**PROCEDURAL PROGRAMMING**

This is about writing procedures and methods that perform operations on data.

**OBJECT ORIENTED PROGRAMMING (OOP)**

*This is about creating objects that contain both data and methods*

**DON’T REPEAT YOURSELF (DRY)**

*The* ***"Don't Repeat Yourself****"* ***(DRY)*** *principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.*

***“The two main aspects of OOP are classes and objects”***

**CLASSES**

*A class is a template for objects eg. Fruit*

**OBJECTS**

*An object is an instance of a class eg. Mango, Orange, Apple*

***…A class can also be referred to as an object constructor or a blueprint for creating an object…***

**STATIC Methods… VS …PUBLIC Methods**

*A static method can be accessed without creating an object of the class while a public method can only be accessed by creating an object of the class*

public class Main {

// Static method

static void myStaticMethod() {

System.out.println("Static methods can be called without creating objects");

}

// Public method

public void myPublicMethod() {

System.out.println("Public methods must be called by creating objects");

}

// Main method

public static void main(String[] args) {

myStaticMethod(); // Call the static method

// myPublicMethod(); This would compile an error

Main myObj = new Main(); // Create an object of Main

myObj.myPublicMethod(); // Call the public method on the object

}

}

**CONSTRUCTORS**

*A constructor in java is a special method that is used to initialize an object.*

*It is called when an object of a class is created*

public class Main {

int modelYear;

String modelName;

public Main(int year, String name) {

modelYear = year;

modelName = name;

}

public static void main(String[] args) {

Main myCar = new Main(1969, "Mustang");

System.out.println(myCar.modelYear + " " + myCar.modelName);

}

}

// Outputs 1969 Mustang

**MODIFIERS IN JAVA (2 types)**

***Access Modifiers and Non Access Modifiers***

**ACCESS MODIFIERS**

**CLASSES – have just 2 (public and default)**

1. ***Public –*** *the class is accessible by any other class*
2. ***Default*** *– the class is only accessible by other classes in same package. Used when you don’t specify a modifier*

**Attributes, Methods and Constructors – 4 modifiers**

1. ***Public*** *– code is accessible for all classes*
2. ***Private*** *– code is accessible only in the class where it is declared*
3. ***Default*** *– code is accessible by classes in same package. Only used when no modifier was defined*
4. ***Protected*** *– code is accessible in same package and subclasses*

**NON-ACCESS MODIFIERS**

**CLASSES (2 types)**

1. ***Final –*** *cannot be inherited by other classes*
2. ***Abstract –*** *cannot be used to create objects. (to access it, it must be inherited from another class).*

**ATTRIBUTES and METHODS (6 types)**

1. ***Final –*** *cannot be overridden or modified*
2. ***Static –*** *attributes and methods belong to the class rather than the object*
3. ***Abstract –*** *can only be used on methods within an abstract class. The methods will have no body. The body can only be provided by a subclass where it is inherited.*
4. ***Transient –***
5. ***Sychronized –***
6. ***Volatile –***

**Example of static and public**

public class Main {

// Static method

static void myStaticMethod() {

System.out.println("Static methods can be called without creating objects");

}

// Public method

public void myPublicMethod() {

System.out.println("Public methods must be called by creating objects");

}

// Main method

public static void main(String[ ] args) {

myStaticMethod(); // Call the static method

// myPublicMethod(); This would output an error

Main myObj = new Main(); // Create an object of Main

myObj.myPublicMethod(); // Call the public method

}

}

**ENCAPSULATION**

*This refers to the act of hiding sensitive data from users.*

*To achieve this, you must do 2 things*

1. *Declare values of variables/attributes as private*
2. *Provide public* ***GET*** *and* ***SET*** *methods to access and manipulate the values of these private variables*

*The* ***GET Method*** *returns the variable values* ***WHILE*** *the* ***SET Method*** *sets the value of the variables*

public class Person {

private String name; // private = restricted access

// Getter

public String getName() {

return name;

}

// Setter

public void setName(String newName) {

this.name = newName;

}

}

public class Main {

public static void main(String[] args) {

Person myObj = new Person();

myObj.setName("John"); // Set the value of the name variable to "John"

System.out.println(myObj.getName());

}

}

// Outputs "John"

**PACKAGES IN JAVA**

-they are used to group related classes (think of it like a folder).

***-2 types of packages are:***

*1. User created packages*

*2. Built-in packages from the Java API*

***To use a class or a package from the java library you must use the import keyword***

import package.name.Class; // ***Import a single class***

import package.name.\*; // ***Import the whole package***

import java.util.Scanner;

**INHERITANCE**

*This describes the ability to inherit attributes and methods from one class to another*

*2 classes of inheritance are:*

1. ***Superclass (parent)*** – the class being inherited from
2. ***Subclass (child)*** – the class that inherits from the parent

class Vehicle {

protected String brand = "Ford"; // Vehicle attribute

public void honk() { // Vehicle method

System.out.println("Tuut, tuut!");

}

}

class Car extends Vehicle {

private String modelName = "Mustang"; // Car attribute

public static void main(String[] args) {

// Create a myCar object

Car myCar = new Car();

// Call the honk() method (from the Vehicle class) on the myCar object

myCar.honk();

// Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class

System.out.println(myCar.brand + " " + myCar.modelName);

}

}

**FINAL KEYWORD**

***“Once a class in declared with the final keyword it cannot be inherited”***

**POLYMORPHISM**

*Means - many forms, this allows us to perform many different actions with methods inherited from other classes.*

class Animal {

public void animalSound() {

System.out.println("The animal makes a sound");

}

}

class Pig extends Animal {

public void animalSound() {

System.out.println("The pig says: wee wee");

}

}

class Dog extends Animal {

public void animalSound() {

System.out.println("The dog says: bow wow");

}

}

class Main {

public static void main(String[] args) {

Animal myAnimal = new Animal(); // Create a Animal object

Animal myPig = new Pig(); // Create a Pig object

Animal myDog = new Dog(); // Create a Dog object

myAnimal.animalSound();

myPig.animalSound();

myDog.animalSound();

}

}

**ABSTRACTION**

*This is used to hide certain details and show only essential information to the users*

*An abstract class can have both abstract and regular methods*

**INTERFACES**

*An interface is an abstraction class used to group related methods with empty bodies*

***-“it is a way to achieve abstraction”***

***“To access interface methods the interface must be implemented by another class with the implements keyword”***

// Interface

interface Animal {

public void animalSound(); // interface method (does not have a body)

public void sleep(); // interface method (does not have a body)

}

// Pig "implements" the Animal interface

class Pig implements Animal {

public void animalSound() {

// The body of animalSound() is provided here

System.out.println("The pig says: wee wee");

}

public void sleep() {

// The body of sleep() is provided here

System.out.println("Zzz");

}

}

class Main {

public static void main(String[] args) {

Pig myPig = new Pig(); // Create a Pig object

myPig.animalSound();

myPig.sleep();

}

}