Current Trends in Software Engineering

Lesson 3: Introduction to Dart

Content

- What is Dart?
- Dart History
- Dart Features
- Flutter & Dart
- Start Coding with Dart...

What is Dart?

- Dart is a **client-optimized programming language** for apps on multiple platforms.
- Dart is an **open-source** general-purpose programming language.
- It is developed by **Google** and is used to build mobile, desktop, backend and web applications.
- Dart is an **object-oriented**, **Strongly Typed**, **garbage-collected** language

Dart History

- Dart was unveiled at the **GOTO** conference in Aarhus, Denmark, October 10–12, 2011
- The project was founded by **Lars Bak** and **Kasper Lund**
- **Dart 1.0** was released on November 14th, 2013.
- Recently release **Dart 2.6** is accompanied with a new extension dart2native.

Dart Usage

- Mobile app development using Flutter
- Web development (using Dart2JS)
- Server side (using various libraries and frameworks)

Dart features

- **Optimized for UI**
  Develop with a programming language specialized around the needs of user interface creation

- A programming language that is easy to learn, with a familiar syntax

Dart

```
class Segment {
  int links = 4;
  toString() => "I have $links links";
}
```

Kotlin

```
class Segment {
  var links: Int = 4
  override fun toString()= "I have $links links"
}
```

Swift

```
class Segment: CustomStringConvertible  {
  var links: Int = 4
  public var description: String { return
  "I have \(links) links"}
}
```

TypeScript

```
class Segment {
  links: number = 4
  public toString = () : string => { return
  `I have ${this.links} links` };
}
```

Dart features Cont…

- **Productive development**
  - Make changes iteratively: use hot reload to see the result instantly in your running app

- Write code using a flexible type system with rich static analysis and powerful, configurable tooling

- Do profiling, logging, and debugging with your code editor of choice

```
var temperature = 25;

// Static code analysis catches errors early.
temperature = 'Freezing';

// Customizable code style checks
class weather{}
```

A String can't be assigned to an 'int'

[dart] Name types using UpperCamelCase

Dart features Cont…

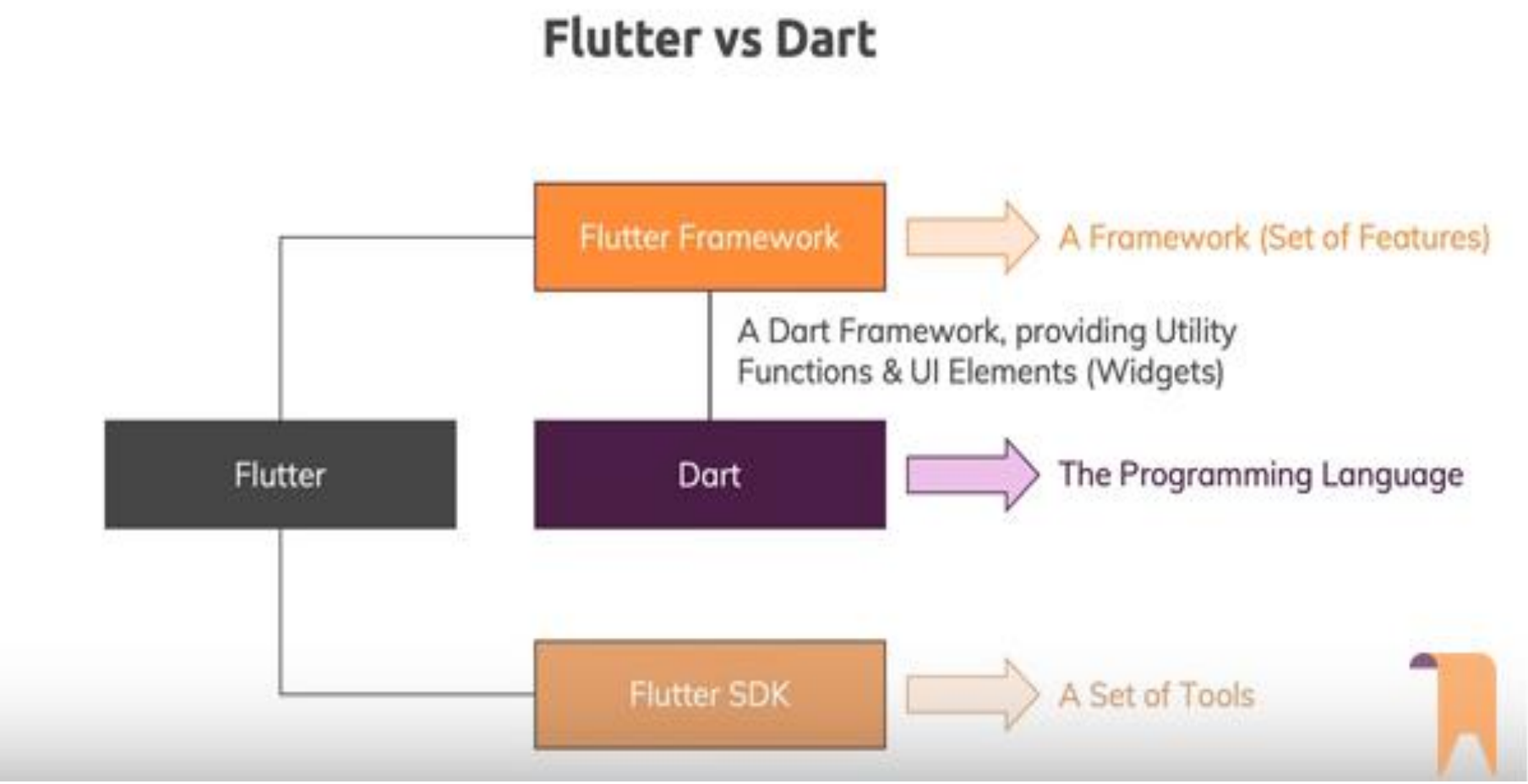
- **Fast on all platforms**
  Compile to ARM & x64 machine code for mobile, desktop, and backend.
  Or compile to JavaScript for the web

- **Dart can also be JIT (Just In Time) compiled for exceptionally fast development cycles**

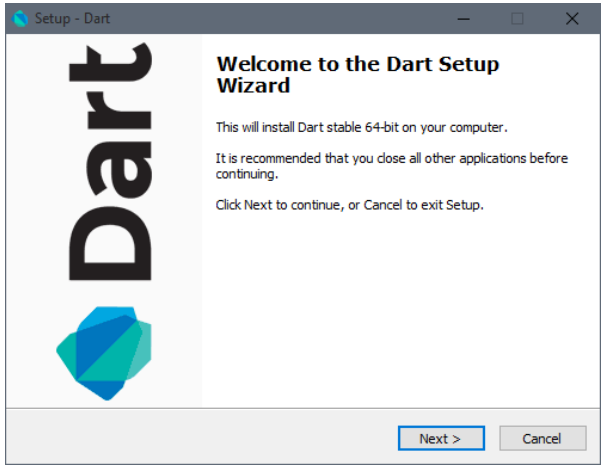
- Dart makes it easier to create **smooth animations and transitions** that run at **60fps**.

- Dart can do object allocation and garbage collection **without locks**.

Flutter and Dart



Dart Installation



Dart SDK

Use the community-supported **Dart SDK** installer for Windows

DartPad

It is an online editor at https://dartpad.dartlang.org

Dart SDK


Android Studio


Visual Studio Code

IntelliJ IDEA
(and other JetBrains IDEs)


Emacs


Atom


Vim


Eclipse

Hello World

DartPad New Pad Reset Format floral-hat-1325

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

RUN

Console

```
Hello, World!
```

Documentation

Data Types

Dart is an optionally typed language

- Numbers
  Int and **double**
- Strings
  String is used to represent string literals.
- Booleans
  Uses the **bool** keyword to represent a Boolean value.
- Lists
  Ordered group of objects and it is synonymous to the concept of an array in other programming languages. **List**
- Maps
  set of values as key-value pairs. **Map**
- Dynamic
  If the type of a variable is not explicitly specified, the variable's type is **dynamic**.

Dart input output handling

```
import 'dart:io';

void main(){
    //read number from user
    print('Enter a number');
    var line = stdin.readLineSync();
    int a = int.parse(line);

    if(a<0){
        print('$a is negative number.');
    } else if(a==0) {
        print('$a is zero. Neither negative nor positive');
    } else {
        print('$a is positive number.');
    }
}
```

Using Labels to Control the Flow

- A label is simply an identifier followed by a colon (:) that is applied to a statement or a block of code.
- A label can be used with break and continue to control the flow more precisely.
- Line breaks are not allowed between the ‘continue’ or ‘break’ statement and its label name. Also, there should not be any other statement in between a label name and an associated loop.

Control flow statements

DartPad New Pad Reset Format

RUN

```dart
void main() {
   outerloop: // This is the label name

   for (var i = 0; i < 5; i++) {
      print("Innerloop: $i");
      innerloop:

      for (var j = 0; j < 5; j++) {
         if (j > 3 ) break ;

         // Quit the innermost loop
         if (i == 2) break innerloop;

         // Do the same thing
         if (i == 4) break outerloop;

         // Quit the outer loop
         print("Innerloop: $j");
      }
   }
}
```

floral-hat-1325

Console

```
Innerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Innerloop: 2
Innerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Innerloop: 4
```

Documentation

Collections - List

- Dart represents **arrays in the form of List objects**. A List is simply an ordered group of objects.
- The dart:core library provides the List class

List

Fixed Length List

var list_name = new List(initial_size)
lst_name[index] = value;

Growable List

var list_name = [val1,val2,val3]
creates a list containing the specified values

OR

var list_name = new List()
creates a list of size zero

Collections - List

Fixed Length List

```
void main() {
   var lst = new List(3);
   lst[0] = 12;
   lst[1] = 13;
   lst[2] = 11;
   print(lst);
}
```

RUN

Console

```
[12, 13, 11]
```

Collections - List

Growable List

DartPad New Pad Reset Format floral-hat-1325

```
void main() {
   var num_list = [1,2,3];
   print(num_list);
}
```

RUN

Console

```
[1, 2, 3]
```

```
void main() {
   var lst = new List();
   lst.add(12);
   lst.add(13);
   print(lst);
}
```

RUN

Console

```
[12, 13]
```

Collections - Map

- The Map object is a simple **key/value pair**. Keys and values in a map may be of any type. A Map is a **dynamic collection**.

- Maps can be declared in two ways –

  Using Map Literals

  var identifier = { key1:value1, key2:value2 [,.....,key_n:value_n] }

  Using a Map constructor

  var identifier = new Map()

  map_name[key] = value

Collections - Map

DartPad New Pad Reset Format floral-hat-1325

```
void main() {
   var details = {'Usrname':'tom','Password':'pass@123'};
   details['Uid'] = 'U1oo1';
   print(details);
}
```

RUN

Console

```
{Usrname: tom, Password: pass@123, Uid: U1oo1}
```

```
void main() {
   var details = new Map();
   details['Usrname'] = 'admin';
   details['Password'] = 'admin@123';
   print(details);
}
```

RUN

Console

```
{Usrname: admin, Password: admin@123}
```

```
void main() {
  Map<String, String> people = new Map<String, String>();
  people.putIfAbsent('firstname', () => 'Sharad');
  print(people);
}
```

RUN

Console

```
{firstname: Sharad}
```

Collections - Queue

```dart
import 'dart:collection';

void main() {
  Queue items = new Queue();
  items.add(1);
  items.add(3);
  items.add(2);
  items.removeFirst();
  items.removeLast();
  print(items); // { 3 }
}
```

RUN

Console

{3}

Imports

```
// Importing core libraries
import 'dart:math';

// Importing libraries from external packages
import 'package:test/test.dart';

// Importing files
import 'path/to/my_other_file.dart';
```

Functions

```
void main() {
   print(factorial(6));
}
factorial(number) {
   if (number <= 0) {
      // termination case
      return 1;
   } else {
      return (number * factorial(number - 1));
      // function invokes itself
   }
}
```

RUN

Console

```
720
```

Lambda Functions

```
void main() {
   printMsg();
   print(test());
}
printMsg()=>
print("hello");

int test()=>123;
// returning function
```

RUN

Console

```
hello
123
```

Classes

Dart supports all the features for Object-oriented programing paradigm like Classes, Inheritance, Interfaces, Polymorphism, etc.

Class example

```
class Mobile {
  String color;
  String brandName;

  String calling() {
    return "Mobile can do calling";
  }

  String musicPlay() {
    return "Mobile can play Music";
  }
}
```

Create Object

```
var myMobile = new Mobile();
```

Classes

```
void main() {
     User user1 = new User();
    User  user2 = User();

     user1.id = 121;
     user1.email = "sharadghimire5551@gmail.com";
     print("${user1.id} and ${user1.email}");

     user1.register();
     user1.login(user1.email,"567");
}
class User {
    int id;
    String lastname;
    String firstname;
    String email;
    String password;
    void login(email, password){
       print('Welcome! Your email is $email');
    }
    void register() => print('Thanks for registering');
}
```

RUN

Console

```
121 and sharadghimire5551@gmail.com
Thanks for registering
Welcome! Your email is sharadghimire5551@gmail.com
```

Documentation

Constructors

- A constructor is an **instance method** that is invoked when an object is created from the class.
- This is a good place to initialize **instance variables**.

```
// constructor function
Person( String firstName, String lastName, [ int age = 18 ] ) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
}
```

Named constructors

Dart provides **multiple constructors** on a class. Apart from default constructors, other constructors must have a name. While creating an object from a class, we need to use the **name named constructor**.

```
class Employee {
  int empID;
  String empName;
  String empDept;

  Employee.iD(this.empID); // Named Constructor Creation

  Employee.name(this.empName);

  Employee.department(this.empDept);
}

main() {
  var myEmployee01 = new Employee.iD(15);
  var myEmployee02 = new Employee.department("Testing");
  var myEmployee03 = new Employee.name("Ashu");

  print(myEmployee01.empID);
  print(myEmployee02.empDept);
  print(myEmployee03.empName);
}
```

RUN

Console

```
15
Testing
Ashu
```

Documentation

Dart Access Specifiers

- Dart doesn't provide Access specifiers keywords like private, public and protected.
- can use _ (underscore) at the start of the name to make a data member of a class becomes private.
- So, a data member is either public (if not preceded by _) or private (if preceded by _)

Dart Access Specifiers

```
class A {
  String first;
  String _second;
}

void main() {
  A a = new A();
  a.first = 'New first';
  a._second = 'New second';
  print('${a.first}: ${a._second}');
}
```

a.dart

other.dart

```
import 'a.dart';

void main() {
  A a = new A();
  a.first = 'New first';
  a._second = 'New second'; // The setter _second is not defined for the class 'A'
  print('${a.first}: ${a._second}'); // The getter _second is not defined for the class
}
```

Dart Access Specifiers

a.dart

```
class A {
  String first;
  String _second;

  String get second {
    return _second;
  }

  void set second(String second) {
    this._second = second;
  }
}
```

other.dart

```
import 'a.dart';

void main() {
  A a = new A();
  a.first = 'New first';
  a.second = 'New second';
  print('${a.first}: ${a.second}');
}
```

Class Inheritance- interface

- Dart has no interface keyword.

```
void main() {
   ConsolePrinter cp= new ConsolePrinter();
   cp.print_data();
}
class Printer {
   void print_data() {
      print("__________Printing Data__________");
   }
}
class ConsolePrinter implements Printer {
   void print_data() {
      print("__________Printing to Console__________");
   }
}
```

RUN

Console

```
__________Printing to Console__________
```

Interface
As normal class is act as Interface technically it can be instantiated.
Abstract method cannot created here.
When interface implements in other class every method and instance variable needs to override.
We can implement multiple interfaces.

Class Inheritance – Abstract class

```
abstract class Person {
  void walk(); //Abstract Method
  void talk(); //Abstract Method
}

class Jay extends Person {
  @override
  void walk() {
    print("Jay can walk");
  }

  @override
  void talk() {
    print("Jay can talk");
  }
}

void main() {
  Jay jay = new Jay();
  jay.talk();
  jay.walk();
}
```

RUN

Console

```
Jay can talk
Jay can walk
```

Abstract Class
Abstract class cannot be instantiated.
Abstract methods can be created here.
When abstract class is extends by other class. Only abstract method needs to override
We can only extend one abstract class.

Mixins

- A **Mixin** is a class that contains methods for use by other classes without having to be the parent class of those other classes.

- **mixins** are normal classes from which we can borrow methods(or variables) from without extending the class.

```
mixin Coder {
  void code() {
    print("Coding intensifies");
  }
}
```

Thank You