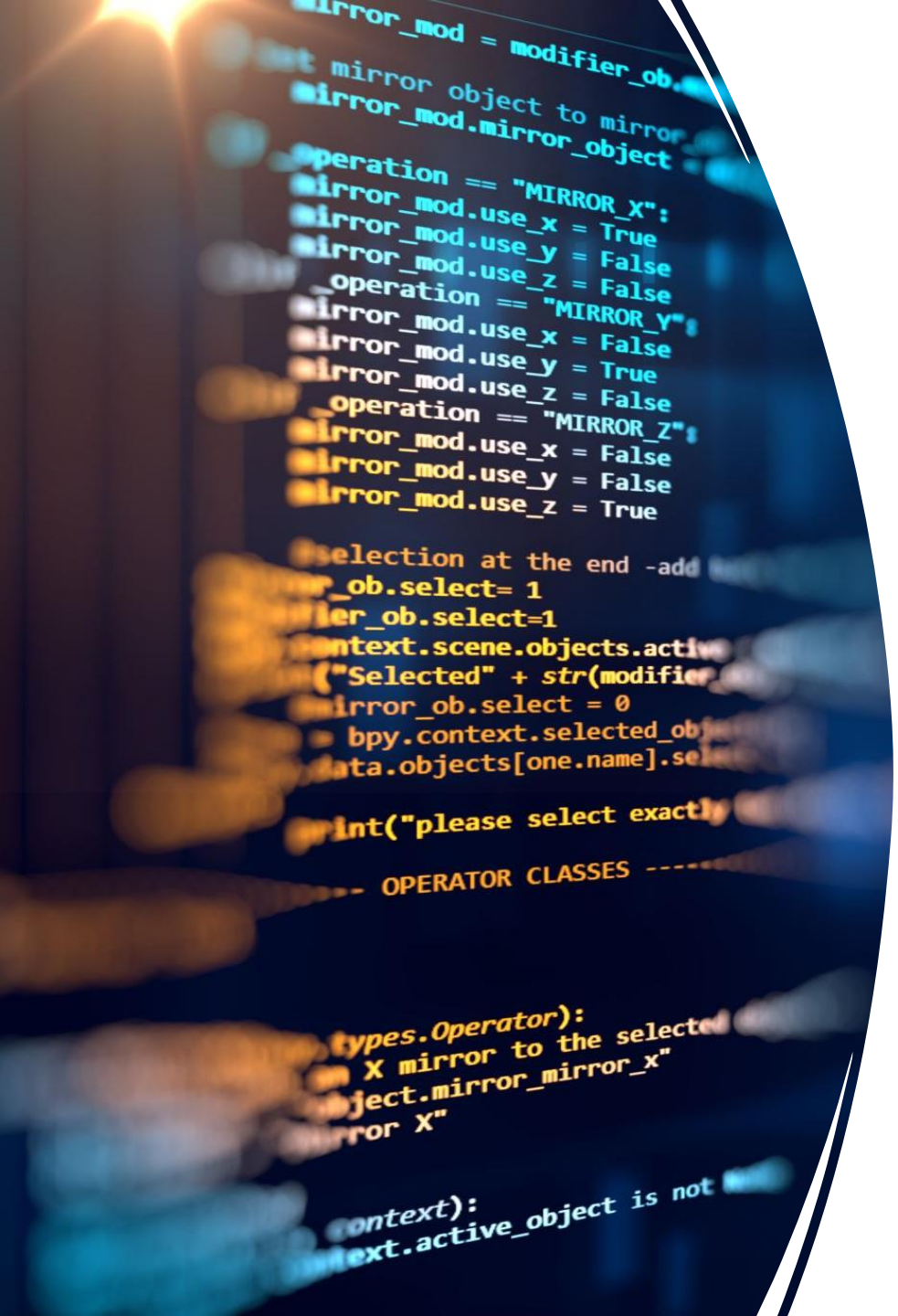
Asynchronous Programming and Null Safety

Written by Thapanapong Rukkanchanunt

Outline

- How and when to use the async and await keywords
- Execution order of async and await
- Error catching
- Recap: nullable and non-nullable
- ? and ! notations
- Null-aware Operators
- late keyword



What is asynchronous code?

- Asynchronous operations let your program complete work while waiting for another operation to finish
  - Fetching data over a network
  - Writing to database
  - Reading data from a file
- Asynchronous operations return result as a Future or Stream

Example code

- fetchUserOrder is an asynchronous function
- The code calls fetchUserOrder but it does not wait
- We got uncompleted future instead

```
String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

void main() {
  print(createOrderMessage());
}
```

Keywords

- **Synchronous operation**:
  - A synchronous operation blocks other operations from executing until it completes.
- **Synchronous function**:
  - A synchronous function only performs synchronous operations.
- **Asynchronous operation**:
  - Once initiated, an asynchronous operation allows other operations to execute before it completes.
- **Asynchronous function**:
  - An asynchronous function performs at least one asynchronous operation and can also perform *synchronous* operations.

What is Future?

- Future is a class.
- future is an instance of Future.
- A future represents the result of an asynchronous operation
  - **Uncompleted future:** When you call an asynchronous function, it returns an uncompleted future. That future is waiting for the function's asynchronous operation to finish or to throw an error.
  - **Completed future:** If the asynchronous operation succeeds, the future completes with a value. Otherwise, it completes with an error.

Future Example

- Two print statements
- Although fetchUserOrder gets called first, its print function is executed later

```
Future<void> fetchUserOrder() {
  return Future.delayed(const Duration(seconds: 2), () => print('Large Latte'));
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

async and await

- async keyword is used to define asynchronous function
  - Future<void> main() async { … }
- await keyword is used to call async function and wait for result
  - print(await createOrderMessage());

Example of async/await

```
String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order’;
}
Future<String> fetchUserOrder() =>
Future.delayed( const Duration(
seconds: 2), () => 'Large Latte', );
void main() {
  print('Fetching user order...’);
  print(createOrderMessage());
}
```

```
Future<String> createOrderMessage() async {
  var order = await fetchUserOrder();
  return 'Your order is: $order’;
}
Future<String> fetchUserOrder() =>
Future.delayed( const Duration(seconds: 2), ()
=> 'Large Latte', );
Future<void> main() async {
  print('Fetching user order...’);
  print(await createOrderMessage());
}
```

Handling Errors

- Use regular try-catch procedure

```
try {
  print('Awaiting user order...’);
  var order = await fetchUserOrder();
} catch (err) {
  print('Caught error: $err’);
}
```

Recap: Nullable and Non-Nullable

- By default, all variables are non-nullable.
- Use ? after the type name to make variable nullable

```
void main() {
  int a;
  a = 145;
  print('a is $a.'); // a is 145.
  int? b;
  b = null;
  print('b is $b.'); // b is null.
}
```

Exercise: Nullable List Declaration

- Where should we put ‘?’ ?

```
void main() {
  List<String> aListOfStrings = ['one', 'two', 'three'];
  List<String> aNullableListOfStrings;
  List<String> aListOfNullableStrings = ['one', null, 'three'];

  print('aListOfStrings is $aListOfStrings.');
  print('aNullableListOfStrings is $aNullableListOfStrings.');
  print('aListOfNullableStrings is $aListOfNullableStrings.');
}
```

Null Assertion Operator ‘!’

- Typically, you can’t assign int? to int because it could be null.
- However, if you **are sure** that it is **not null**, you can add ‘!’ at the end.

```
int? couldReturnNullButDoesnt() => -3;
void main() {
  int? couldBeNullButIsnt = 1;
  List<int?> listThatCouldHoldNulls = [2, null, 4];

  int a = couldBeNullButIsnt;
  int b = listThatCouldHoldNulls.first!; // first item in the list
  int c = couldReturnNullButDoesnt()!.abs(); // absolute value
  print('a is $a.');
  print('b is $b.');
  print('c is $c.');
}
```

Null Assertion Error

- For null assertion operator !, if the value happens to be null, it will throw error.
- There are two ways of handling this situation.
  - Use null-aware operators
  - Use type promotion

Null-Aware Operators

- Null-aware operators can be used to handle nullable values
- Conditional property access (?.) will return null if the value is null
- Null-coalescing operators (??) will return alternative value if the value is null

```
// Conditional property access
nullableObject?.action();

// Null-coalescing operators
print(nullableString ?? 'alternate’);
print(nullableString != null ? nullableString : 'alternate');
```

Type Promotion

- Dart’s compiler can analyze your code flow.
- Nullable variables that can’t possibly contain null values are treated like non-nullable variables.
- Dart’s type system can track where variables are assigned and read, and can verify that non-nullable variables are given values before any code tries to read from them. This process is called definite assignment.

Example of Type Promotion

- Code on the left will give error but code on the right will not.

```
void main() {
  String text;

  print(text);
  print(text.length);
}
```

```
void main() {
  String text;

  if (DateTime.now().hour < 12) {
    text = "Morning! Let's make aloo paratha!";
  } else {
    text = "Afternoon! Let's make biryani!";
  }

  print(text);
  print(text.length);
}
```

Exercise on Null Checking

- What does the following code output?

```
int getLength(String? str) {
  return str?.length ?? 0;
}

void main() {
  print(getLength('This is a string!’));
  print(getLength(null));
}
```

Late Assignment

- When creating a class, fields should be non-nullable but often times they are assigned value when we create objects of that class.
- We put late keyword in front that tells Dart:
  - Don’t assign that variable a value yet.
  - We will assign it a value later.
  - We’ll make sure that the variable has a value before the variable is used.
- If we declare late variable and violate above condition, an error will be thrown.

Example of late assignment

- _description is non-nullable but it can wait to be assigned after object creation.

```
class Meal {
  late String _description;

  set description(String desc) {
    _description = 'Meal description: $desc';
  }

  String get description => _description;
}

void main() {
  final myMeal = Meal();
  myMeal.description = 'Feijoada!';
  print(myMeal.description);
}
```

Late Circular References

- Late can help with circular references.
- Note that late final means you can assign later but once assigned, it will become read-only.

```
class Team {
  late final Coach coach;
}

class Coach {
  late final Team team;
}

void main() {
  final myTeam = Team();
  final myCoach = Coach();
  myTeam.coach = myCoach;
  myCoach.team = myTeam;

  print('All done!');
}
```

Lazy Initialization

- Lazy initiailzation will execute the initialzer when being read.
- _computeValue() gets called only when provider.value is trying to read value.

```
int _computeValue() {
  print('In _computeValue...');
  return 3;
}

class CachedValueProvider {
  late final _cache = _computeValue();
  int get value => _cache;
}

void main() {
  print('Calling constructor...');
  var provider = CachedValueProvider();
  print('Getting value...');
  print('The value is ${provider.value}!');
}
```

In-Class Exercise

- Complete Asynchronous programming and Null safety codelab
  - https://dart.dev/codelabs/async-await
  - https://dart.dev/codelabs/null-safety
- Inform staff after you complete all exercises