****

**ASSIGNMENT COVER PAGE**

**Please fill in all the required details for your assignment to be accepted.**

|  |  |  |  |
| --- | --- | --- | --- |
| Student’s Name | ALAKU EMMANUEL AVRE | | |
| Student’s Matric No | LUC-NGA-002-ADM-1000421 | | |
| Year/Semester | 2023/SECOND SEMESTER | | |
| Program | COMPUTER SCIENCE/CYBER SECURITY | | |
| Subject Name / Subject Code | OBJECT ORIENTED | | |
| Lecturer’s Name | MR MONSOR | | |
| Assignment Title | Object oriented | | |
| No. of Page (excluding this page) | 13 | | |
| Required words |  | Actual # of words |  |
| Soft copy included | Yes / No | | |
| **DECLARATION BY STUDENTS:**  *I certify that this assignment is my own work in my own words. All resources have been acknowledged and the content has not been previously submitted for assessment to LINCOLN or elsewhere. I also confirm that I have kept a copy of this assignment.*  Sign: Date: 2/06/2023 | | | |

**Functional Programming**

In recent years, JavaScript, a functional programming language that has accumulated more popularity recently, has brought attention to the functional programming paradigm.

Language independence is one of the benefits of functional programming, which has its roots in mathematics. Executing a series of mathematical functions is the key principle of this paradigm.

Functional Programming

This type of programming style is declarative. In contrast to an imperative style, which emphasizes "how to solve", this style focuses on "what to solve".

Rather than using statements, it uses expressions. Expressions are evaluated to produce values, while statements are executed to assign variables.

**Aspects of Functional Programming**

**1. Pure Functions**

There are two main characteristics of these functions. Their output is the same for the same arguments regardless of any other factor.

The second advantage is that they do not have any side-effects, that is, they do not modify any arguments or local/global variables or input/output streams.

Immutability is one of the more recent properties. The value returned by a pure function is its only result. The results are deterministic.

The function-based programming approach makes debugging programs easy due to the lack of side effects and hidden input/output. Using pure functions also simplifies the process of writing parallel/concurrent applications.

As long as the input does not change, a smart compiler can parallelize the instructions, evaluate results when needed, and memorize the results since the results never change as long as the input does not.

The following is an example of a pure function:

// sum is function taking x and y as arguments

const sum = (x, y) => {

// sum is returning sum of x and y without changing them

return x + y;

};

//always returns same result given same inputs

sum(5, 5);

**2. Recursion**

Unlike the other programming paradigm, functional programming does not employ for and while loops. Recursion is used instead for iteration in these languages. Recursive functions are used to execute recursion, which calls themselves continually until the base case is obtained.

public class Recursion {

static int factorial(int n) {

if (n == 1)

return 1;

else

return (n \* factorial(n - 1));

}

public static void main(String[] args) {

System.out.println("Factorial of 5 is: " + factorial(5));

}

}

**3. Referential Transparency**

In functional programming, variables cannot change their values once they have been defined. As a result, there is referential transparency. By using this method, the same language expression will produce the same result.

Functional programming does not use assignment statements. It is necessary to define new variables in a functional paradigm program in order to store additional values. At any given moment, a variable's state will never change in such a program.

Since variables may be replaced with their actual values at any time during the execution of the program, Referential transparency eliminates the slightest possibility of undesirable effects.

**Example:**The string.replace() method is referentially transparent, since using jane.replace(‘j’, ‘J’) will always result in the same result because replacing returns the new object rather than updating the original.

**4. Functions are First-Class and Can be Higher-Order**

The functional programming style treats functions as variables. Thus, they qualify as first-class functions. A data structure is a collection of characteristics that can be passed around as parameters or returned from functions or stored in data structures.

Functions that take other functions as arguments or return functions are higher-order functions. Functions that are first-class in functional programming languages can be higher-order functions.

**5. Immutability**

In the case of variables, they are immutable, meaning they cannot be modified once they've been initialized. A new variable, however, can be created. Varables are immutable, so they maintain state throughout the program.

// class is declared final

final class Immutable {

// private class members

private String name;

private int date;

Immutable(String name, int date) {

// class members are initialized using constructor

this.name = name;

this.date = date;

}

// getter method returns the copy of class members

public String getName() {

return name;

}

public int getDate() {

return date;

}

}

class Main {

public static void main(String[] args) {

// create object of Immutable

Immutable obj = new Immutable("Programiz", 2011);

System.out.println("Name: " + obj.getName());

System.out.println("Date: " + obj.getDate());

}

}

**Pros of Functional Programming**

* The reason why functional programs are easier to verify is because their states do not change during their execution. Functional programs can thus be modeled and reasoned more easily this way.
* In Concurrent programming, multiple tasks are executed concurrently (not neccessarily simultaneously) while in parallel programming, mulitple tasks are executed parallely (simultaneously).
* Compilers can optimize and substituting complex expressions into simpler ones easily. Compilers are given more flexibility in rearranging expressions because of referential integrity.
* Because functions are independent of one another, programs are guaranteed to be secure and safe. The problem of leaks of critical information and accidental modifications of data is now over.
* Programs can be expressed elegantly.

**Drawbacks of Functional Programming**

* There will be significant performance degradation for compilers that do not optimize for recursion.
* You need to think mathematically when you write functional programs. It imposes a mathematical model for programs that isn't always convenient to organize ideas. Putting it another way, writing functional programs isn't easy.
* A significant amount of memory may be consumed by immutability since every value occupies an extra space.

**Function-oriented programming languages**

* C++
* Python
* Java
* Lisp
* Haskell
* Julia

**Examples of Applications**

* A mathematical computation relies on it.
* Parallelism or concurrency are required wherever these are needed.

**Procedural Programming**

In programming, it is defined as a language derived from structure programming and based on the use of call procedures. Essentially, procedures are the functions, routines, or subroutines that specify the computational steps that need to be performed.

The process involves breaking a task down into variables and routines by following a series of instructions. Programs can call procedures at any point during execution, either by other procedures or by themselves.

Procedural Programming

Procedural programming can be done in ALGOL, COBOL, BASIC, PASCAL, FORTRAN, and C.As Procedural programming is considered to be less secure than object-oriented programming.

A program is designed using a top-down approach in procedural programming. As a result, it emphasizes the concept of functions and divides large programs into smaller parts.

In terms of procedural programming, there is no complexity. Procedural programming does not introduce access modifiers like object-oriented programming. The subsequent is an instance;

public class Example {

public static void main(String[] commandLine) {

display( "Hello world" );

}

private static void display(String text) {

System.out.println(text);

}

}

**Aspects of procedural programming**

**1. Scoping**

The scoping concept is concerned with setting the locality of entities (objects) within a procedural program. Without explicit permission, scoping prevents non-local variables from being accessed from other procedures.

**2. Modularity**

Using this method, components are broken down into interchangeable chunks of functionality that can be reused and shared.

**3. Parameter Passing**

A Parameters can be passed into a procedure call via this mechanism. It is possible to pass parameters as references or as values.

**Pros of Procedural Programming**

* Parsing through a Procedural program can be implemented with relatively simple compilers and interpreters. It is also easy to learn and practice this approach.
* By modularizing procedural programs, less redundant and productivity is increased.
* Building new modules is easier if concepts are contained and organized in modules that can be combined.
* It is well suited to general-purpose programming.

**Drawbacks of Procedural Programming**

* The data is publicly accessible. Data can be accessed and modified without restriction. As a result, unintended alterations occur and serious mistakes are more likely to be committed.
* Rather than focusing on data organization, this approach emphasizes procedures. Concepts can be obscured by this, making it difficult to map them into code.

**Programs that follow Procedural approach**

* Python
* Dart
* Swift
* Rust
* Bash
* PowerShell

**Object-Oriented Programming**

In computer programming, object-oriented programming organizes/models software design around data or objects instead of functions and logic. It consists of two words, "object" and "oriented".

As defined by the dictionary, an object is an article or entity in the real world. It describes something or someone who is oriented towards a particular kind of entity or thing. Essentially, it is a programming pattern that wraps around an object.

As a programming paradigm, object-oriented programming means representing everything as an object. Programming languages are becoming increasingly object-oriented, and smalltalk is one of them.

OOPs

Programming models based on OOP are considered to be the most popular among developers. Programs that are large, complex, and continuously updated and maintained benefit from this technique.

As a result, abstraction, inheritance, polymorphism, and encapsulation are all fundamental concepts that facilitate the development and maintenance of software. As with object-oriented programming, these are the four pillars.

It is much easier to simulate real-world events with OOPs than with traditional programming languages. Object-Oriented Programming allows us to provide real-world solutions to problems.

Data hiding is provided by OOPs, while global data can be accessed anywhere in a procedure-oriented programming language. Below is an example of OOPs;

import java.io.\*;

class abc {

public static void main(String[] args)

{

System.out.println("abc");

Signup s1 = new Signup();

s1.create(30, "xyz", "xyz@yahoo.com", 'X', XXXXX);

}

}

class Signup {

int userId;

String name;

String emailId;

char gender;

long mob;

public void create(int userId, String name,

String emailId, char gender, long mob)

{

System.out.println("Creating new account...");

this.userId = userId;

this.name = name;

this.emailId = emailId;

this.gender = gender;

this.mob = mob;

System.out.println("Signup successful!");

}

}

**Aspects of OOPs**

**1. Abstraction**

It is a process of hiding implementation details and revealing only what is necessary to the outside world. Objects have interfaces through which the outside world can interact with them. An example would be a user who only knows how to use WhatsApp, but not how to use code or other things that are hidden.

// Abstract class

abstract class Car {

abstract void accelerate();

}

// Concrete class

class Suzuki extends Car {

void accelerate() {

System.out.println("Suzuki::accelerate");

}

}

class Main {

public static void main(String args[]){

Car obj = new Suzuki(); // Car object => contents of Suzuki

obj.accelerate(); // call the method

}

}

**2. Encapsulation**

The term encapsulation refers to the act of wrapping code or methods (properties) and the associated fields or variables into a single unit. It is possible to create variables or fields that can be accessed or updated only by getters or setters, and to hide them from other classes by using the private access modifier.

package encapPrograms;

public class Car {

private String name;

public String getName() {

return this.name;

}

public void setName(String carName) {

this.name = carName;

}

}

To access the carInfo:

class EncapsulationTest {

public static void main(String[] args) {

// Creating object of Car class by using new keyword.

Car obj = new Car();

// setting the value of variable.

obj.setName("Suzuki");

// reading the value of variable.

String carName = obj.getName();

System.out.println(carName);

}

}

**3. Inheritance**

The concept involves a subclass inheriting properties and behaviors from a superclass. It is also called a child class or derived class. In this case, the subclass has access to the methods and attributes of a superclass, and can add more methods and characteristics. For instance, Mercedes-Benz cars inherit the properties of CAR while CAR inherit the properties of VEHICLE.

class Car {

void getCarDetails() {

System.out.println("Car Details...");

}

}

class Suzuki extends Car {

void getSuzukiDetails() {

System.out.println("Suzuki Detail...");

}

}

public class Car {

public static void main(String args[]) {

Suzuki s = new Suzuki();

s.getCarDetails();

s.getSuzukiDetails();

}

}

**4. Polymorphism**

Using the process, a single name can take on a variety of forms. Depending on the context, the same object can behave differently. There are two types of polymorphism in OOP:

* In runtime polymorphism (overriding), a function can override a previous implementation of a method.
* The concept of compile-time polymorphism (overloading) enables multiple implementations of a single function based on the signature of the function.

class Car {

// method to render a shape

public void displayInfo() {

System.out.println("Displaying Car Information...");

}

}

class Suzuki extends Car {

// renders Suzuki

public void displayInfo() {

System.out.println("Displaying Suzuki Info...");

}

}

class SuzukiPro extends Suzuki {

// renders SuzukiPro

public void displayInfo() {

System.out.println("Displaying SuzukiPro Info...");

}

}

class Main {

public static void main(String[] args) {

// create an object of Suzuki

Suzuki s1 = new Suzuki();

s1.displayInfo();

// create an object of SuzukiPro

SuzukiPro s2 = new SuzukiPro();

s2.displayInfo();

}

}

**Pros of Object-Oriented Programming**

* Organizing concepts is a simple process. Programs written in Object-Oriented can map directly to concepts and methods within a problem domain.
* Object-Oriented programs can scale with the right design. It is easy to impose a hierarchy of concepts.
* Interacts with objects and communicates with them using a simple interface.
* The reusability of software components. Reduced redundancy in the codebase.
* Data encapsulation increases resource safety by hiding data.

**Drawbacks of Object-Oriented Programming**

* In order to implement parallelizable programs, some work is required.
* An unreadable esoteric code can result from bad design, which results in unnecessary indirections.
* As a result, programs are often larger than procedural codes.

**Object-oriented programming languages**

* Simula
* Ruby
* Python
* Small Talk
* TypeScript
* Dart
* **Functional vs Procedural vs OOPS**

|  |  |  |
| --- | --- | --- |
| **Functional** | **OOPS** | **Procedural Programming** |
| There is no difference between data and functions; the focus is on the composition of functions. | The main focus of objects is communication among objects; they are composed of methods and attributes. | There is a difference between data and functions, with the focus on procedures (operations). |
| Suitable for concurrent and parallel programming | Scalable software systems are ideal | Suitable for general-purpose programming |
| Data are immutable for functional programming. | Hidden data and methods are possible. | Every context has access to the data, and it can be changed at any time. |
| In a declarative style, the developer describes what your objectives are in code | In imperative style, the developer specifies how to reach the goal through code. | This is an imperative style of programming, in which the developer specifies the code to reach the goal. |
| Modules and functions are used to organize code. | Objects are used to organize code. | Modules and procedures are used to organize the code. |

## ****Difference Between Class And Object:****

There are many differences between object and class. Some differences between object and class are given below:

| **Class** | **Object** |
| --- | --- |
| Class is used as a template for declaring and  creating the objects. | An object is an instance of a class. |
| When a class is created, no memory is allocated. | Objects are allocated memory space whenever they are created. |
| The class has to be declared first and only once. | An object is created many times as per requirement. |
| A class can not be manipulated as they are not available in the memory. | Objects can be manipulated. |
| A class is a logical entity. | An object is a physical entity. |
| It is declared with the class keyword | It is created with a class name in C++ and  with the **new** keywords in Java. |
| Class does not contain any values which  can be associated with the field. | Each object has its own values, which are associated with it. |
| A class is used to bind data as well as methods together as a single unit. | Objects are like a variable of the class. |
| **Syntax:** Declaring Class in C++ is as follows:  class <classname> {}; | **Syntax:** Instantiating an object for a Class in C++ is as follows:  class Student {     public:        void put(){            cout<<“Function Called”<<endl;        }  };   // The class is declared here  int main(){           Student s1;   // Object created           s1.put();  } |
| **Example:** Bike | **Example:** Ducati, Suzuki, Kawasaki |

<?php  
class Fruit {  
  // Properties  
  public $name;  
  public $color;  
  
  // Methods  
  function set\_name($name) {  
    $this->name = $name;  
  }  
  function get\_name() {  
    return $this->name;  
  }  
  function set\_color($color) {  
    $this->color = $color;  
  }  
  function get\_color() {  
    return $this->color;  
  }  
}  
  
$apple = new Fruit();  
$apple->set\_name('Apple');  
$apple->set\_color('Red');  
echo "Name: " . $apple->get\_name();  
echo "<br>";  
echo "Color: " . $apple->get\_color();  
?>