Null Safety

By: Abla Saad

Understanding Null Safety

Null safety is a concept that helps us to avoid one of the most common issues in programming - null reference errors. These errors occur when we try to access properties or methods on a null value. With Dart's null safety, we can prevent these errors at compile time, making our code safer and more predictable.

What is Null Safety?

Null safety is a feature in Dart that helps us distinguish between nullable variables and non-nullable variables. A nullable variable is one that can hold either a non-null value or a null value. On the other hand, a non-nullable variable is one that must always hold a non-null value.

```
 void main() {

 int nonNullableVariable = 10; // Non-nullable variable

int? nullableVariable = null; // Nullable variable

}
```

The Problem of Null Reference Errors

Null reference errors, also known as the billion-dollar mistake, are a common type of runtime error that occurs when we try to access properties or methods on a null value. These errors can be hard to debug and can lead to unexpected behavior in our code.

```
void main() {

    String? nullableVariable = null;

   print(nullableVariable.length); // This will throw a runtime
error }
```

How Dart Null Safety Helps

Dart Null Safety helps us to avoid null reference errors by distinguishing between nullable and non-nullable variables. It also provides null safety support through null-aware operators, which allow us to perform operations on nullable variables without causing null reference errors.

```
void main() {
    String? nullableVariable = null;
    print (nullableVariable?.length); // This will not throw a
runtime error
   }
```

Non-Nullable and Nullable Types:

In Dart Null Safety, we can distinguish between non-nullable and nullable types using the '?' suffix. A non-nullable type must always contain a non-null value, while a nullable type can contain either a non-null value or a null value.

```
 void main() {
   int nonNullableVariable = 10; // Non-nullable variable
   int? nullableVariable = null; // Nullable variable
}
```

The '!' Operator:

The '!' operator, also known as the null assertion operator, converts a nullable type to a non-nullable type. If the variable is null, it throws a runtime error.

```
void main() {
    String? nullableVariable = null;
    print (nullableVariable!.length); // This will throw a runtime error
   }
```

How Dart Null Safety Solves the Problems of Null

Dart Null Safety solves the problems of null in several ways:

1. **Prevents Null Reference Errors**: By distinguishing between nullable and non-nullable variables, Dart Null Safety helps us prevent null reference errors.
2. **Improves Code Readability**: With Dart Null Safety, we can see at a glance whether a variable can be null or not. This makes our code more readable and easier to understand.
3. **Improves Performance**: Dart Null Safety can also improve the performance of our Flutter app. By catching null errors at compile time, we can avoid unnecessary runtime checks for null values.

Functions with Non-Nullable Return Types

Functions can also have non-nullable return types. This means that the function must always return a non-null value.

```
  int getLength(String str) {

    return str.length; // This function always returns a non-null
value

  }
```

Null Aware Operators:

Dart Null Safety introduces null aware operators, which allow us to perform operations on nullable variables without causing null reference errors. The most common null aware operators are **'??', '?.', and '??='.**

```
  void main() {

   String? nullableVariable = null;

     print (nullableVariable?.length); // This will not throw a runtime error

     nullableVariable ??= 'Default String'; // Assigns a default value if
nullableVariable is null}
```

1) Safe Navigation Operator (?.)

```
Void main(){

String? firstString;

String? secondString;

firstString="hussain";

print(firstString.toUpperCase());

print(secondString?.toUpperCase());
```

2) Default Null-Aware Operator (??)

```
Void main(){ String? firstString;

print(firstString??"Hello Flutter!");

}
```

3) Null-Aware Spread Operator (…?)

```
Void main(){

List<int>? firstList;

List<int>? secondList;

firstList = [2,5,4];

List<int>? thirdList

thirdList = [...firstList, print(thirdList); ?secondList] ;  }
```

4) Fallback Assignment Operator (??=)

```
void main(){    int? myNumber;
myNumber ??= 3;
print(myNumber);}
```