Fakultas Informatika
School of Computing
Telkom University

CRI3I3
Pemrograman Perangkat Bergerak

Intro Dart

Dart

- Dart is an open-source general-purpose programming language. It is originally developed by Google and later approved as a standard by ECMA
- Dart is an object-oriented language with C-style syntax which can optionally trans compile into JavaScript
- Dart can be used for the server as well as the browser

Difference between Dart and Javascript

Feature	Dart	JavaScript
Type system	Optional, dynamic	Weak, dynamic
Classes	Yes, single inheritance	Prototypical
Interfaces	Yes, multiple interfaces	No
Concurrency	Yes, with isolates	Yes, with HTML5 web workers

Variables & Data Types

- Rules of naming
  - Case sensitive
  - Start with a letter or underscore
  - Must not a Dart's reserved word
  - Must not contains operational characters
- Variable declaration

```
var namevariable = value;
<type> namevariable = value;
```

- Data types:
  - Standard (bool, int, double, String)
  - List
  - Map
  - Enum
  - Dynamic

Example: standard data types

- `bool flag = true;` (boolean)
  - unlike Javascript, only true value is considered a true condition
- `int i = 2;` (integer)
- `double j = 2.3;` (double)
- `String name1 = "double";` (string double quote)
- `String name2 = 'single';` (string single quote)

Code commenting

- Same with C++ / Java / PHP
  - `// one line`
  - `/* ..`

    `multi-line`

    `.. */`

Example: List

- Initialization

```
//fixed length
var L1 = new List(5);
//growable list
var L2 = new List(); //or,
var L3 = []; //or,
var L4 = ["IF", "IT", "SE"];
```

- Call

```
print(L4[0]); //output: IF
print(L3[1]); //output: Exception
```

Example: List

- Assignment

```
L2[0] = "S2 IF";
L3.add("S3 IF");
```

Example: Map

- Initialization

```
var M2 = new Map(); //or,
var M3 = {}; //or,
var M4 = {"IF":1301, "IT":1303, "SE":1302};
```

- Call

```
print(M4['IF']); //output: 1301
print(M3[1]); //output: null
```

Example: Map

- Assignment

```
M4["S2 IF"] = 2301;
M3.addAll({"S3 IF":3301});
```

Example: Enum

- Initialization

```
enum Status {
    none,
    running,
    stopped,
    paused
}
```

- Call

```
print(Status.values[1]); //output: running
```

Operator

- Arithmetic: + - * / ~/ % ++ --
- Assignment : = ??= += -= *= /=
- Comparison : == != > < >= <=
- Logical: && || !
- Type test: is !is
- Conditional:
  - condition ? expr1 : expr2
  - expr1 ?? expr2

Conditional (if)

- Example:

```
var i = 1;
if (i>0) {
  print("$i = positive number");
}
```

Conditional (if-else)

- Example:

```
var i = -8;
if (i>0) {
  print("positive");
} else if (i==0) {
  print("zero");
} else {
  print("negative");
}
```

Conditional (switch-case)

- Example:

```
var bil=5;
   switch (bil) {
   case 0 :
      print("zero");
      break;
   case 2 :
      print("two");
      break;
   default :
      print("other");
 }
```

Looping (while)

- Example:

```
var i=0;
while (i<4) {
  print("loop $i");
  i++;
}
```

Looping (do-while)

- Example:

```
var i=0;
do {
  print("loop $i");
  i++;
 } while (i<8);
```

Looping (for)

- Example:

```
for (var i=1;i<7;i++) {
  print("heading $i");
}
```

Looping (for in)

- Example:

```
var obj = [12,13,14];
for (var prop in obj) {
  print(prop);
}
```

Function

- Dart function is not similar to Javascript function:
  - Just the function name
  - Begins with keyword void if function doesn't return value
  - Can be declared with Lambda

Function

- Example:

```
//function declaration
  void sayhello(str) {
    print("hello $str");
  }
  add(a,b) {
    return a+b;
  }
  sayHi(n)=>print("hi $n!");
```

Function

- Example:

```
//function execution
  sayhello("bro!");
  var x = add(5,6);
  print("$x + 3 = " + add(x,3).toString());
  sayHi('Dart');
```

Class

- Structure:

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

- Execution:

```
var object_name = new class_name([ arguments ])
```

Constructor

- Similar like Java:

```
class class_name {
  class_name([arguments]) {
    //..
  }
  //named constructor
  class_name.constructor_name([arguments]) {
    //..
  }

}
```

Getter and Setter

- Structure:

```
class class_name {
  //getter
  return_type get function_name {
    //..
  }
  //setter
  set function_name {
    //..
  }

}
```

Sync Process

- In computing, we say something is synchronous when it waits for an event to happen before continuing.
- A disadvantage in this approach is that if a part of the code takes too long to execute, the subsequent blocks, though unrelated, will be blocked from executing.
- Consider a webserver that must respond to multiple requests for a resource

Sync Process

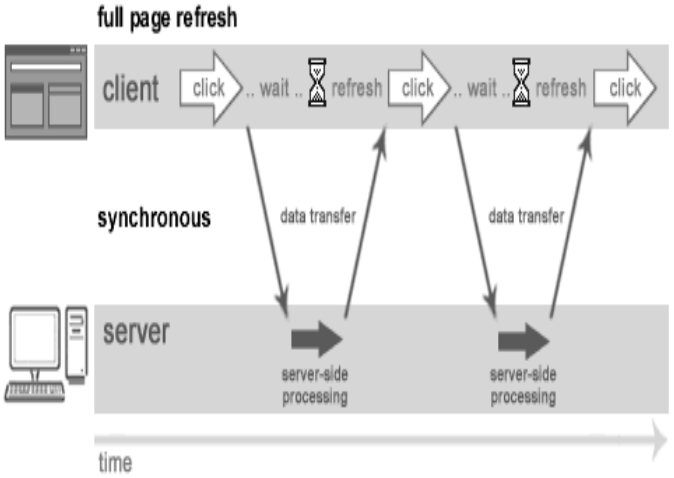
- A synchronous execution model will block every other user's request till it finishes processing the current request. In such a case, like that of a web server, every request must be independent of the others.
- This means, the webserver should not wait for the current request to finish executing before it responds to request from other users.
- Simply put, it should accept requests from new users before necessarily completing the requests of previous users. This is termed as asynchronous.

Async

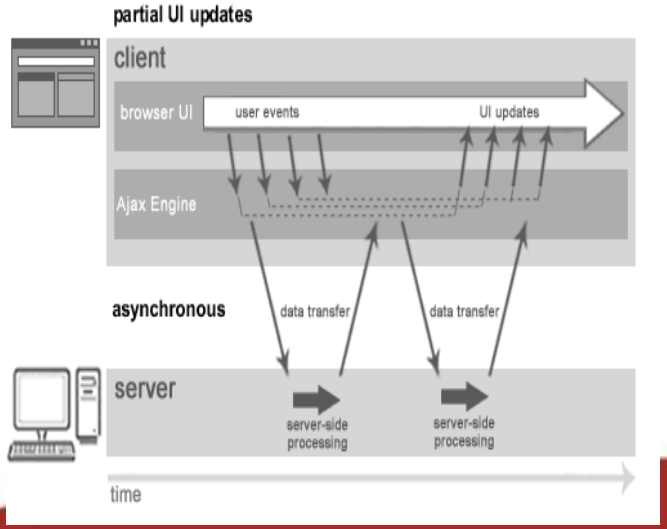
- Asynchronous programming basically means no waiting or non-blocking programming model
- An asynchronous operation executes in a thread, separate from the main application thread
- When an application calls a method to perform an operation asynchronously, the application can continue executing while the asynchronous method performs its task

Understanding Synchronous vs Asynchronous

- Synchronous (Classic Web-Application Model)



- Asynchronous (AJAX/API Web-Application Model)



Future

- a Future as "a means for getting a value sometime in the future." Simply put, Future objects are a mechanism to represent values returned by an expression whose execution will complete at a later point in time.
- Several of Dart’s built-in classes return a Future when an asynchronous method is called.
- Dart is a single-threaded programming language. If any code blocks the thread of execution (for example, by waiting for a time-consuming operation or blocking on I/O), the program effectively freezes.

Future

- Asynchronous operations let your program run without getting blocked. Dart uses Future objects to represent asynchronous operations

```
Future<void> fetchData() async {
  try {
    List<Order> orders = await Cord.getAll();
    //..
  } catch(e) {
    //..
  }
}
```

Any question?

References

- https://dart.dev/
- https://www.tutorialspoint.com/dart_programming/index.htm

Fakultas Informatika
School of Computing
Telkom University

THANK YOU