Dart – Null Safety

Null Safety in simple words means a variable cannot contain a ‘null’ value unless you initialized with null to that variable. With null safety, all the runtime null-dereference errors will now be shown in compile time.

```
String name = null ; // This means the variable name has a null value.
```

Example :

Dart

```
class Car {

String carName = "Aston Martin";

}



void main() {

  Car cars;

  print(cars.carName);

}
```

In the above dart code, we have created a class and inside it, we have initialized a variable named carName In the main() function we have declared a variable type of car ie, cars. But when we run the above code, an error occurs.

Output:

```
Error: Non-nullable variable 'cars' must be assigned before it can be
used.
```

In the above code, the object was not initialized. To solve this problem we have to do :

Dart

```
class Car {

String carName = "Aston Martin";

}



void main() {

  Car cars = new Car();

  print(cars.carName);

}
```

Output:

```
Aston Martin
```

In the above simple code, it is easy to identify which variable is not initialized, but in a huge codebase it might be very difficult and time-consuming to identify such errors. This issue is solved by null safety.

Null Safety Principle

- **Non-nullable By Default:** By default, Dart variables aren't nullable unless you explicitly specify that they can be null. This is because non-null was by far the most common option in API research.
- **Incrementally Adoptable:** Migration to null safety is completely up to you. You can choose what to migrate to null safety, and when. You can migrate incrementally, combining null-safe and non-null-safe code within the same project.
- **Fully Sound:** As a result of Dart's sound null safety, compiler optimizations are possible. If the type system determines that something isn't null, then it cannot be null. Null safety leads to fewer bugs, smaller binaries, and faster execution once your entire project and its dependencies are migrated to null safety.

Non-Nullable Types

When we use null safety, all the types are non-nullable by default. For example, when we declare a variable of type int, the variable will contain some integer value.

Dart

```
void main() {

  int marks;



  // The value of type `null` cannot be

  // assigned to a variable of type 'int'

  marks = null;

}
```

Non-nullable variables must always be initialized with non-null values.

Nullable Types

To specify if the variable can be null, then you can use the nullable type   operator, Lets see an example:

```
String? carName;  // initialized to null by default

int? marks = 36;  // initialized to non-null

marks = null; // can be re-assigned to null
```

Note: You don’t need to initialize a nullable variable before using it. It is initialized to null by default.

The Assertion Operator (!)

Use the null assertion operator (!) to make Dart treat a nullable expression as non-nullable if you’re certain it isn’t null.

Example:

```
int? someValue = 30;

int data = someValue!; // This is valid as value is non-nullable
```

In the above example, we are telling Dart that the variable   is null, and it is safe to assign it to a non-nullable variable i.e.

Type Promotion :

Dart’s analyzer, which tells you what compile-time errors and warnings, is intelligent enough to determine whether a nullable variable is guaranteed to have values that are not null. Dart uses **Flow Analysis** at runtime for type promotion (flow analysis is a mechanism that determines the control flow of a program).

Example:

Dart

```
int checkValue(int? someValue) {

  if (someValue == null) {

    return 0;

  }

  // At this point the value is not null.

  return someValue.abs();

}



void main(){

  print(checkValue(5));

  print(checkValue(null));

}
```

In the above code, if statement checks if the value is null or not. After the if statement value cannot be null and is treated (promoted) as a non-nullable value. This allows us to safely use `someValue.abs()` instead of `someValue?.abs()` (with the null-aware operator). Here `.abs()`function will return an absolute value.

Late Keyword

As we knew that all variables are non-null by default, we can use either the? operator or the *late* keyword.

Example:

Dart

```
String? carName;       // using ? operator

late String bikeName;  // using "late" keyword
```

Benefits of Null Safety:

- It provides a better development environment with fewer runtime errors.
- Dart Compiler can optimize our code, which will led to smaller and faster programs.