**Dart Advanced Concepts (Null Safety)**

Dart has supported sound null safety since **version 2.12**

In null safety, variables cannot be null unless you explicitly specify that they can

**Before Dart 2.12**

you can assign null to the message variable and access the length property. However, it’ll cause a runtime error:

void main() {

String messge = null;

print(message.length); // runtime error

}

**Now**

• With null safety, types in the code are non-nullable by default. If you attempt to assign null to a variable, the code editor will issue an error. In other words, null safety turns the runtime errors into edit-time errors. This makes your code more robust.

• To specify that a variable can be null, you add a question mark (?) to the type in variable declaration. For example:

void main() {

String? message = 'Hello';

print(message);

message = null; // OK

}

**Nullable types**

A nullable type contains null in addition to its own values of the type.

For example :

• int? – a nullable integer such as 1, 2, and null.

• double? – a nullable double such as 3.14, 2.5, and null.

• bool? – nullable boolean such as true, false, and null.

• String? – a nullable string such as ‘Hello’, ‘Bye’, and null.

• Point? a nullable user-defined class Point.

For example, point(10,20) and null. In Dart, every non-nullable type has a corresponding nullable type. By adding the ?, y

**Null Safety**

**Working with nullable types**

Before null safety, it’s easy to forget to add the code that handles null. However, with null safety, Dart makes it impossible to forget. Because you really cannot do much with null unless you deal with the null possibility.

• For example, Dart will not allow you to run the following code:

void main() {

String? message;

print(message.length);

}

• It issues a compile-time error.

Dart has a tool called Dart analyzer that is smart enough to tell if a nullable variable contains null or not

**So**

**• Type promotion**

Type promotion allows you to assign a value to a nullable variable without requiring any extra work. For example:

void main() {

String? message;

message = 'Hello';

print(message.length);

}

Therefore, Dart implicitly promotes the type of the message variable from String? to String automatically.

**• Flow analysis**

Besides type promotion, Dart uses a sophisticated flow analysis to check every possible case the code would take. And if none of these cases come up with the possibility of being null, it promotes the variable to a non-nullable type using type promotion:

bool isEven(int? x) {

if (x == null) {

return false;

}

return x.isEven;

}

**Dart Null-aware Operators**

To deal with null values, Dart uses flow analysis and type promotion. In addition, it provides you with various null-aware operators:

|  |  |
| --- | --- |
| Operator | Meaning |
| ?? | The if-null operator |
| ??= | The null-aware assignment operator |
| ?. | The null-aware access operator |
| ! | Null assertion operator |
| ?[] | Null-aware index operator |
| …? | Null-aware spread operator |

**The if-null operator (??)**

|  |  |
| --- | --- |
| Without (??) | With (??) |
| void main() {  String? input;  message;  if (input != null) {  message = input;  } else {  message = 'Error';  }  print(message);  } | void main() {  String? input;  String message = input ?? 'Error'; print(message); } Output: Error |

**• The null-aware assignment operator (??=)**

|  |  |
| --- | --- |
| Without(??=) | With(??=) |
| input = input ?? 'Error'; | input ??= 'Error' |

**• The null-aware access operator (?.)**

Suppose you have a nullable string variable called input like this: String? input;

|  |  |
| --- | --- |
| Without (?.) | With(?.) |
| void main() {  String? input;  if (input != null) {  print(input.length);  print(input.toLowerCase());  }  } | void main() {  String? input;  print(input?.length); // null  print(input?.toLowerCase()); // null  } |

**• Null assertion operator (!)**

The following code causes a compile-time error:

bool? isTextFile(String? filename) {

if (filename != null) {

return filename.endsWith('.txt') ? true : false;

}

return null;

}

void main() {

bool result = isTextFile('readme.txt'); print(result);

}

Since you’re sure that the isTextFile('readme.txt') doesn’t return a null value, you can use the null assertion operator (!)

**Null-aware index operator (?[])**

The null-aware index operator ?[] allows you to access an element of a list when the list might be null.

void main() {

List? scores = [1, 2, 3, 4, 5]; // somewhere in the code

scores = null;

print(scores?[3]); // null

}

**Null-aware spread operator (…?)**

The null-aware spread operator …? allows you to spread list in another when the list might be null without error.

void main() {

List?<int> num

List<int> numEx = […?num,1, 2, 3, 4, 5]; // somewhere in the code

print(numEx); // [1,2,3,4,5]

}