**BWT Flutter TASK\_03**

**Introduction to OOP in Dart**

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects," which can contain data in the form of fields (often known as properties or attributes) and code in the form of procedures (often known as methods). Dart is an object-oriented language, and it supports the key OOP principles such as encapsulation, inheritance, Abstraction and polymorphism.

**Classes and Objects**

* **Classes:** A class in Dart is a blueprint for creating objects (instances). It encapsulates data for the object and methods to manipulate that data.

class Person {

String name;

int age;

// Constructor

Person(this.name, this.age);

// Method

void display() {

print('Name: $name, Age: $age');

}

}

* **Objects:** Objects are instances of classes. Each object can have different values for the properties defined in the class.

void main() {

Person person1 = Person('Ali', 30);

person1.display();

}

**Different Types of Constructors**

Dart supports several types of constructors to initialize objects:

* **Default Constructor:** This is a constructor that does not have any parameters.

class Point {

int x;

int y;

Point( ) {

x = 0;

y = 0;

}

}

* **Parameterized Constructor:** This constructor takes parameters to initialize an object with specific values.

class Point {

int x;

int y;

Point(this.x, this.y);

}

* **Named Constructor:** Dart allows named constructors to provide multiple ways to create an instance of a class.

class Point {

int x;

int y;

Point(this.x, this.y);

Point.origin() {

x = 0;

y = 0;

}

}

**Inheritance**

Inheritance is a way to form new classes using classes that have already been defined. In Dart, a class can inherit from another class, gaining access to its properties and methods.

* **Example:**

class Animal {

void makeSound() {

print('Animal makes a sound');

}

}

class Dog extends Animal {

@override

void makeSound() {

print('Dog barks');

}

}

void main() {

Dog dog = Dog();

dog.makeSound(); // Output: Dog barks

}

**Mixins**

Mixins are a way of reusing a class’s code in multiple class hierarchies. A mixin is declared using the mixin keyword and can be used by multiple classes.

* **Example:**

abstract class Animal {}

abstract class Mammal extends Animal {}

abstract class Bird extends Animal {}

abstract class Fish extends Animal {}

mixin Walker {

void walk() {

print("I'm walking");

}

}

mixin Swimmer {

void swim() {

print("I'm swimming");

}

}

mixin Flyer {

void fly() {

print("I'm flying");

}

}

class Dolphin extends Mammal with Swimmer {}

class Bat extends Mammal with Walker, Flyer {}

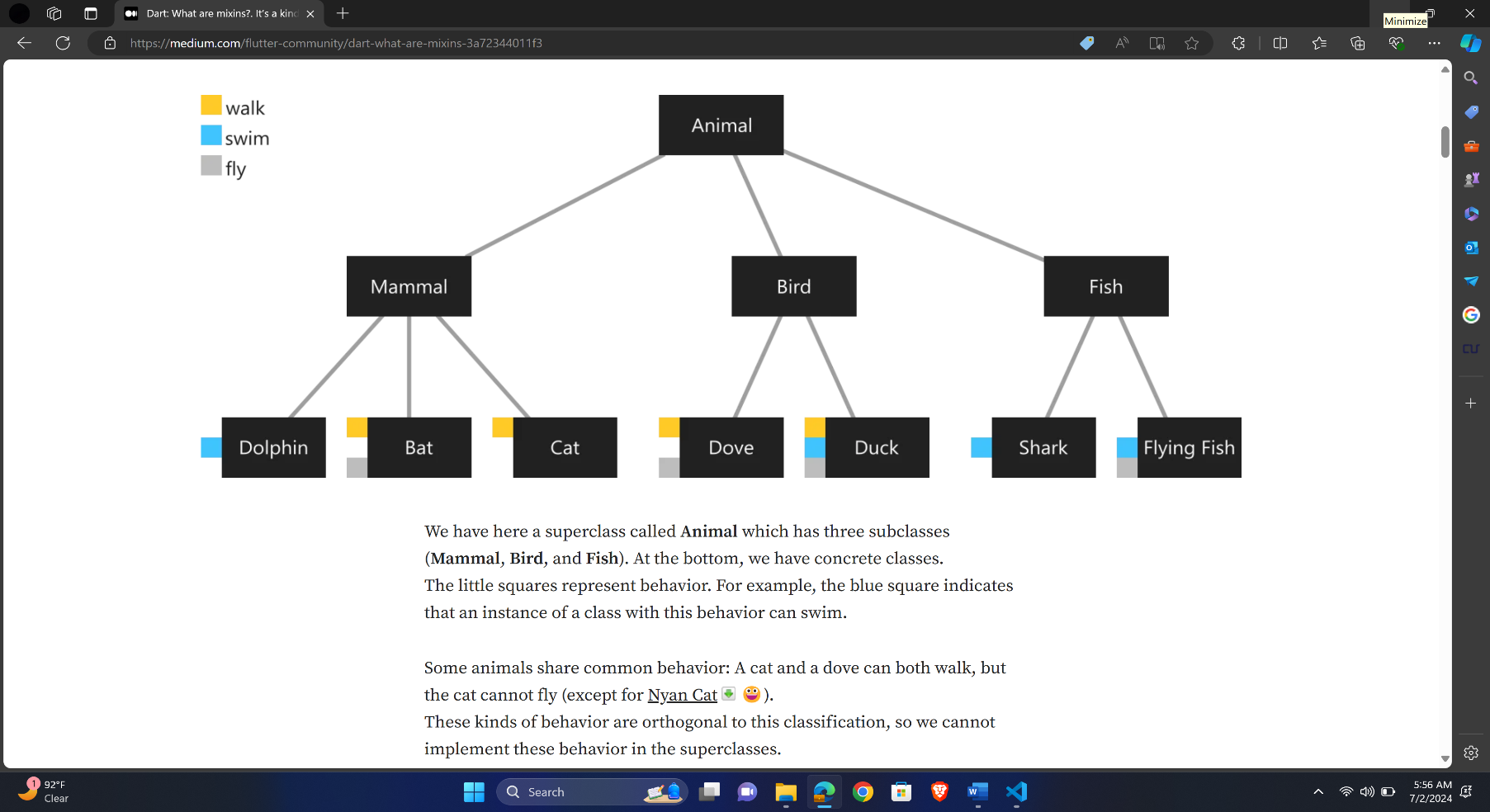
class Cat extends Mammal with Walker {}

class Dove extends Bird with Walker, Flyer {}

class Duck extends Bird with Walker, Swimmer, Flyer {}

class Shark extends Fish with Swimmer {}

class FlyingFish extends Fish with Swimmer, Flyer {}



**Polymorphism**

Polymorphism allows methods to do different things based on the object it is acting upon. It provides the ability to call the same method on different objects and each object responds in a unique way. It is also known as overriding, means the parent and child class have same methods(same name and type). @override is used as mentioned below.

* **Example:**

void main(){

Player p1;

p1 = new Footballer();

p1.display();

Player p2 = new Wrestler();

p2.display();

}

class Player{

void display(){

print('Player');

}

}

class Footballer extends Player{

@override

void display(){

super.display(); // it will display parent class display

print('Footballer');

}

}

class Wrestler extends Player{

@override

void display(){

print('Wrestler');

}

}

**Abstraction:**

* **Abstract Classes:** An abstract class is a class that cannot be instantiated. It is often used as a base class for other classes. All Abstract methods of Parent class must be override in child class, otherwise it will give error. However, normal methods of abstract class may or may not be override in child class.

void main(){

Footballer p1 = new Ronaldo();

p1.position();

Pepe p2 = new Pepe();

p2.position();

}

abstract class Footballer{

void position(); // abstract class

void stamina(){} // normal class

}

class Ronaldo extends Footballer{

// It will give error if position function is not implemented

void position(){

print('Striker');

}

}

class Pepe extends Footballer{

void position(){

print('Defender');

}

}

* **Interfaces:** In Dart, all classes implicitly define an interface. An interface is a contract that a class must follow. In Dart, multiple inheritance is not allowed. So, Interfaces are used for that case. Multiple inheritance means, a child class has one or more than one parent classes. Super keyword can never be used in interfaces. All the parent class methods must be implemented in child class, otherwise it will give error. **Implements** keyword is used instead of extends keyword.

void main(){

person p1 = new person();

p1.footballer();

p1.hobby();

}

class Footballer{

void footballer(){

print('I am Footballer');

}

}

class Hobby{

void hobby(){

print('I do gardening');

}

}

class person implements Footballer,Hobby{

@override

void footballer() {

print('I am Footballer in my city.');

}

@override

void hobby() {

print('I do gardening as my hobby.');

}

}

**Null Safety**

Dart's null safety feature helps to avoid null reference errors by making types non-nullable by default. This means that unless you explicitly specify that a variable can be null, it cannot be assigned a null value.

* **Non-nullable types:**

int nonNullableInt = 5; // Cannot be null

* **Nullable types:**

int? nullableInt = null; // Can be null

* **Handling null values safely:**

String? name;

print(name?.length); // Safe access, prints null if name is null

name = 'Dart';

print(name!.length); // Non-null assertion, throws an error if name is null

void main(){

// Declaring a nullable variable by using ?

String? name;

// Assigning John to name

name = "John";

// Assigning null to name

name = null;

// Checking if name is null using if statement

if(name == null){

print("Name is null");

}

// Using ?? operator to assign a default value

String name1 = name ?? "Stranger";

print(name1);

// Using ! operator to return null if name is null

String name2 = name!;

print(name2);

}