Introduction to Object-Oriented Paradigm

By: Laams Innovation Lab

Lecture Contents

What You Will Learn Throughout This Month

Revision of the Dart Language

New Class Schedule

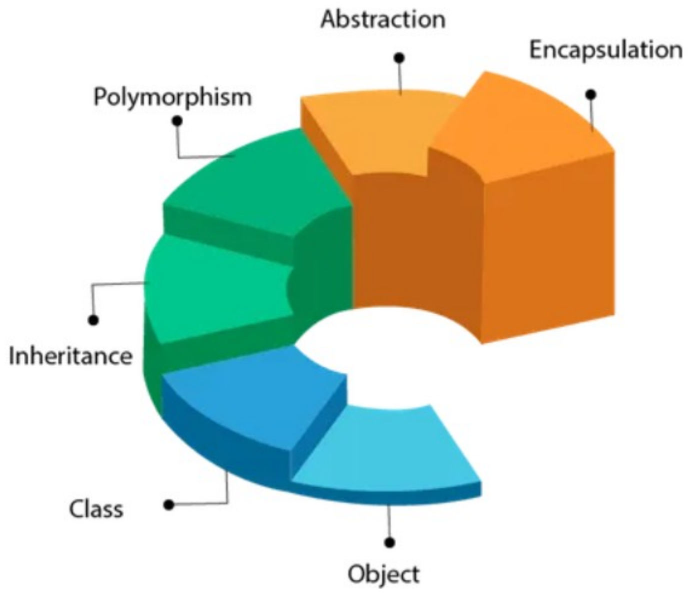
Introduction to OOP

What You Will Learn throughout This Month

OOP & Gand of Fours Design Patterns

The primary focus of this semester is to introduce you to **OOP & design principles**, and then help you practice with **Gang of Fours Design Patterns**.

You will not only learn to **develop well-designed software**, **refactor** your code based on design principles and **test** your software, but also learn to **read** other peoples code.



Technologies You Will be Familiarized With

You will be introduced to **Browser** technology, and start using **HTML**, **CSS** and a bit of **JavaScript** & create a **Dart Compiled to JavaScript** client-side Application.

You will learn to **version control** your code, commit your repositories online, and work in teams using **Git & GitHub**.

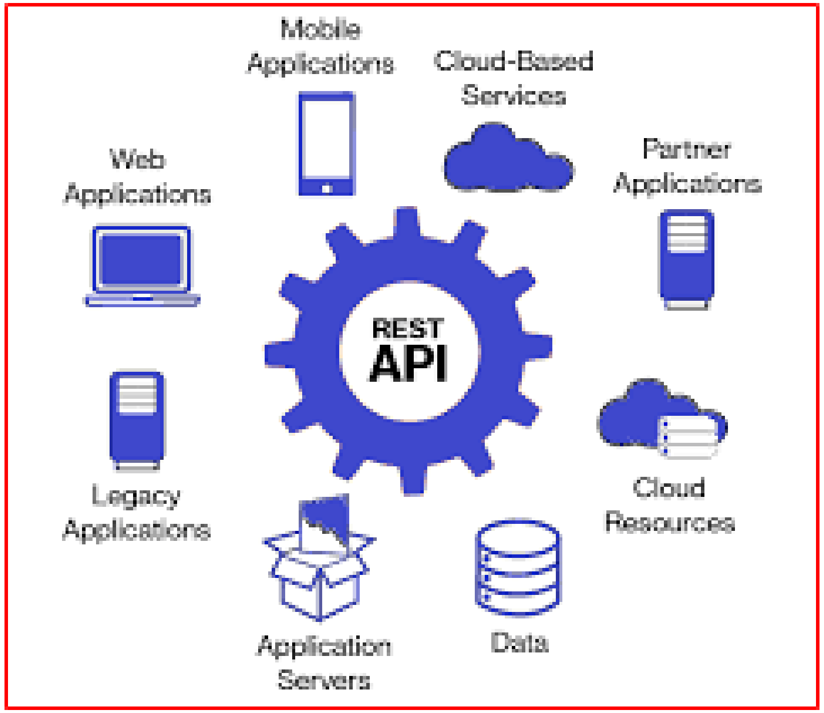
You will learn to use **MongoDB** as a **database** for caching your applications users' data.

You will learn to use **Docker** to **containerize** your back-end REST API services.

The Applications You Will Create

You will learn to create a **REST API** using **Shelf Router** for the **Laams Teams** app, whose GUI we will develop in flutter later in the course. The app features includes:

1. User Authentication
2. Data Retrieval Using JSON
3. A Console Client App
4. Other functionalities..



The Third-Party Packages You Will Self-Learn

You will **self-learn** the following and multiple other **third-party packages** available on pub.dev:

- Characters Package
- Args Package
- Derry Package
- Ferry Package
- Puppetear Package
- Dio Package
- Source Gen Package
- Hive Package
- Universal HTML Package
- Markdown & Yaml Packages
- UUID & Faker Packages
- Equatable Package
- Path Package
- Path Provider Package
- Crypto & Encrypt Packages
- Logger & Logging Packages
- PDF Package
- Email Validator
- Mockito Package
- English Words & translator
- Intl & ShamsiDate Package
- BuiltValue & Build Collection
- Lint Package
- Googleapis Package
- Ansicolor Package
- Cryptography Package
- RxDart Package
- Mockito Package
- RxDart Package
- Redux Package

Revision of the Dart Language

The Why of Dart, Real Life Usage

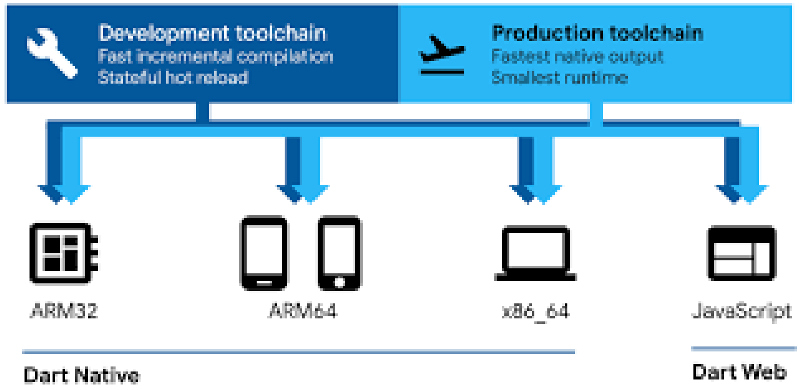
Dart is one of the most **practical** and **productive** languages ever created. You can use **Flutter SDK** (written in Dart) to develop very **efficient GUI applications** (60 frames per second) for **Android**, **IOS**, **Web**, **Mac**, **Linux**, **Windows**, **Fuchsia** & other IoT devices & platforms.

It is pretty **easy to learn**, and a lot **resources** (documentations, book, articles and videos) are available online to help you learn.

The Why of Dart, Technical

Dart is **small & modern Java** (my experience), which takes the **best** of other languages. It takes **null safety** from Swift, **package management** from Rust & Go, and **type inference** from dynamic languages (JavaScript & Python).

Dart also solves the problems faced by other languages. It has features such as **Isolates**, **native compilation** to any platform, even to JavaScript, and is also equipped with **JIT** & **AOT** compilers.



The Why of Dart, Maintenance

Dart is developed and maintained by **Google Engineers**, and adapted by numerous **big companies**, such as Microsoft, Grab, Toyota...

It means that more **features** & **packages** will be added to the language. Also, companies are starting to use it for creating efficient **back-end services** and for the **cloud**, As it is supported in **GCP** and **AWS**.

The Why of Dart, Personal

I **love** this language, **after Go**, and it is used in **Laams LLC** for its **efficiency** and **productivity**. Learn it to get a job at Laams, or **many other companies** around the world. It should **always** be the language of your **choice**.

Dart is the future.

لمس

Revision of Class Principles

No Self-victimization

The **world** and **nature** are **cruel**,
so don’t be a crying baby!

Don’t be *leaded* by your **emotions**,
*lead your **emotions**!*

Be Disciplined

"We are what we **repeatedly do**. **Excellence**, then, is not an **act**, but a **habit**."

Will Durant

Stay Curious

“Always go **beyond** memorizing formulas, passing tests, to always **go deep into** the **underlying principles** of a subject, to track any problem down to the root cause buried in the dirt and the dark.”

Elon Musk

Practice, Practice, Practice!

Talent is Overrated. The difference between a 10x Programmer and a normal one is their **practice** **level**.

Learn to Work in Teams

“**Teamwork** begins by building **trust**. And the only way to do that is to overcome our need for **invulnerability**”

Patrick Lencioni

New Class Schedule

Part 01: Warm up questions (10-15 minutes)
Part 02: Design Principle or Pattern (40-60 minutes)
Part 02: Working on Project (30 Minutes)
Part 03: Package Introduction (30 Minutes)

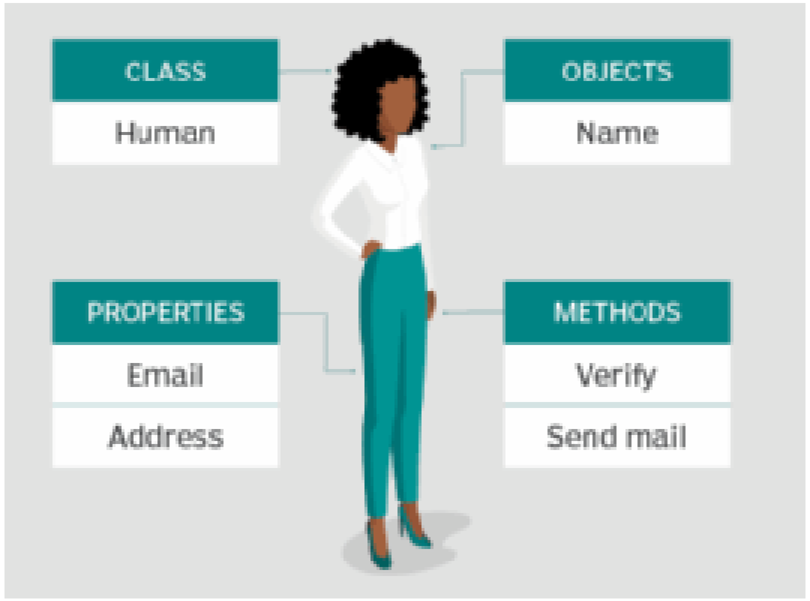
OOP Part I: Object-Oriented Modeling

Object-Oriented Modeling

Object Oriented Modeling is about **thinking** of everything in **terms of objects**. it is about representing **key concepts** in software through **objects**.

When thinking of objects, know that they have **properties**, **behaviors**, and are **self-aware**. Also there are three types of objects: **entity** objects, **control** objects and **boundary** objects. They are what holds **space** in **memory**.

Object Oriented Modeling helps your code be **flexible**, **reusable** and **maintainable**.



Look around you, identify and list all the **objects** you see, choose one of the listed objects and define all its **properties** and **behaviors**!

Object Oriented Modeling Principles

To better **model** objects, you should follow OOM design principles, which include: **Abstraction**, **Encapsulation**, **Decomposition** and **Generalization**.

Abstraction Principle

Abstraction principle is about **simplifying** a **concept** to its **essentials** within **some context**.

Abstraction helps you *better understand* the concept by breaking it down to its *simplified description*, and ignore *unimportant details*.

When abstracting you must apply **the rule of least astonishment**, which is about **not** defining anything behind the **scope** of the **concept**, and capturing its **essential** **properties** and **behaviors**.

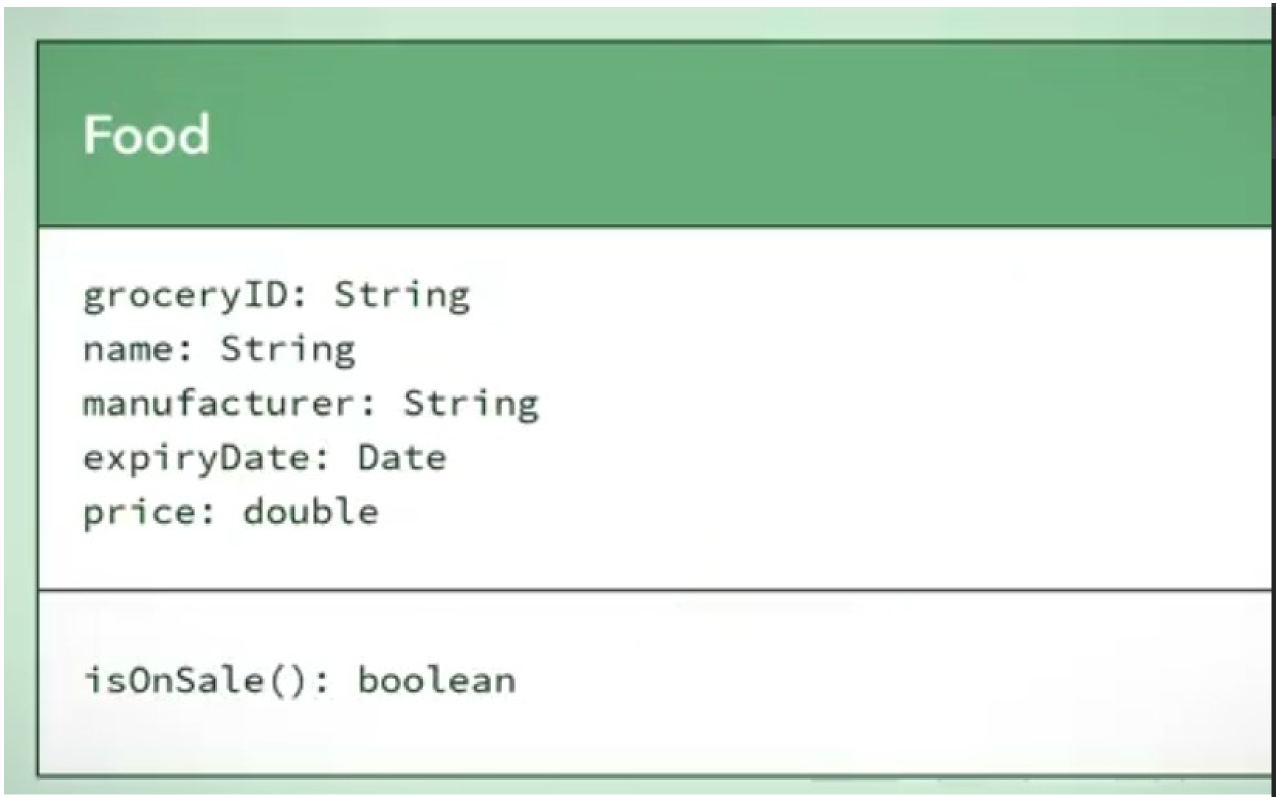
Abstraction UML Class Diagram

Class Name

Attributes | Properties | Instance Variables
Syntax: Property Identifier: Property Type
Example: firsName: String

Operations | Behaviors | Instance Methods
Syntax: Behavior Name (parameters): Return Type
Example: getFullName(userType: String): String

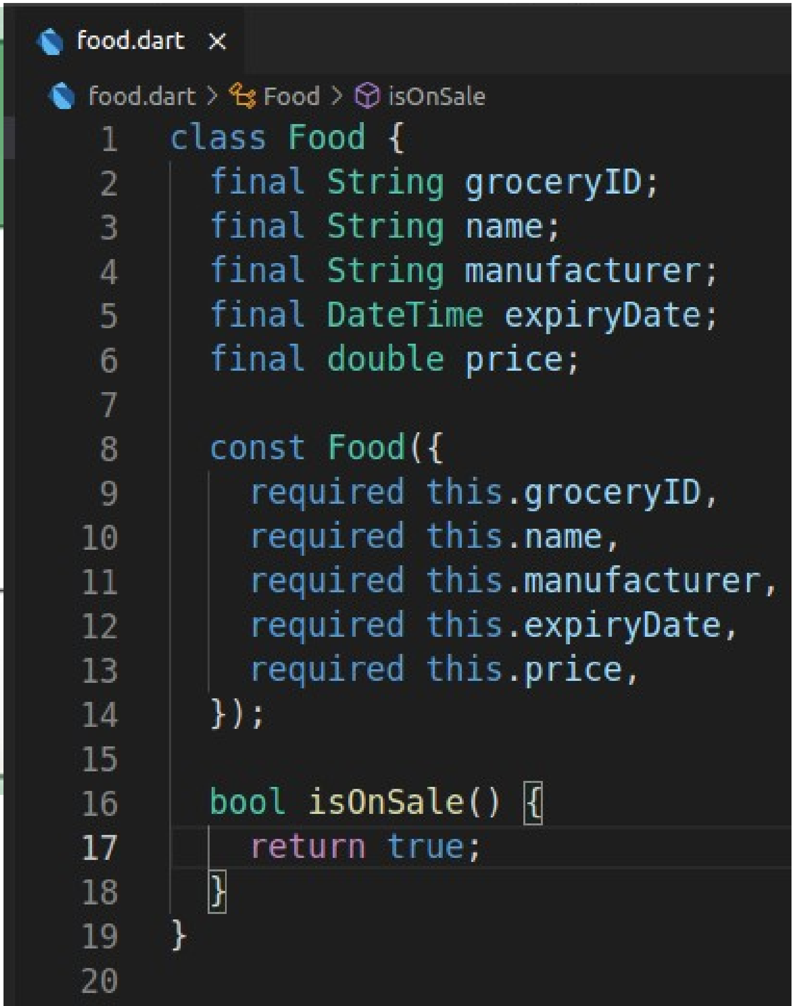
Abstraction UML Class Diagram Example



```
Food

groceryID: String
name: String
manufacturer: String
expiryDate: Date
price: double

isOnSale(): boolean
```



```dart
class Food {
  final String groceryID;
  final String name;
  final String manufacturer;
  final DateTime expiryDate;
  final double price;

  const Food({
    required this.groceryID,
    required this.name,
    required this.manufacturer,
    required this.expiryDate,
    required this.price,
  });

  bool isOnSale() {
    return true;
  }
}
```

Define the properties and behaviors of **Variable** in Dart, create a **class diagram** for it, and change diagram to **code**!

Dart Commands and Tools

The IO Package

File: create, read, delete, update.

Directory: create, read, delete, update.

Platform: isIOS, isAnroid, isWindows, isLinux, isMac, isFuschia

HttpServer: creating a server.