

Flutter & Dart Complete Guide

Prepared by Eng. Yousef Salah Kassem

DAFL 1
Dart OOP Part 1

Lecture 5



Chapter

05

Table Of Content

- Introduction
- Classes and Objects
- Methods in Classes
- Constructors



Introduction

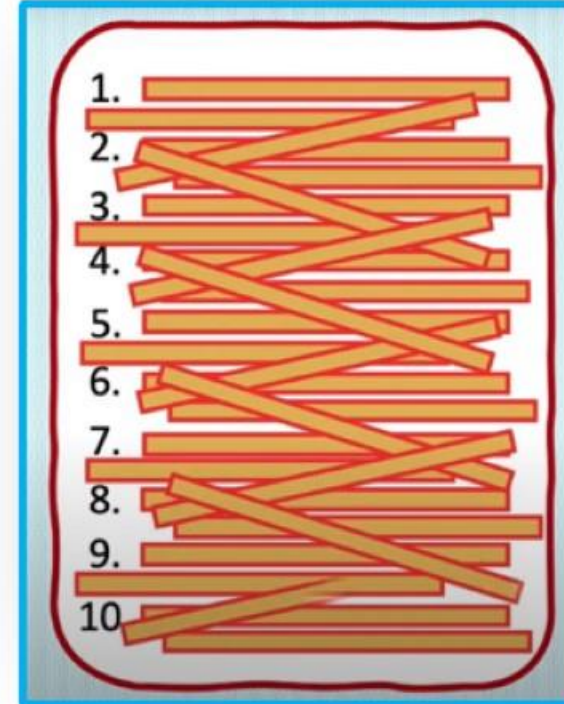
Introduction

- ◆ Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.



Chapter
05**Complex Projects**

In complex projects, a program can consist of thousands of lines of code. What will happen if we do not properly organize it?



Spaghetti Code



OOP

- ◆ Object-oriented Programming, or OOP for short, is a programming paradigm which provides a means of structuring programs so that properties and behaviors are bundled into individual objects.



Classes and Objects

OOP

For instance, an object could represent a person with a name property, age, address, etc., with behaviors like walking, talking, breathing, and running. Or an email with properties like recipient list, subject, body, etc., and behaviors like adding attachments and sending.



OOP

Put another way, object-oriented programming is an approach for modeling concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc.



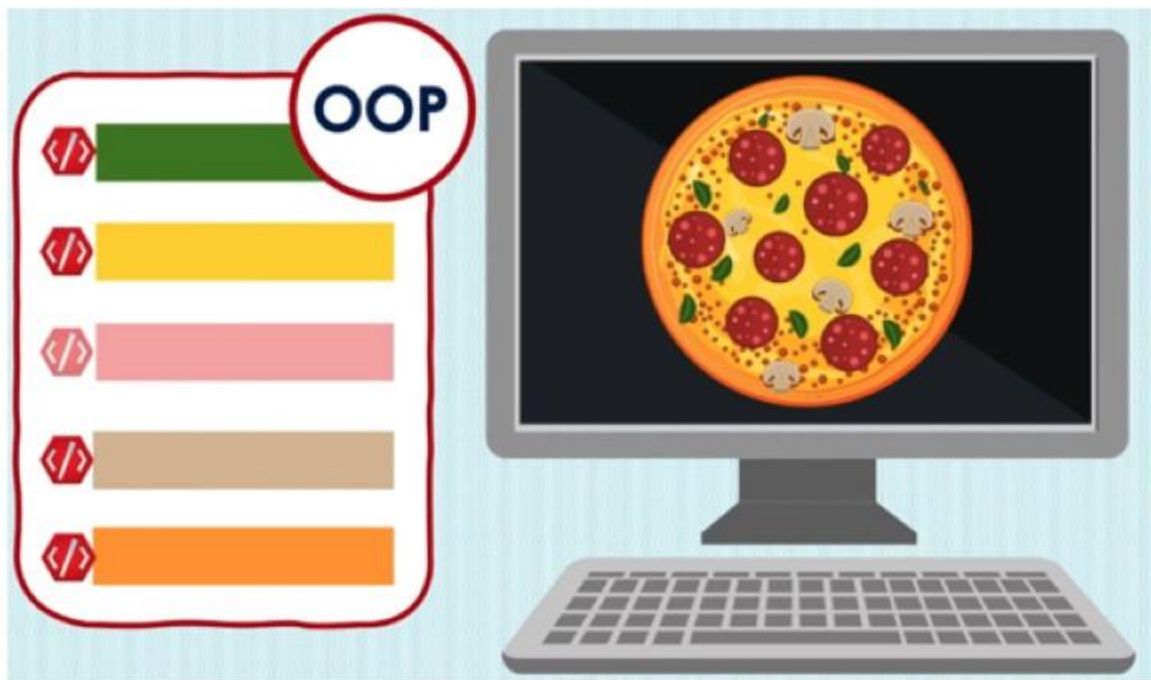
OOP

OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.



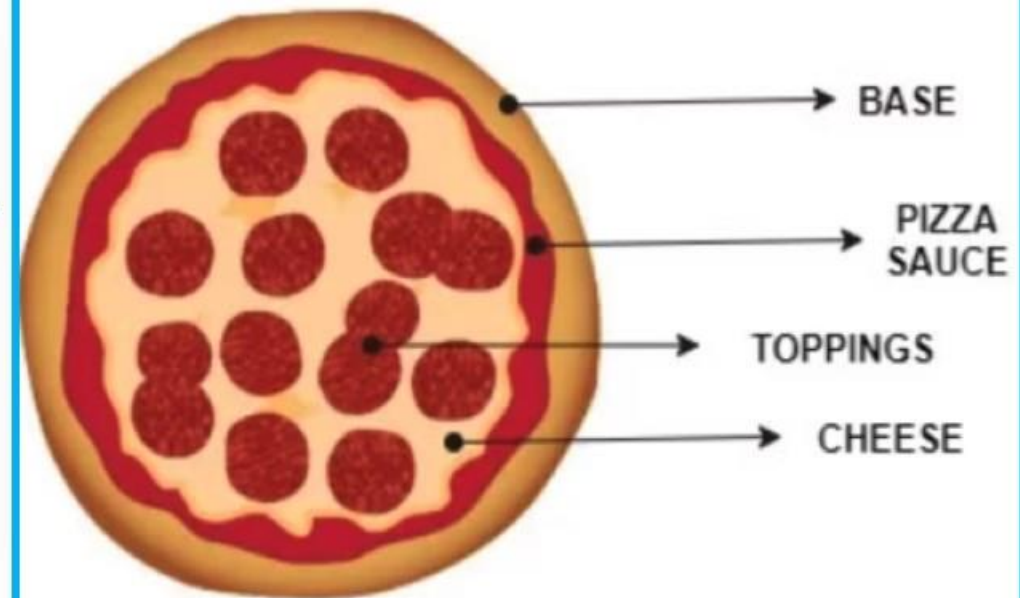
Why OOP ?

- ◆ It enhances code reusability without losing anything, simplifies maintenance, and aligns with real-world concepts.



Classes are used to create new user-defined data structures that contain arbitrary information about something.

This recipe, when created in OOPS, represents a blueprint and is called a class.



Classes

The class by itself is not an actual pizza, it only represents what elements and processes constitute the making of the pizza. This is why when you declare a class, no memory is allocated to it. This is kind of like building a house where you first need to create a blueprint or a schematic drawing of what your house will look like.

Objects

While the class is the blueprint, an instance is a copy of the class with actual values, literally an object belonging to a specific class.

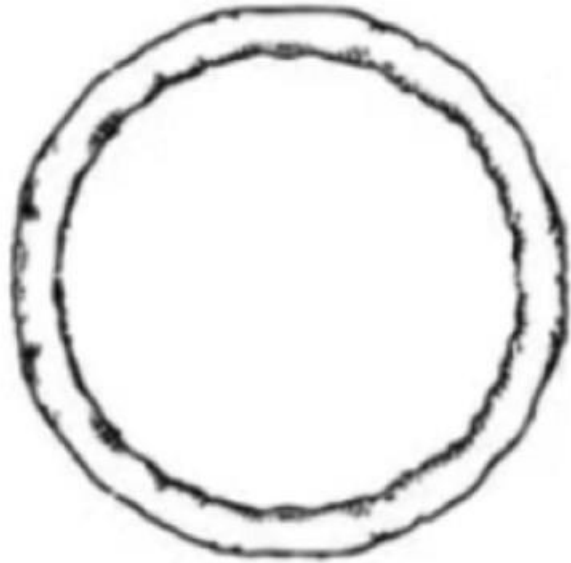
Put another way, a class is like a form or questionnaire.

It defines the needed information. After you fill out the form, your specific copy is an instance of the class; it contains actual information relevant to you.

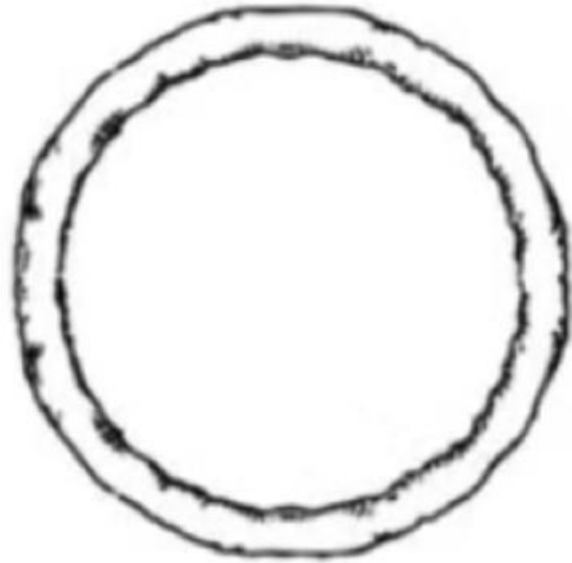
Objects

From a class, we create objects. These are real-world entities with some characteristics and behaviour. In our example, the objects would be pizzas created from a recipe that is present in the class.

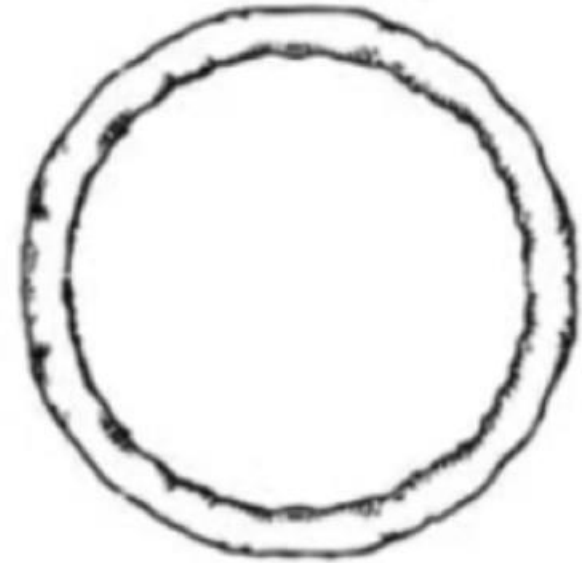
Objects of a class are also referred to as instances of that class and the creation of objects is called instantiation.



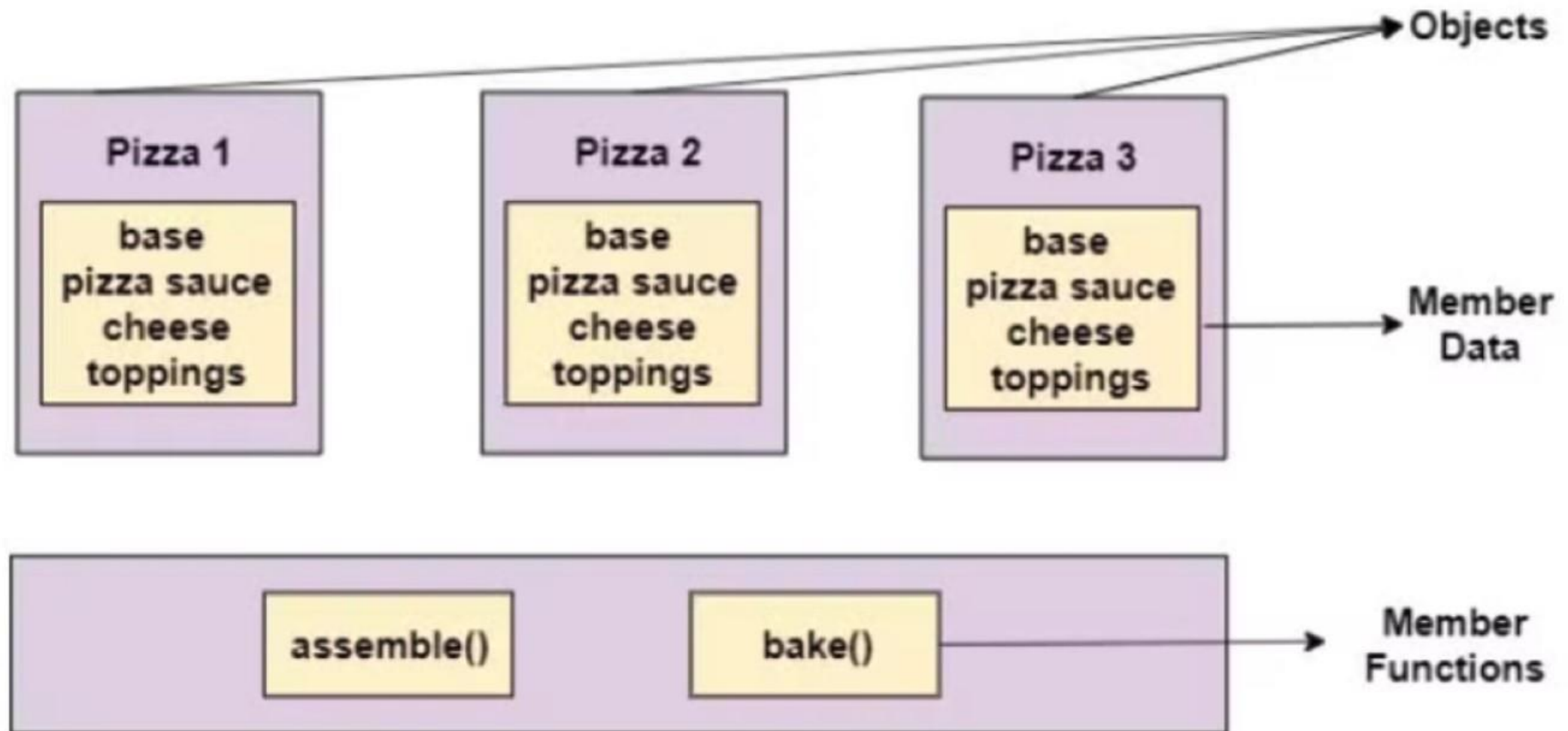
Pizza 1



Pizza 2



Pizza 3



OOP Features

- Reduction in the complexity
- Importance of data
- Creation of new data structure
- Data Hiding
- Characterization of the objects
- Communication among objects
- Extensibility
- Bottom-up programming approach

Creating a class

```
class ClassName {  
    <fields>  
    <getter/setter>  
    <constructor>  
    <functions>  
}
```

```
class Laptop {  
    final String processorName;  
    final String? dedicatedVgaName;  
    final String integratedVgaName;  
    final bool isRamMultiChannel;  
    final int ramSize;  
    final int romSize;  
  
    Laptop({  
        required this.integratedVgaName,  
        required this.processorName,  
        required this.isRamMultiChannel,  
        required this.ramSize,  
        required this.romSize,  
        this.dedicatedVgaName,  
    });  
  
    String? _price;  
  
    set setPrice(String price) => _price = price;  
  
    String? get price => _price;  
  
    void benchmarkTest() {  
        print('Benchmark test is running');  
    }  
}
```

Live Coding

OOP Concepts

01

Inheritanc
e

02

Polymorphism

03

Abstraction

04

Encapsulation

Methods in Classes

Dart provides us with the ability to create methods of our own. The methods are created to perform certain actions in class. Methods help us to remove the complexity of the program. It must be noted that methods may and may not return any value and also it may or may not take any parameter as input. Methods in a class can be either an object method or a class method.

There are two types of methods in Dart:

1. **Instance Method**
2. **Class Method**

Instance Method in Dart

Unless the method is declared as static it is classified as an instance method in a class. They are allowed to access instance variables. To call the method of this class you have to first create an object.

```
// Creating Class named Gfg
class Gfg {
    // Declaring instance variable
    int a;
    int b;

    // Creating instance method name
    // sum with two parameters
    void sum(int c, int d)
    {
        // Using this to set the values of the
        // input to instance variable
        this.a = c;
        this.b = d;

        // Printing the result
        print('Sum of numbers is ${a + b}');
    }
}

void main()
{
    // Creating instance of the class Gfg(Creating Object)
    Gfg geek = new Gfg();

    // Calling the method sum with the use of object
    geek.sum(21, 12);
}
```

All the methods declared with static keyword are termed as **class method**. They can't access non-static variables and can't invoke non-static methods of the class. It must be noted that unlike instance method class method can directly be called by using class name.

```
// Creating Class named Gfg
class Gfg {

    // Creating a class method name
    // sum with two parameters
    static void sum(int c, int d)
    {
        // Printing the result
        print('Sum of numbers is ${c + d}');
    }
}

void main()
{
    // Calling the method sum without the
    // use of object i.e with class name
    Gfg.sum(11, 32);
}
```

Constructors

- A constructor is a different type of function which is created with same name as its class name.
- The constructor is used to initialize an object when it is created. When we create the object of class, then constructor is automatically called.
- It is quite similar to class function but it has no explicit return type.
- The generative constructor is the most general form of the constructor, which is used to create a new instance of a class.

Chapter
05

Constructor No Arguments

- A Constructor which has no parameter is called default constructor or no-arg constructor. It is automatically created (with no argument) by
- Dart compiler if we don't declare in the class. The Dart compiler ignores the default constructor if we create a constructor with argument or no argument.

- We can also pass the parameters to a constructor that type of constructor is called parameterized constructor.
- It is used to initialize instance variables. Sometimes, we need a constructor which accepts single or multiple parameters.
- The parameterized constructors are mainly used to initialize instance variable with own values.

- The named constructors are used to declare the multiple constructors in single class.

The syntax is given below.

- Syntax: `className.constructor_name(param_list)`

- The `this` keyword is used to refer the current class object. It indicates the current instance of the class, methods, or constructor.
- It can be also used to call the current class methods or constructors. It eliminates the uncertainty between class attributes and the parameter names are the same.
- If we declare the class attributes same as the parameter name, that situation will create ambiguity in the program, then the `this` keyword can remove the ambiguity by prefixing the class attributes.
- It can be passed as an argument in the class method or constructors.

Chapter
05**Constructor**

```
void main() {  
    // Creating an object  
    Student std = new Student("Alisha",26);  
  
}  
class Student{  
  
    // Declaring a constructor Student(String str, int age){  
    print("The name is: ${str}");  
    print("The age is: ${age}");  
}
```

Example

Library Management System

In this problem, you'll create a system for managing a collection of books in a library. Each book will have a title, author, ISBN, and availability status. The system should allow you to add new books, remove books by ISBN, check the availability of books, and borrow or return a book.

What is Next ?

Example

Requirements:

1. Create a Book Class:

- Properties:
 - **title**: The title of the book (String)
 - **author**: The author of the book (String)
 - **isbn**: The ISBN of the book (String)
 - **isAvailable**: A boolean representing the availability of the book (initially true)
- Constructor:
 - Initializes all properties.
- Methods:
 - **borrowBook()**: Marks the book as borrowed (sets **isAvailable** to **false**).
 - **returnBook()**: Marks the book as returned (sets **isAvailable** to **true**).
 - **isBookAvailable()**: Returns the current availability status.

Example

Requirements:

Create a Library Class:

- Properties:
 - `books`: A list of `Book` objects.
- Constructor:
 - Initializes an empty list of books.
- Methods:
 - `addBook(Book book)`: Adds a new book to the library.
 - `removeBook(String isbn)`: Removes a book from the library by ISBN.
 - `findBook(String isbn)`: Finds and returns a book by ISBN, or null if it doesn't exist.
 - `listAvailableBooks()`: Lists all books currently available for borrowing.
 - `borrowBook(String isbn)`: Checks if the book is available, borrows it if it is, and returns a message if it is unavailable.
 - `returnBook(String isbn)`: Returns the book to the library if it exists and is currently borrowed.



Thank You