|  |
| --- |
| **NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY (NUST)** |
|  |
| **SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE (SEECS)** |
| EC-303 Mobile Application Development for Small and Medium Enterprises |
| LAB-03  Generics, Asynchronous Programming & Iterable Collections |

SYEDA SANA ZEHRA ZAIDI

339278

BESE-11 A

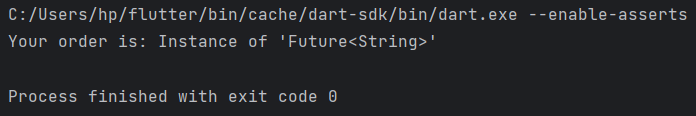
**Task 1**

Follow the guidelines mentioned in <https://dart.dev/codelabs/async-await>and try solving the example problems. (ETA 30 – 60 minutes)

**Note:** You can ask the instructor or lab engineer regarding any issues or challenges that you may face while exploring the task.

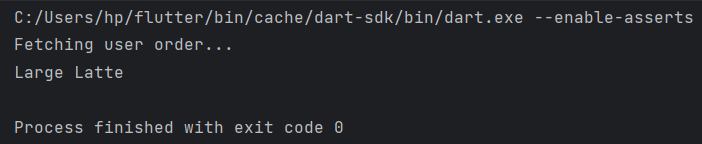
**Example 1: Incorrect Use of Async and Await**

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

****

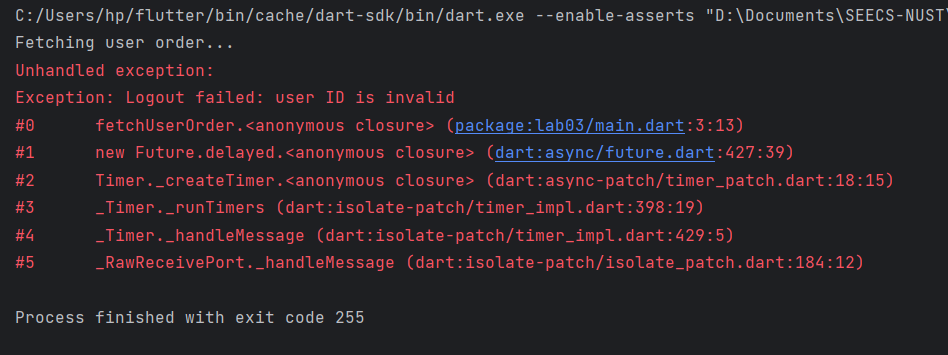
**Example 2: Introducing Futures**

|  |
| --- |
| Future<void> fetchUserOrder() {  return Future.delayed(const Duration(seconds: 2), () => print('Large Latte')); }  void main() {  fetchUserOrder();  print('Fetching user order...'); } |

****

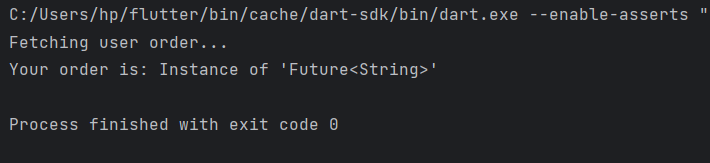
**Example 3: Completing with an Error**

|  |
| --- |
| Future<void> fetchUserOrder() {  return Future.delayed(const Duration(seconds: 2),  () => throw Exception('Logout failed: user ID is invalid')); }  void main() {  fetchUserOrder();  print('Fetching user order...'); } |



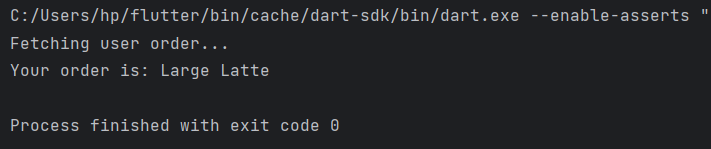
**Example 4: Synchronous Functions**

|  |
| --- |
| 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()); } |



**Example 4: Asynchronous Functions**

|  |
| --- |
| 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()); } |



**Example 5: Execution within Async Function**

|  |
| --- |
| Future<void> printOrderMessage() async {  print('Awaiting user order...');  var order = await fetchUserOrder();  print('Your order is: $order'); }  Future<String> fetchUserOrder() {  return Future.delayed(const Duration(seconds: 4), () => 'Large Latte'); }  void main() async {  countSeconds(4);  await printOrderMessage(); }  void countSeconds(int s) {  for (var i = 1; i <= s; i++) {  Future.delayed(Duration(seconds: i), () => print(i));  } } |

****

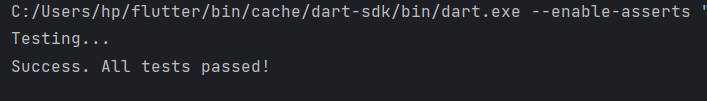
**Reversing Line 2 and 3**

|  |
| --- |
| Future<void> printOrderMessage() async {  var order = await fetchUserOrder();  print('Awaiting user order...');  print('Your order is: $order'); }  Future<String> fetchUserOrder() {  return Future.delayed(const Duration(seconds: 4), () => 'Large Latte'); }  void main() async {  countSeconds(4);  await printOrderMessage(); }  void countSeconds(int s) {  for (var i = 1; i <= s; i++) {  Future.delayed(Duration(seconds: i), () => print(i));  } } |

****

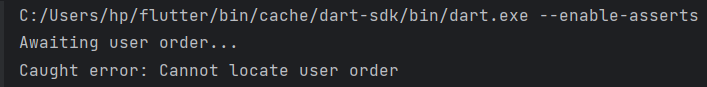
**Example 6: Practice using Async and Await**

|  |
| --- |
| Future<String> reportUserRole() async {  final username = await fetchRole();  return 'User role: $username'; }  Future<String> reportLogins() async {  final logins = await fetchLoginAmount();  return 'Total number of logins: $logins'; }  Future<String> fetchRole() => Future.delayed(\_halfSecond, () => \_role); Future<int> fetchLoginAmount() => Future.delayed(\_halfSecond, () => \_logins);  void main() async {  print('Testing...');  List<String> messages = [];  const passed = 'PASSED';  const testFailedMessage = 'Test failed for the function:';  const typoMessage = 'Test failed! Check for typos in your return value';  try {  messages  ..add(\_makeReadable(  testLabel: 'Part 1',  testResult: await \_asyncEquals(  expected: 'User role: administrator',  actual: await reportUserRole(),  typoKeyword: \_role,  ),  readableErrors: {  typoMessage: typoMessage,  'null':  'Test failed! Did you forget to implement or return from reportUserRole?',  'User role: Instance of \'Future<String>\'':  '$testFailedMessage reportUserRole. Did you use the await keyword?',  'User role: Instance of \'\_Future<String>\'':  '$testFailedMessage reportUserRole. Did you use the await keyword?',  'User role:':  '$testFailedMessage reportUserRole. Did you return a user role?',  'User role: ':  '$testFailedMessage reportUserRole. Did you return a user role?',  'User role: tester':  '$testFailedMessage reportUserRole. Did you invoke fetchRole to fetch the user\'s role?',  }))  ..add(\_makeReadable(  testLabel: 'Part 2',  testResult: await \_asyncEquals(  expected: 'Total number of logins: 42',  actual: await reportLogins(),  typoKeyword: \_logins.toString(),  ),  readableErrors: {  typoMessage: typoMessage,  'null':  'Test failed! Did you forget to implement or return from reportLogins?',  'Total number of logins: Instance of \'Future<int>\'':  '$testFailedMessage reportLogins. Did you use the await keyword?',  'Total number of logins: Instance of \'\_Future<int>\'':  '$testFailedMessage reportLogins. Did you use the await keyword?',  'Total number of logins: ':  '$testFailedMessage reportLogins. Did you return the number of logins?',  'Total number of logins:':  '$testFailedMessage reportLogins. Did you return the number of logins?',  'Total number of logins: 57':  '$testFailedMessage reportLogins. Did you invoke fetchLoginAmount to fetch the number of user logins?',  }))  ..removeWhere((m) => m.contains(passed))  ..toList();   if (messages.isEmpty) {  print('Success. All tests passed!');  } else {  messages.forEach(print);  }  } on UnimplementedError {  print(  'Test failed! Did you forget to implement or return from reportUserRole?');  } catch (e) {  print('Tried to run solution, but received an exception: $e');  } }  const \_role = 'administrator'; const \_logins = 42; const \_halfSecond = Duration(milliseconds: 500);  // Test helpers. String \_makeReadable({  required String testResult,  required Map<String, String> readableErrors,  required String testLabel, }) {  if (readableErrors.containsKey(testResult)) {  var readable = readableErrors[testResult];  return '$testLabel $readable';  } else {  return '$testLabel $testResult';  } }  Future<String> \_asyncEquals({  required String expected,  required dynamic actual,  required String typoKeyword, }) async {  var strActual = actual is String ? actual : actual.toString();  try {  if (expected == actual) {  return 'PASSED';  } else if (strActual.contains(typoKeyword)) {  return 'Test failed! Check for typos in your return value';  } else {  return strActual;  }  } catch (e) {  return e.toString();  } } |

****

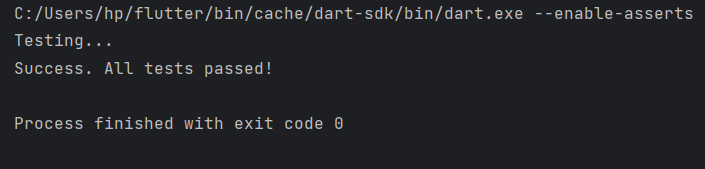
**Example 7: Async and Await with Try-Catch**

|  |
| --- |
| Future<String> fetchUserOrder() {  var str = Future.delayed(  const Duration(seconds: 4), () => throw 'Cannot locate user order');  return str; }  void main() async {  await printOrderMessage(); } |

****

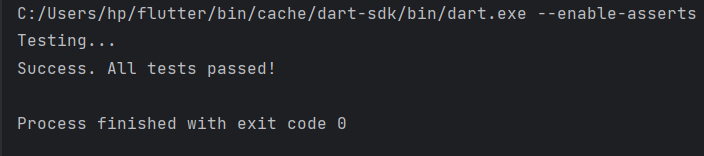
**Example 8: Practice Handling Errors**

|  |
| --- |
| Future<String> changeUsername() async {  try {  return await fetchNewUsername();  } catch (err) {  return err.toString();  } }  Future<String> fetchNewUsername() =>  Future.delayed(const Duration(milliseconds: 500), () => throw UserError());  class UserError implements Exception {  @override  String toString() => 'New username is invalid'; }  void main() async {  final List<String> messages = [];  const typoMessage = 'Test failed! Check for typos in your return value';   print('Testing...');  try {  messages  ..add(\_makeReadable(  testLabel: '',  testResult: await \_asyncDidCatchException(changeUsername),  readableErrors: {  typoMessage: typoMessage,  \_noCatch:  'Did you remember to call fetchNewUsername within a try/catch block?',  }))  ..add(\_makeReadable(  testLabel: '',  testResult: await \_asyncErrorEquals(changeUsername),  readableErrors: {  typoMessage: typoMessage,  \_noCatch:  'Did you remember to call fetchNewUsername within a try/catch block?',  }))  ..removeWhere((m) => m.contains(\_passed))  ..toList();   if (messages.isEmpty) {  print('Success. All tests passed!');  } else {  messages.forEach(print);  }  } catch (e) {  print('Tried to run solution, but received an exception: $e');  } }  // Test helpers. String \_makeReadable({  required String testResult,  required Map<String, String> readableErrors,  required String testLabel, }) {  if (readableErrors.containsKey(testResult)) {  final readable = readableErrors[testResult];  return '$testLabel $readable';  } else {  return '$testLabel $testResult';  } }  Future<String> \_asyncErrorEquals(Function fn) async {  final result = await fn();  if (result == UserError().toString()) {  return \_passed;  } else {  return 'Test failed! Did you stringify and return the caught error?';  } }  Future<String> \_asyncDidCatchException(Function fn) async {  var caught = true;  try {  await fn();  } on UserError catch (\_) {  caught = false;  }   if (caught == false) {  return \_noCatch;  } else {  return \_passed;  } }  const \_passed = 'PASSED'; const \_noCatch = 'NO\_CATCH'; |

****

**Example 9: Putting it all Together**

|  |
| --- |
| // Part 1 String addHello(String user) => 'Hello $user';  // Part 2 // Call the provided async function fetchUsername() // to return the username. Future<String> greetUser() async {  final username = await fetchUsername();  return addHello(username); }  // Part 3 // Call the provided async function logoutUser() // to log out the user. Future<String> sayGoodbye() async {  try {  final result = await logoutUser();  return '$result Thanks, see you next time';  } catch (e) {  return 'Failed to logout user: $e';  } }  Future<String> fetchUsername() => Future.delayed(\_halfSecond, () => 'Jean');  Future<String> logoutUser() => Future.delayed(\_halfSecond, \_failOnce);  void main() async {  const didNotImplement =  'Test failed! Did you forget to implement or return from';   final List<String> messages = [];   print('Testing...');  try {  messages  ..add(\_makeReadable(  testLabel: 'Part 1',  testResult: await \_asyncEquals(  expected: 'Hello Jerry',  actual: addHello('Jerry'),  typoKeyword: 'Jerry'),  readableErrors: {  \_typoMessage: \_typoMessage,  'null': '$didNotImplement addHello?',  'Hello Instance of \'Future<String>\'':  'Looks like you forgot to use the \'await\' keyword!',  'Hello Instance of \'\_Future<String>\'':  'Looks like you forgot to use the \'await\' keyword!',  }))  ..add(\_makeReadable(  testLabel: 'Part 2',  testResult: await \_asyncEquals(  expected: 'Hello Jean',  actual: await greetUser(),  typoKeyword: 'Jean'),  readableErrors: {  \_typoMessage: \_typoMessage,  'null': '$didNotImplement greetUser?',  'HelloJean':  'Looks like you forgot the space between \'Hello\' and \'Jean\'',  'Hello Instance of \'Future<String>\'':  'Looks like you forgot to use the \'await\' keyword!',  'Hello Instance of \'\_Future<String>\'':  'Looks like you forgot to use the \'await\' keyword!',  '{Closure: (String) => dynamic from Function \'addHello\': static.(await fetchUsername())}':  'Did you place the \'\$\' character correctly?',  '{Closure \'addHello\'(await fetchUsername())}':  'Did you place the \'\$\' character correctly?',  }))  ..add(\_makeReadable(  testLabel: 'Part 3',  testResult: await \_asyncDidCatchException(sayGoodbye),  readableErrors: {  \_typoMessage:  '$\_typoMessage. Did you add the text \'Thanks, see you next time\'?',  'null': '$didNotImplement sayGoodbye?',  \_noCatch:  'Did you remember to call logoutUser within a try/catch block?',  'Instance of \'Future<String>\' Thanks, see you next time':  'Did you remember to use the \'await\' keyword in the sayGoodbye function?',  'Instance of \'\_Future<String>\' Thanks, see you next time':  'Did you remember to use the \'await\' keyword in the sayGoodbye function?',  }))  ..add(\_makeReadable(  testLabel: 'Part 3',  testResult: await \_asyncEquals(  expected: 'Success! Thanks, see you next time',  actual: await sayGoodbye(),  typoKeyword: 'Success'),  readableErrors: {  \_typoMessage:  '$\_typoMessage. Did you add the text \'Thanks, see you next time\'?',  'null': '$didNotImplement sayGoodbye?',  \_noCatch:  'Did you remember to call logoutUser within a try/catch block?',  'Instance of \'Future<String>\' Thanks, see you next time':  'Did you remember to use the \'await\' keyword in the sayGoodbye function?',  'Instance of \'\_Future<String>\' Thanks, see you next time':  'Did you remember to use the \'await\' keyword in the sayGoodbye function?',  'Instance of \'\_Exception\'':  'CAUGHT Did you remember to return a string?',  }))  ..removeWhere((m) => m.contains(\_passed))  ..toList();   if (messages.isEmpty) {  print('Success. All tests passed!');  } else {  messages.forEach(print);  }  } catch (e) {  print('Tried to run solution, but received an exception: $e');  } }  // Test helpers. String \_makeReadable({  required String testResult,  required Map<String, String> readableErrors,  required String testLabel, }) {  String? readable;  if (readableErrors.containsKey(testResult)) {  readable = readableErrors[testResult];  return '$testLabel $readable';  } else if ((testResult != \_passed) && (testResult.length < 18)) {  readable = \_typoMessage;  return '$testLabel $readable';  } else {  return '$testLabel $testResult';  } }  Future<String> \_asyncEquals({  required String expected,  required dynamic actual,  required String typoKeyword, }) async {  final strActual = actual is String ? actual : actual.toString();  try {  if (expected == actual) {  return \_passed;  } else if (strActual.contains(typoKeyword)) {  return \_typoMessage;  } else {  return strActual;  }  } catch (e) {  return e.toString();  } }  Future<String> \_asyncDidCatchException(Function fn) async {  var caught = true;  try {  await fn();  } on Exception catch (\_) {  caught = false;  }   if (caught == true) {  return \_passed;  } else {  return \_noCatch;  } }  const \_typoMessage = 'Test failed! Check for typos in your return value'; const \_passed = 'PASSED'; const \_noCatch = 'NO\_CATCH'; const \_halfSecond = Duration(milliseconds: 500);  String \_failOnce() {  if (\_logoutSucceeds) {  return 'Success!';  } else {  \_logoutSucceeds = true;  throw Exception('Logout failed');  } }  bool \_logoutSucceeds = false; |

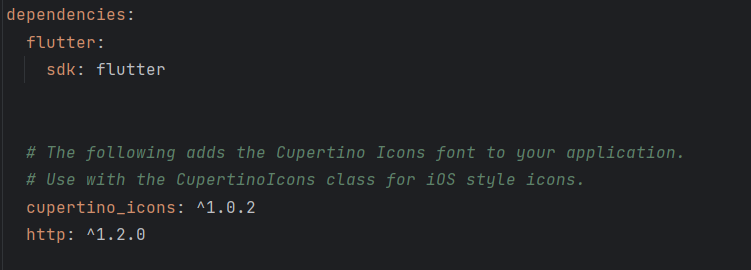
****

**Task 2**

**Part-I**

Part 1 requires you to modify the given classes and provide functionality for following:

* Create pubspec.yaml file to load the http library into the code.



* Write your model class to store the contents of the JSON file.

|  |
| --- |
| class MyModel {  List<Article> results;   MyModel({required this.results});   factory MyModel.fromJson(Map<String, dynamic> json) {  List<Article> articles = [];  if (json['results'] != null) {  json['results'].forEach((articleJson) {  articles.add(Article.fromJson(articleJson));  });  }  return MyModel(results: articles);  } }  class Article {  String url;  String title;  String contentType;  List<String> tags;  String date;   Article({  required this.url,  required this.title,  required this.contentType,  required this.tags,  required this.date,  });   factory Article.fromJson(Map<String, dynamic> json) {  return Article(  url: json['url'],  title: json['title'],  contentType: json['contentType'],  tags: List<String>.from(json['tags']),  date: json['date'],  );  } } |

* Write code to receive and parse response from the APIs in a JSON format.

|  |
| --- |
| import 'dart:convert'; import 'package:http/http.dart' as http; import 'models.dart';  Future<MyModel> fetchData() async {  final response = await http  .get(Uri.*parse*('https://codewithandrea.com/search/search.json'));   if (response.statusCode == 200) {  Map<String, dynamic> data = json.decode(response.body);  return MyModel.fromJson(data);  } else {  throw Exception('Failed to load data');  } } |

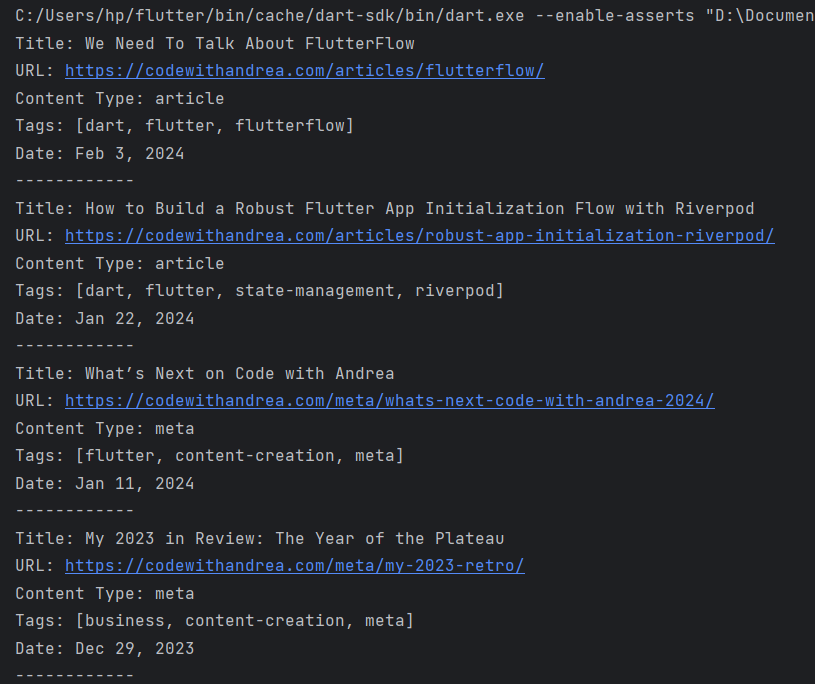
* You should also write code for initializing your model classes from JSON object(s).

|  |
| --- |
| factory Article.fromJson(Map<String, dynamic> json) {  return Article(  url: json['url'],  title: json['title'],  contentType: json['contentType'],  tags: List<String>.from(json['tags']),  date: json['date'],  ); } |

* You can use following URLs to fetch and parse the JSON: <https://codewithandrea.com/search/search.json>
* Write the main method with all cases covered for this task

|  |
| --- |
| void main() async {  try {  MyModel myModel = await fetchData();   for (Article article in myModel.results) {  print('Title: ${article.title}');  print('URL: ${article.url}');  print('Content Type: ${article.contentType}');  print('Tags: ${article.tags}');  print('Date: ${article.date}');  print('------------');  }  } catch (e) {  print('Error: $e');  } } |

* You can modify existing method signatures and write additional classes / methods to complete this task.



**Part-II**

Part 2 requires you to perform following tasks:

* Modify the code in Part 1 to make it generic and be able to parse and return any given model class(es)

|  |
| --- |
| class Model<T> {  List<T> results;   Model({required this.results});   factory Model.fromJson(  List<dynamic> jsonArray, Function(Map<String, dynamic>) fromJson) {  List<T> items = [];  if (jsonArray != null) {  jsonArray.forEach((itemJson) {  items.add(fromJson(itemJson));  });  }  return Model<T>(results: items);  } } |

* You should also write the model class(es) and methods for initializing your model classes from JSON object(s).

|  |
| --- |
| class Photo {  int albumId;  int id;  String title;  String url;  String thumbnailUrl;   Photo({  required this.albumId,  required this.id,  required this.title,  required this.url,  required this.thumbnailUrl,  });   factory Photo.fromJson(Map<String, dynamic> json) {  return Photo(  albumId: json['albumId'],  id: json['id'],  title: json['title'],  url: json['url'],  thumbnailUrl: json['thumbnailUrl'],  );  } } |

* You can use following URL to fetch the JSON and verify your functionality. <https://jsonplaceholder.typicode.com/photos>

|  |
| --- |
| import 'dart:convert'; import 'package:http/http.dart' as http; import 'models.dart';  Future<Model<Photo>> fetchPhotoData(String apiUrl) async {  final response = await http.get(Uri.*parse*(apiUrl));   if (response.statusCode == 200) {  List<dynamic> data = json.decode(response.body);  return Model<Photo>.fromJson(data, (json) => Photo.fromJson(json));  } else {  throw Exception('Failed to load data');  } } |

* Write the main method with all cases covered for this task

|  |
| --- |
| void main() async {  try {  // Fetch and print Photos  Model<Photo> photoModel =  await fetchPhotoData('https://jsonplaceholder.typicode.com/photos');   for (Photo photo in photoModel.results) {  print('Album ID: ${photo.albumId}');  print('Photo ID: ${photo.id}');  print('Title: ${photo.title}');  print('URL: ${photo.url}');  print('Thumbnail URL: ${photo.thumbnailUrl}');  print('------------');  }  } catch (e) {  print('Error: $e');  } } |

* You can modify existing method signatures and write additional classes / methods to complete this task.

