includes 15 Tutor-Marked Assignments and each must be done and submitted accordingly. Your best scores however, will be recorded for you. Be sure to send these assignments to your tutor before the deadline to avoid loss of marks.

Presentation Schedule

The Presentation Schedule included in your course materials gives you the important dates for the completion of Tutor- Marked Assignments and attending tutorials. Remember, you are required to submit all your assignments by the due date. You should guard against lagging behind in your work. The assignments must be submitted to your tutor for formal assessment in accordance with the deadlines stated in the Assignment File.

The work you submit to your tutor for assessment will count for 30% of your total course mark. At the end of the course, you will need to sit for a final three-hour examination.

Tutor -Marked Assignment

You need to submit all the assignments. The total marks for the best four assignments will be 30% of your total course mark. Assignment questions for the units in this course are contained in the Assignment File. After completing the assignment, you should send it together with the form to your tutor.

Make sure that each assignment reaches your tutor on or before the deadline given. If, however, you cannot complete your work on time, contact your tutor before the assignment is done to discuss the possibility of an extension.

Final Examination and Grading

The examination will cover every aspect of the course, so you are advised to revise all your corrected assignments before the examination. This means that you teach yourself and that you learn, as your learning capabilities would allow. It also means that you are in a better position to determine and to ascertain the what, the how, and the when of your learning. The course units are similarly designed with the introduction following the table of contents, then a set of objectives and then the dialogue and so on.

The objectives guide you as you go through the units to ascertain your knowledge of the required terms and expressions. Viii This table shows how the actual course marking is broken down.

[...]

In distance learning the study units replace the university lecturer. Think of it as reading the lecture instead of listening to a lecturer. In the same way that a lecturer might set you some reading to do, the study units tell you when to read your set books or other material. Just as a lecturer might give you an in-class exercise, your study units provide exercises for you to do at appropriate points.

The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next is a set of learning objectives. These objectives enable you know what you should be able to do by the time you have completed the unit. You should use these objectives to guide your study.

When you have finished the units you must go back and check whether you have achieved the objectives. If you make a habit of doing this you will significantly improve your chances of passing the course. Remember that your tutor's job is to assist you. When you need help, don't hesitate to call and ask your tutor to provide it.

Note the time you are expected to spend on each unit and how the assignments relate to the units. Whatever method you choose to use, you should decide on it and write in your own dates for working on each unit. Once you have created your own study schedule, do everything you can to stick to it. You will always need both the study unit you are working on and one of your set of books on your desk at the same time.

The content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit you will be instructed to read sections from your set books or other articles. Use the unit to guide your reading. Review the objectives for each study unit to confirm that you have achieved them.

When you are confident that you have achieved a unit's objectives, you can then start on the next unit. Proceed unit by unit through the course and try to pace your study so that you keep yourself on schedule. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutormarked assignment form and also written on the assignment.

Check that you have achieved the unit objectives and the course objectives .

Facilitators/Tutors and Tutorials

There are 15 hours of tutorials provided in support of this course. You will be notified of the dates, times and location of these tutorials, together with the name and phone number of your tutor, as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course. You must mail or submit your Tutor-Marked Assignments to your tutor well before the due date.

They will be marked by your tutor and returned to you as soon as possible. The following might

be circumstances in which you would find help necessary. You should try your best to attend the tutorials. You can raise any problem encountered in the course of your study.

You will learn a lot from participating in discussions actively. Introduction to object-oriented programming will introduce you to the concepts of objects-data structures consisting of data fields and methods and their interactions to design applications and computer programs. The methodology focuses on data rather than processes, with programs composed of self-sufficient modules, each containing all information needed to manipulate its own data structure. Concrete examples with illustrations to help you get the best from the course are given.

Therefore, I wish you all the best you can get with the hope that you will understand and find the course interesting.

[...]

Programming techniques may include features such as information hiding, data abstraction, encapsulation, modularity, polymorphism, and inheritance. It was not commonly used in mainstream software application development until the early 1990s. The methodology focuses on data rather than processes, with programmes composed of self-sufficient modules each containing all the information needed to manipulate its own data structure. The actions on these objects are closely associated with the object.

For example, the data structures tend to carry their own operators around with them . Define the term classes list object-oriented programming concepts describe objects enumerate how to access a class member. Objects are keys to understanding object-oriented technology. Once created, an object can be easily passed around inside the system.

This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code. This is analogous to fixing mechanical problems in the real world.

Object Creation and Destruction

Once a class has been specified, a data type exists with the same name. You do not need to destroy or remove an object when it is no longer needed. Components of a class are called the members of the class. Where modifier are used to alter the behaviour of the class and, usually, class members consist of named constants, variable declarations, and/ or methods.

Note

If a member of a class is a named constant, you declare it just like any other named constant.

If a member of a class is a method, you define it just like any other method

The data members of a class are also called fields. These will be discussed later. Each bicycle was built from the same set of blueprints and therefore contains the same components. In

objectoriented terms, we say that your bicycle is an instance of the class of objects known as bicycles.

A class is the blueprint from which individual objects are created. You may have noticed that the Bicycle class does not contain a main method. Once an object of a class is created, the object can access the members of the class. If the object is created in the definition of a method of the class, then the object can access both the public and private members.

If the object is created elsewhere then the object can access only the public members of the class. A class specifies the properties and methods that objects can work with. It is a template or prototype for each of the many objects made to the class design. The default is package or visible within the current package only.

The second optional group indicates the capability of a class to be inherited or extended by other classes. Abstract classes must be extended and final classes can never be extended by inheritance. The default indicates that the class may or may not be extended at the programmers' discretion. The third option of extends is described in the tutorial on inheritance.

The fourth option of implements is described in the tutorial on interfaces. The box has length, width and height properties as well as a method for displaying its volume. A class file can contain more than one associated class but normally its filename is that of the first defined file. A driver program is named the same as the class that contains the main.

Its file may contain other classes as well. Properties are sometimes called field variables or states.

The items in the first optional group indicate the visibility or accessibility from other objects; public means visible everywhere

The third optional group indicates how long a value is retained in the variable. Static indicates that the value is shared by all members of the class and exists for all runtime. Static properties can be referenced without creating an instance of the class. Transient prevents the variable from being transferred during a serial operation such as file i/o.

Volatile is used in multithreading to prevent overwrite issues. Many programmers make all properties private and force access through public accessors and mutators which can include validation steps. Class behaviour is represented in Java by methods. The default is package or visible within the current package only.

Static methods are shared by all members and exist for all runtime. Static methods can be referenced without creating an instance of the class. Abstract methods must be redefined on inheritance. Native methods are written in C but accessible from Java.

The return data type defines the type of value that the calling routine receives from the object. It can be any of the primitive types or the reserved word void if no message is to be returned. The parameter list can contain from zero to many entries of datatype varName pairs. Parameters are

passed by value, thus upholding the encapsulation principle by not allowing unexpected changes or side effects.

The projects page has some simple problems based on array passing. This allows objects to start with values appropriate to use. The best programming device is to declare the constructor with no parameters as private and use it to initialise all properties. Then other constructors can first call it using this and then do their own specific property validations/initialisation.

The exceptions to writing accessor/mutator methods for each property is for those that are used only within the class itself or for properties that are set in more complex ways. Helper methods are those routines that are useful within the class methods but not outside the class. Recursive methods are methods that are defined in terms of it.

Int factorial

Bad input should always be trapped. Inner classes are classes nested inside another class. They have access to the outer class fields and methods even if marked as private. Inner classes are used primarily for data structures, helper classes, and event handlers.

These variables are called instance variables. Methods within the class manipulate these variables. Most instance variable declarations are preceded with the private access modifier. Variables or methods declared with access modifier private are accessible only to methods of the class in which they are declared.

Classes often provide public methods that allow clients of the class to set or get private instance variables. The names of these methods need not begin with set or get, but this naming convention is highly recommended in Java.

Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication components of a class are called the members of the class

Since this is such a short program, you may want to use only one file say MakeCircle.java that contains both the main class and the Circle class. The driver creates an instance of the Circle class and then displays the diameter and area.

The diameter should read as 6.0 and the area as 28.25999. Once it is working, see if you can factor it into two files and compile separately, .

PASSING MESSAGE AND ENCAPSULATION

Outline what message passing entails define encapsulation describe the concept of encapsulation list the uses of message passing. Such messaging is used in Web Services by SOAP. This concept is the higher-level version of a datagram, except that messages can be larger than a packet and can optionally be made reliable, durable, secure, and/or transacted.

[...]

In the terminology of some object-oriented programming languages, a message is the single means to pass control to an object. Sending the same message to an object twice will usually result in the object applying the method twice. Two messages are considered to be the same message type, if the name and the arguments of the message are identical. Objects can send messages to other objects from within their method bodies.

In OOD, objects combines data and operations on that data in a single unit, the feature called encapsulation. The object is first identified, then the relevant data and then the operations which are needed to manipulate the objects. Encapsulation is the ability of an object to be a container for related properties and methods . Older languages did not enforce any property/method relationships.

Encapsulation is one of the three fundamental principles in object- oriented programming. Data hiding is the ability of objects to shield variables from external access. Those variables marked as private can only be seen or modified through the use of public accessor and mutator methods. This permits validity checking at run time.

Access to other variables can be allowed but with tight control on how it is done. Methods can also be completely hidden from external use. Encapsulation is the concept of hiding the implementation details of a class and allowing access to the class through a public interface. For this, there is need to declare the instance variables of the class as private or protected.

The client code should access only the public methods rather than accessing the data directly. Also, the methods should follow the Java Bean's naming convention of set and get. Encapsulation makes it easy to maintain and modify code. The client code is not affected when the internal implementation of the code changes as long as the public method signatures are unchanged.

Message passing is a concept more important than objects in Alan Kay's view of object-oriented programming, however people often miss the point and place too much emphasis on objects themselves and not enough on the messages being sent between them. Encapsulation is the ability of an object to be a container related properties and methods. Messages are also commonly used in the same sense as a means of inter-process communication. This permits validity checking at run time Encapsulation makes it easy to maintain and modify code. List some examples of passing message style. Write a program to explain encapsulation. How does message passing affect other programming model.

INHERITANCE

In Java, the mechanism that allows us to extend the definition of a class without making any physical changes to the class is the principle of inheritance. Inheritance has to do with relationship. Define inheritance differentiate between a subclass and a super class outline the uses of a super class differentiate between inheritance and overridden. Inheritance is the capability of a class to use the properties and methods of another class while adding its own functionality.

An example of where this could be useful is with an employee records system. One could create a generic employee class with states and actions that are common to all employees. Then more specific classes could be defined for salaried, commissioned and hourly employees. The generic class is known as the super class or base class and the specific classes as subclasses or derived classes.

The concept of inheritance greatly enhances the ability to reuse code as well as making design a much simpler and cleaner process. For instance, once the problem domain is partitioned into types, one will likely want to model relationships in which one type is a more specific or specialised version of another. For example you may have identified in your problem domain two types, Cup and CoffeeCup, and you want to be able to express in your solution that a CoffeeCup is a more specific kind of Cup. Inheritance allows you to build hierarchies of classes, such as the one shown in Figure 3.1.

The upside-down tree structure shown in Figure 51 is an example of an inheritance hierarchy displayed in UML form. Note that the classes become increasingly more specific as you traverse down the tree.

The is-a Relationship of Inheritance

When two classes are right next to each other in the inheritance hierarchy, their relationship is said to be direct. The act of declaring a direct subclass is referred to in Java circles as class extension. For example, a Java guru might be overheard saying, “Class CoffeeCup extends class Cup.

English language, Java in-the-knows may also employ the term

” One other way to say the same thing is, “Class CoffeeCup descends from class Cup. An inheritance hierarchy, such as the one shown in Figure 3.1, defines a family of types. The most general class in a family of types, the one at the root of the inheritance hierarchy is called the base class. In Figure, the base class is Cup.

Because every class defines a new type, you can use the word “type” in many places you can use “class. If a class is declared with no extends clause, it by default extends the Object class. The inheritance hierarchy of Figure 3.1 could also have shown the Object class hovering above the Cup class, in its rightful place as the most super of all superclasses. In this case, class Object remained invisible, because the purpose of the figure was to focus on one particular family of types, the Cup family.

In Java, a class can have only one direct superclass. In object-oriented parlance, this is referred to as single inheritance. It contrasts with a multiple inheritance, in which a class can have multiple direct superclasses. Modeling is-a relationship is called inheritance because the subclass inherits the interface and, by default, the implementation of the superclass.

Inheritance of interface guarantees that a subclass can accept all the same messages as its

superclass. A subclass object can, in fact, be used anywhere a superclass object is called for. For example, a `CoffeeCup` as defined in Figure 3.1 can be used anywhere a `Cup` is needed. In a Java program, this means you can invoke on a subclass object any method you can invoke on the superclass object.

A subclass has one other way to specialise besides overriding the implementation of methods that exist in its direct superclass. It can also extend the superclass's interface by adding new methods. This possibility will be discussed in detail later.

Suppose there is a method in class `Cup` with the following signature: `public void addLiquid`
{

The `addLiquid` method could be invoked on any `Cup` object. Because `CoffeeCup` descends from `Cup`, the `addLiquid` method could also be invoked on any `CoffeeCup` object. So far, you have only been introduced to two access levels, public and private. In the case of the `addLiquid` method, because class `Cup` declares it with public access, class `CoffeeCup` must declare it public also.

If `CoffeeCup` attempted to override `addLiquid` with any other access level, class `CoffeeCup` would not compile. For an illustration of the difference between inheriting and overriding the implementation of a method, see Figure 3.2. Liquid added to an instance of the `CoffeeCup` class defined on the left will also swirl clockwise, because that `CoffeeCup` inherits `Cup`'s implementation of `addLiquid`, which swirls clockwise. By contrast, liquid added to an instance of the `CoffeeCup` class defined on the right will swirl counterclockwise, because this `CoffeeCup` class overrides `Cup`'s implementation with one of its own.

In addition to the bodies of public methods, the implementation of a class includes any private methods and any fields defined in the class. Using the official Java meaning of the term “inherit,” a subclass does not inherit private members of its superclass. Well- designed classes most often refuse other classes direct access to their non-constant fields and this policy generally extends to subclasses as well. If a superclass has private fields, those fields will be part of the object data in its subclasses, but they will not be “inherited” by the subclass.

Methods defined in the subclasses will not be able to directly access them. You can access the superclass's version by qualifying the simple name with the `super` keyword, as in `super.fieldName`. If your customers subclass the classes in your library, you will likely have no idea what new fields they have declared in their subclasses. Java's willingness to tolerate hidden fields makes subclasses more accepting of changes in their superclasses.

As you perform an object-oriented design, you may come across classes of objects that you would never want to instantiate. An example of such a class might be the `Liquid` class from the previous discussions. Class `Liquid` served as a base class for the family of types that included subclasses `Coffee`, `Milk`, and `Tea`. Java provides a way to declare a class as conceptual only, not one that represents actual objects, but one that represents a category of types.

Such classes are called abstract classes. Neither the Java compiler nor the Java Virtual Machine

will allow an abstract class to be instantiated. Any class with one or more abstract methods is itself abstract and must be declared as such. In the Liquid class, you may decide that there is no such thing as a default swirling behaviour that all liquids share.

Any subclasses that descend from the Liquid class shown above will have to either implement swirl or declare themselves abstract. For example, if you decided there were so many varieties of coffee that there is no sensible default implementation for Coffee, you could neglect to implement swirl in Coffee. In that case, however, you would need to declare Coffee abstract. If you didn't, you would get a compiler error when you attempted to compile the Coffee class.

You would have to subclass Coffee and implement swirl in the subclasses, if you wanted the Coffee type to ever see any action. Most often you will place abstract classes at the upper regions of your inheritance hierarchy, and non-abstract classes at the bottom. Nevertheless, Java does allow you to declare an abstract subclass of a non-abstract superclass. For example, you can declare a method inherited from a non-abstract superclass as abstract in the subclass, thereby rendering the method abstract at that point in the inheritance hierarchy.

This design implies that the default implementation of the method is not applicable to that section of the hierarchy. As long as you implement the method again further down the hierarchy, this design would yield an abstract class sandwiched in the inheritance hierarchy between a non-abstract superclass and non-abstract subclasses. Inheritance can be viewed as a hierarchical structure wherein a superclass is declared with its subclass. Rather than creating completely new classes from scratch, inheritance can be used to reduce software complexity.

This unit shows you how to model relationships using composition and inheritance. It describes many facets of inheritance in Java, analyse the flexibility and performance implications of inheritance and composition with guidelines on the appropriate use of each. The generic class is known as the superclass or base class and the specific classes as subclasses or derived classes. When a class has only one direct super class, it is referred to as single inheritance. In contrast with multiple inheritance, in which a class can have multiple direct superclasses. Well-designed classes most often refuse other classes direct access to their non-constant fields, and this policy generally extends to subclasses as well. If a super class has private fields, those fields will be part of the object data in its subclasses, but they will not be «inherited» by the subclass.

Suppose that a class Employee is derived from the class Person. Give examples of data and method members that can be added to the class Employee.

POLYMORPHISM AND MODULATION

This is the third basic principle of object-oriented programming. Java allows one to treat an object of a subclass as an object of its superclass. In other words, a reference variable of a superclass can point to an object of its subclass. The statement in line 1 declares name and nameRef to be reference variables of the type Person.

The statement in line 2 declares employee and employeeRef to be reference variables of the type

PartTimeEmployee. The statement in line 3 instantiates the object name while the statement in line 4 instantiates the object employees. After executing the statement in line 5, the object nameRef is treated as an object of the class PartTimeEmployee. This is because when the statement in line 6 executes to output nameRef, the method toString of the class PartTimeEmployee executes, not the method toString of the class Person.

In a class of hierarchy, several methods may have the same name and the same formal parameter list. Moreover, a reference variable of a class can refer to either an object of its own class or an object of its subclass. Therefore, a reference variable can invoke, that is, execute, a method of its own class or of its subclass. In early binding, a method's definition is associated with the method's invocation at execution time, that is, when the method is executed.

Except for a few cases, java uses late binding for all methods. Furthermore, the term polymorphism means assigning multiple meanings to the same method name. The reference variable name or nameRef can point to any object of the class Person or the class PartTimeEmployee. Loosely speaking, one could say that these reference variables have many forms, that is, they are polymorphism reference variables.

They can refer to either objects of their own class or objects of the subclasses inherited from their class.



National Open University of Nigeria
Plot 91, Cadastral Zone, Nnamdi Azikiwe Expressway, Jabi - Abuja
Faculty of Sciences
NOVEMBER, 2018 EXAMINATIONS

COURSE CODE: CIT 383

COURSE TITLE: INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

CREDIT: 2 Units

TIME ALLOWED: 2 Hours

INSTRUCTION: Answer Question ONE (1) and any other THREE (3) Questions

QUESTION ONE

- 1a. Describe four (4) benefits of bundling code into individual software objects **[8 marks]**
- b. Explain in details what message passing entails and list the corresponding examples **[10 marks]**
- c. Describe in details the meaning of hiding fields in Java **[7 marks]**

QUESTION TWO

2. a. What is modularity? **[5 marks]**

- b. With the aid of a well labelled diagram illustrate the concept of modular programming [5 marks]
- c. State five (5) advantages modular programming [5 marks]

QUESTION THREE

- 3a. Give an explicit description of the term “Interface” within the context of Java programming language [5 marks]
- b. Write a non-executable class that has a method. This method prints a line of text that reads “Object- Oriented Programming is superb”. [7 marks]
- c. What is argument constructor? [3 marks]

QUESTION FOUR

- 4a. Create a class that has a **String** instance variable and an **int** instance variable, in the class, create a constructor that has parameters that uses these instance variables as its data. [10 marks]
- b. List five (5) Java static predefined methods that are used to perform common mathematical calculations [5 marks]

QUESTION FIVE

- 5a. Define static method? [3 marks]
- b. State and explain three (3) motivations for modularising a program into methods [6 marks]
- c. What is recursive method? Explain in detail [6 marks]