|  |  |
| --- | --- |
| LAB 4  INFT 3101-01 | Varun Singh (100865156)  Date submitted: 1-11-2024 |

Table of Contents

[**Topic Overview (40%)** 2](#_Toc181392662)

[Topic assigned: Generics. 2](#_Toc181392663)

[Definition: 2](#_Toc181392664)

[How and why, it is used in programming? 2](#_Toc181392665)

[**Comparison with similar or related programming concepts in Dart (or another language if applicable)** 2](#_Toc181392666)

[Advantages and limitations of using this feature in Dart. 3](#_Toc181392667)

[**Example Implementation (40%)** 4](#_Toc181392668)

[Example 1 4](#_Toc181392669)

[Example 2: 5](#_Toc181392670)

[**Real-World Scenario** 6](#_Toc181392671)

[Reason for usage: 6](#_Toc181392672)

# **Topic Overview (40%)**

## Topic assigned: Generics.

### Definition:

The term generics in Dart refers to the technology that enabled developers to write flexible and reusable code by allowing the usage of Object-oriented programming principles such as classes, methods, and functions with extended safety in a variety of types. It allows the user to create a placeholder for a type of object that can later be modified using classes or methods.

Examples: List<String> and List<Boolean>.

The examples above specify the type of data that is going to be stored in the list. The purpose of generics is to make your code safer by handling type safety errors at compile time rather than runtime.

### How and why, it is used in programming?

Some ways in which generics can be used to make the code safer and reusable are listed as follows:

* **Code Reusability**: It allows the developer to make the components of their code work with various data types. Example: List<T> can store any type of data resulting in a more flexible use of the variable.
* **Performance**: Since the type is defined and selected at compile time. The compiler can increase the code optimization and performance.
* **Type Safety**: The usage of generics enables the developer to add an extra layer of safety to his code by handling type checks at compile time, reducing the risk of type safety errors.
* **Predictable Behavior**: Generics are used in building APIs and collections to handle other types.

### **Comparison with similar or related programming concepts in Dart (or another language if applicable)**

Most of the advanced programming languages have generics introduced into them to increase performance. Some of those languages and how generics are used in them are explained below.

* TypeScript: The powerful language typescript mostly used for developing the back end in web-based applications also has generics. The generics in JavaScript and TypeScript are majorly associated with function and objects.
* C#: The generics in C# overrule the generics in Dart as they have some advanced features such as constraints which is still a newer technology in Dart and not as powerful as that of C#.
* Java: In Dart, the generic type of information is retained at runtime whereas in Java it is not the same case. The Java compiler erases the generic type of information due to type erasure. So, Dart offers a better type-safety check.

### Advantages and limitations of using this feature in Dart.

#### Benefits of using Generics in Dart:

* Enhanced Flexibility and Reusability of Code: Developers can implement the same logic of various data types with generics instead of writing for each type separately.
* Improved safety for type errors: Since all the type-safety errors are thrown at compile time it’s more type-safe as compared to fixed data types.
* Organized code: Helps the user to maintain a clean and readable code.

#### Some Limitations of using Dart Generics:

* Requires Learning: As generics is a complex topic, it can be difficult for new learners to cope with the learning curve.
* Limited Support for old/primitive data types: Generics in Dart do not support type optimization.

# **Example Implementation (40%)**

### Example 1

A screenshot of a computer program

Description automatically generated

This example illustrates a class that uses the power of generics to implement a data structure called stack. The stack can store data types like int, double, char, and string. The output of the code is as follows:

A screenshot of a computer

Description automatically generated

**Explanation:** The data structure stack uses a generic data type T, which gives it the power to store any primitive data type, such as int, double, and string. The user can call the stack method using different calls with different data types and create a stack of the desired data type, such as intStack, doubleStack, or stringStack.

**Issues and Considerations:** Using generics in general will not cause any issues. However, since the usage has been done on a data structure. The user has to handle the peek-and-pop methods when the stack does not have anything in it.

Example 2: A screenshot of a computer program

Description automatically generated

The above code uses a generic class which can be used in several different ways using the power of generics.

For example:

* One might use it to store map coordinates using a double data type.
* They can also store casual information such as name and age. Which will use 2 different data types.
* Same way as the second usage both the data types can be the same.

Possible Issues: Flexibility provided to the KeyValue pair by generics must be used carefully.

## **Real-World Scenario**

Since generics enable a developer’s code to handle different kinds of data types with less amount to code. The powerful tool can be used in performing tasks such as Form Handling and Validation for an application made in Flutter.

### Reason for usage:

Any normal forms consist of several data types for inputs and the developer needs to make sure that it is validated. Writing validation logic for different data types separately can be a hectic task and developers prefer not to repeat their code. Using generics, a single validation logic can be applied to all the inputs and the code’s length can be shortened.

#### Steps to implement

1. Define a generic validator class for the inputs let’s say Validator<T>.
2. Add a form to the Flutter app, After the submission of the form handle and validate each field through the validator function.
3. Look for any exceptions and errors that can occur and handle them.

#### How it fits in the big architecture

The topic can be pretty useful when managing state management solutions to manage the form’s state more efficiently. The introduction of Validator<T> class also makes the whole application modular as any input across the app can be validated with one function making code consistent and safer. This also makes the app more scalable and easier to read for logics dealing with different data types. They also will have a better user experience and quality of code.  
  
  
  
  
  
  
  
  
References:

*Generics*. Dart. (n.d.-b). <https://dart.dev/language/generics>

*State management*. Flutter. (n.d.). <https://docs.flutter.dev/data-and-backend/state-mgmt>

*Cookbook*. Flutter. (n.d.-a). <https://docs.flutter.dev/cookbook>

*Elevate your flutter skills with Dart Generics*. Mastering Dart Generics: Elevate Your Flutter Development. (n.d.). <https://www.dhiwise.com/post/supercharge-your-flutter-apps-with-dart-generics>

Rasmussen, M. (2020, July 11). *Generics in Dart and flutter*. Dart Academy. <https://dart.academy/generics-in-dart-and-flutter/>