CS4039
SOFTWARE FOR
MOBILE DEVICES

ABDUL WAHAB USMANI

- Lead Software Engineer at Blink Co. Technologies (Jan 2023 - Present)
- Senior Software Engineer at VentureDive (Sep 2021- Dec 2022)
- Software Engineer at Aban Tech(July 2019 - Aug 2021)
- BS(CS) from FAST-NUCES

MOBILE DEVICES OVERVIEW

Android

- Open source (AOSP) but customized by OEMs (Samsung, Xiaomi, Oppo, etc.)
- Largest market share worldwide.
- Apps packaged as APK / AAB.
- Built on Linux kernel.

iOS

- Closed ecosystem, only on Apple devices.
- Strict app review guidelines, uniform hardware/software integration.
- Apps packaged as IPA.
- Built on XNU kernel (Darwin/Unix-based).

Others (Harmony OS, KaiOS (feature phones))

App Stores & Distributions

Google Play Store (Android):

- Developer account: one-time $25 fee.
- Review process: automated + light human check, usually quick (hours).
- Supports staged rollouts, open/closed testing, in-app subscriptions.

Apple App Store (iOS):

- Developer account: $99/year.
- Stricter review process: can take days, high rejection rate.
- Enforces guidelines (privacy, design, security).

Alternative stores (Android only):

- Amazon Appstore, Samsung Galaxy Store, Huawei AppGallery.

App Lifecycles

Android:

- States:
- onCreate() → app initialized.
- onStart() / onResume() → visible & interactive.
- onPause() / onStop() → partially hidden or backgrounded.
- onDestroy() → killed/cleaned.

Special cases:

- OS may kill background apps for memory.
- Users expect apps to restore state (back stack, form inputs).
- Services/foreground services run even when UI is closed.

App Lifecycles

iOS:

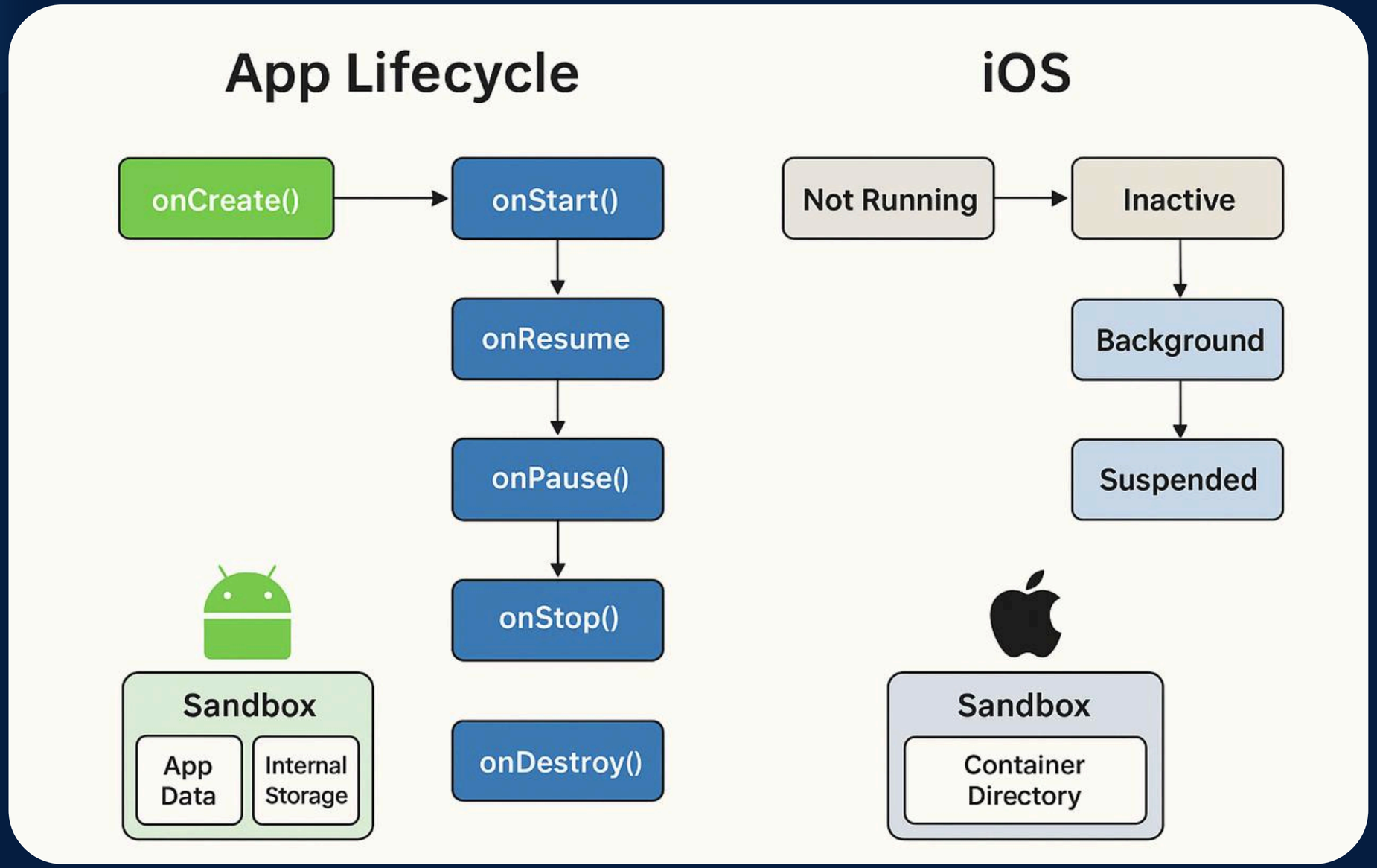
States:

- Not Running → app closed.
- Inactive → transitioning, e.g., phone call interruption.
- Active → foreground, interactive.
- Background → executing short tasks (location, audio).
- Suspended → in memory, not executing code.

Restrictions:

- Apps in background are heavily throttled.
- Must request explicit permissions for background tasks.
-

App Lifecycles



DART BASICS

WHAT IS DART?

- Dart is an object oriented language
- Dart is an open source and scalable programming language with many built-in libraries
- Useful for developing web, server and mobile applications
- optionally typed, and single threaded programming language.
- Similar to other OOP languages it supports classes, objects and methods.

ORIGINS

- Developed by Google
- Dart was unveiled at the GOTO conference in Aarhus, Denmark, October 10, 2011.
- Founded by Lars Bak and Kasper Lund, legacy developers of V8 JavaScript Engine used in Chrome
- The first version Dart 1.0 was released on November 14, 2013
- Latest version (DART 3.9) was released on August 14, 2025.

CONCEPTS

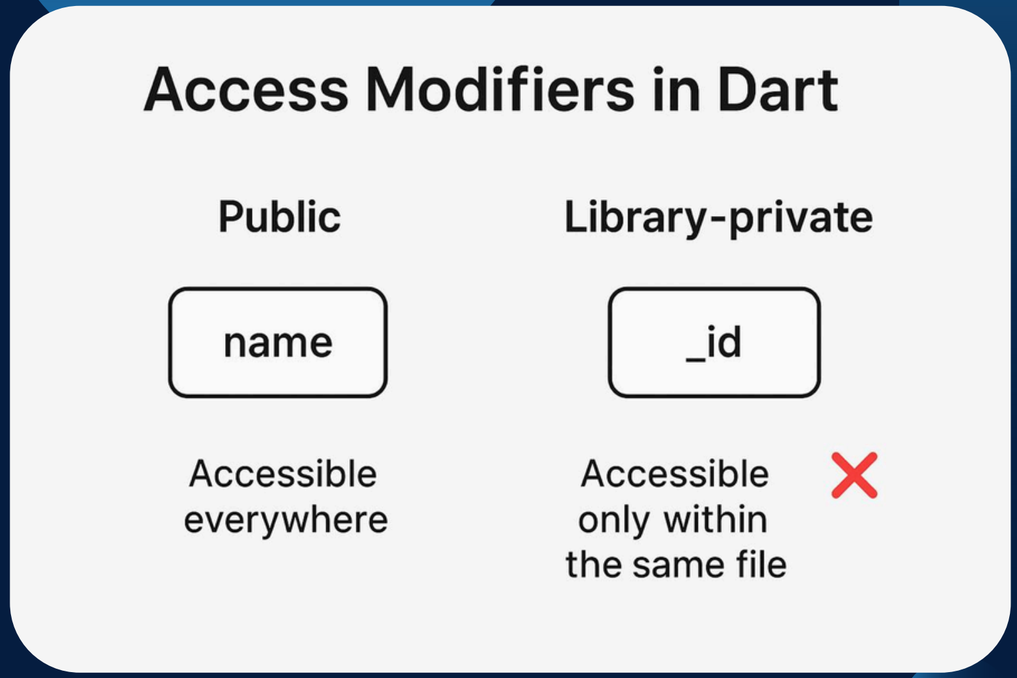
01 Everything is an object

02 Even numbers, Booleans, functions and null are objects

03 Every object is an instance of a class that inherit from Object

04 Dart doesn’t support keywords public, private and protected

05 Dart doesn’t support keywords public, private and protected



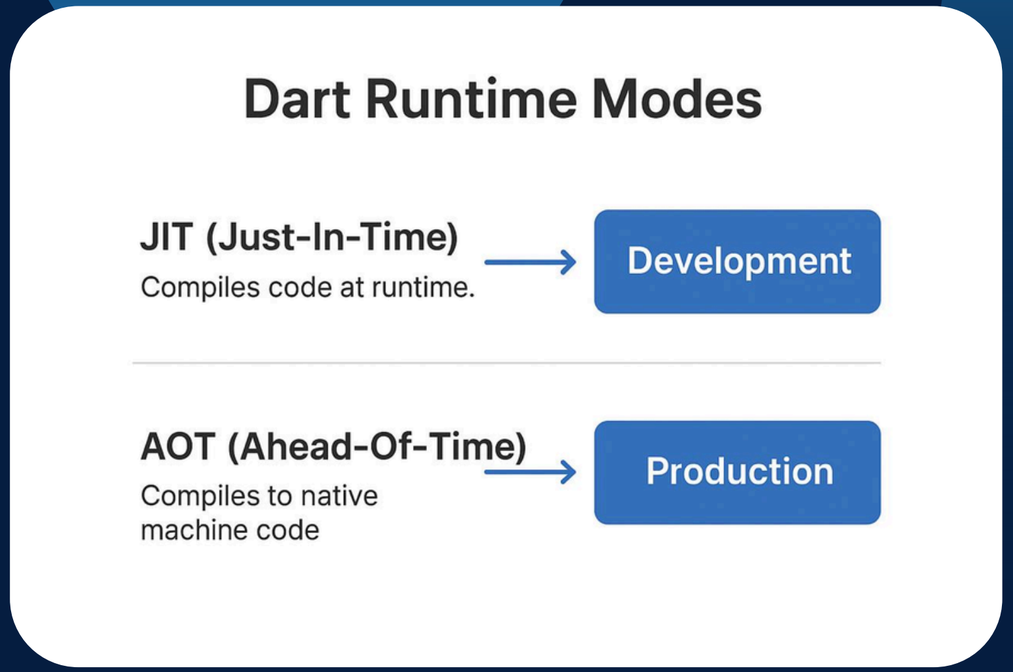
CONCEPTS

06 Dart tools can report two kinds of problems: warnings and errors

07 Runtime Modes in Dart

JIT (Just-In-Time): Compiles code at runtime. Ideal for development with hot reload and quick iteration.

AOT (Ahead-Of-Time): Compiles to native machine code before deployment. Used in production for faster startup and better performance.



HELLO WORLD

- Program execution starts from the top-level main() function.
- Example:

```
void main() {
  print('Hello, World!');
}
```

- No return value → void is default

DART TOOLS

DARTPAD

DartPad is an online tool to practice Dart syntax and features without installation, supporting core libraries but not VM libraries like dart:io.

IDE ‘S AND EDITOR

Android studio, Visual Studio Code, IntelliJ IDEA

COMMAND-LINE TOOLS

A command-line interface (CLI) for creating, formatting, analyzing, testing, documenting, compiling, and running Dart code, as well as working with the pub package manager.

DATA TYPES

- Numbers (Integer, Double)
- Strings
- Booleans
- Lists
- Maps
- Dynamic (optionally typed language)

VARIABLES AND CONSTANTS

- Use var, final, const, or explicit types
- Dart infers types when you don’t specify them
- Final and const for immutable values, with subtle differences
- Example

```
var name = 'Smith';
String name = 'Smith';
int num = 10;
dynamic x = "tom";
```

Final and Const

```
final val1 = 12;
const pi = 3.14;
```

OPERATORS

- Arithmetic Operators → +, -, *, /, %
- Relational Operators → ==, !=, >, <, >=, <=
- Logical Operators → &&, ||, !
- Assignment Operators → =, +=, -=, *=, /=
- Null-aware Operators → ??, ?., ??=

```
int a = 10, b = 5;
print(a + b);   // 15
print(a > b);   // true
print(null ?? "default"); // default
```

LOOPS

TYPES OF LOOPS

- **for** → Fixed number of repetitions
- **while** → Runs while condition is true
- **do-while** → Runs at least once
- **for-in** → Iterates over collections

```
for (int i = 1; i <= 5; i++) print(i);      // for
while (i <= 5) { i++; }                     // while
do { i++; } while (i <= 5);                 // do-while
for (var fruit in ["Apple","Mango"]) print(fruit); // for-in
```

CONDITIONAL STATEMENTS

IF / ELSE IF / ELSE

Executes code based on conditions

```
if (x > 10) {
  print("Greater than 10");
} else if (x == 10) {
  print("Equal to 10");
} else {
  print("Less than 10");
}
```

SWITCH

Compares a variable against multiple values

```
switch (day) {
  case "Mon": print("Start of week"); break;
  case "Fri": print("Weekend soon"); break;
  default:    print("Other day");    break;
}
```

STRING INTERPOLATION

- Embeds variables or expressions inside strings.
- Syntax: $variable or ${expression}.
- Example

```
void main() {
   String str1 = "hello";
   String str2 = "world";
   String res = str1 + str2;

   print("The concatenated string : ${res}");
   // Output: The concatenated string : helloworld
}
```

LISTS

- Ordered collection of objects (like arrays).
- Two types are

FIXED-LENGTH

size cannot change

Fixed-Length

```
var lst = new List(3);
lst[0] = 12;
```

GROWABLE

size can increase dynamically

Growable

```
var num_list = [1, 2, 3];
var lst = new List();
lst.add(12);
```

MAPS

- A Map is a collection of key-value pairs.
- Keys are unique, values can be duplicated.

Ways to Create Maps:

USING MAP LITERAL

```
var user = { 'id': 1, 'name': 'Alice' };
```

USING MAP CONSTRUCTOR

```
var details = new Map();
details['Username'] = 'admin';
```

CLASSES

- A class is a blueprint for creating objects.
- It defines properties (fields) and behaviors (methods).
- Supports object-oriented programming: encapsulation, inheritance, polymorphism.

Declare a class

```
class class_name {
   <fields>
   <getters/setters>
   <constructors>
   <functions>
}
```

Instantiating

```
var obj = new Car("Engine 1")
```

Accessing

```
//accessing an attribute
obj.field_name

//accessing a function
obj.function_name()
```

Named Constructor

```
class Car {

  Car() {
print("Non-parameterized constructor invoked");
}

  Car.namedConst(String engine) {
print("The engine is : ${engine}");
}

}
```

OBJECT

- Objects are instances of classes.
- Represent data + behavior.
- Created directly from a class.

```
class Car {
String brand;
Car(this.brand);
}
void main() {
var car1 = Car("Toyota");
print(car1.brand);
}
```

The cascade operator (..)

The cascade operator can be used as a shorthand in cases where there is a sequence of invocations.

```
void main() {
    new Student()
    ..testMethod1()
    ..testMethod2();
}
```

SET AND QUEUE

Set represents a collection of objects in which each object can occur only once.

```
Set numberSet = new  Set();
Set numberSet = new Set.from([12,13,14]);
```

A HashSet is an unordered hash-table based Set implementation.

```
Set numberSet = new  HashSet();
```

A Queue is a collection that can be manipulated at both ends.

```
  Queue queue = new Queue();
    queue.add(10);
    queue.add(20);
    queue.add(30);
```

```
numQ.addFirst(400); //at the beginning of a Queue
numQ.addLast(400);  //at the end of Queue
```

EXCEPTION HANDLING

- Exceptions handle runtime errors gracefully.
- Dart provides try, catch, on, and finally blocks.
- Example

```
void main() {
  try {
    var result = 12 ~/ 0;   // Integer division by zero
  } on IntegerDivisionByZeroException {
    print("Cannot divide by zero");
  } catch (e) {
    print("Error: $e");
  } finally {
    print("Program ends");
  }
}
```