Chapter 1: Dart Basics

1.1. Introduction to Dart Programming Language

Dart is a client-optimized programming language for apps on multiple platforms. It is developed by Google and is used to build mobile, desktop, server, and web applications. Dart is an object-oriented, class-based, garbage-collected language with C-style syntax. Dart can compile to either native code or JavaScript.

Dart has many similarities to other languages you may already be familiar with, such as Java, C#, Swift, and Kotlin. Dart has the following language characteristics:

- Statically typed
- Type inference
- String expressions
- Multi-paradigm including OOP and functional
- Can mimic dynamic types using `dynamic`

1.2. Dart SDK

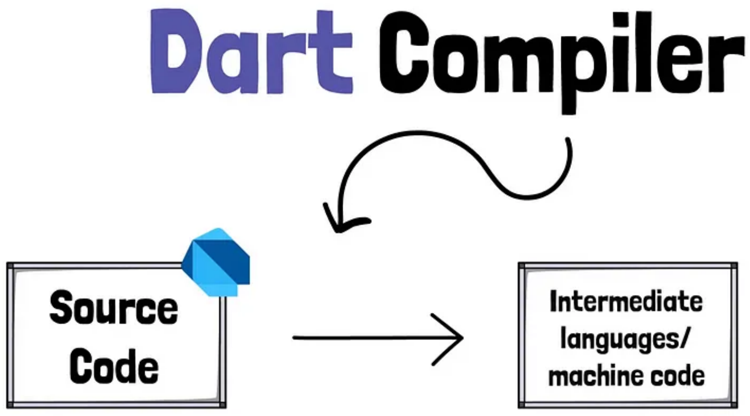
SDK means Software Development Kit. SDK is a package in which various tools are packed to develop software for a particular platform. Every platform's SDK gives you different tools, libraries, and software that can help you create the software for that platform.

Dart SDK contains seven tools that are necessary for developing dart applications.

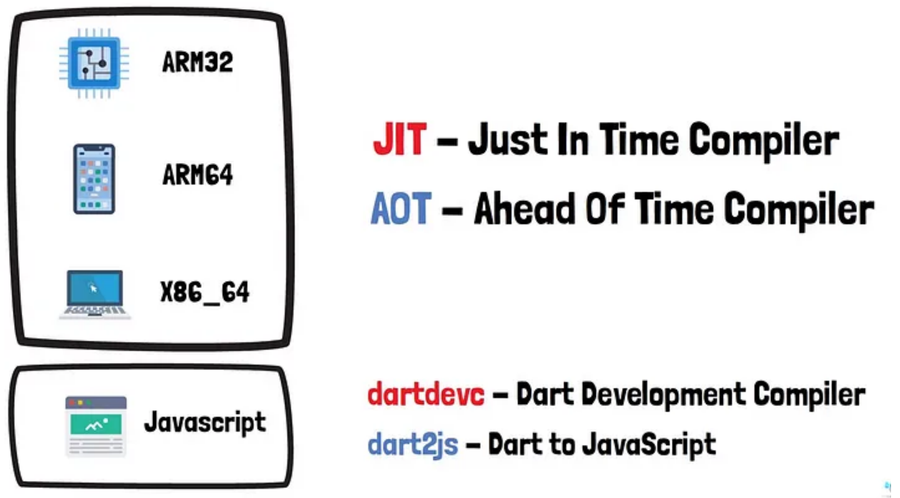
1. DartVM — The Dart Virtual Machine
2. dart2js — Dart to JavaScript transpiler (for web use only) (for deployable JavaScript)
3. dartdevc — Dart to JavaScript transpiler (for web use only) (for testing purpose)
4. dartfmt — The Dart code formatter
5. dartanalyzer — Analyze the code for errors and warnings that are specified in the dart language specification. DartPad, code editors, and IDEs such as Android Studio, IntelliJ IDEA, and VS Code use the same analysis engine that dartanalyzer uses.
6. Dartdoc — The API documentation generator.
7. pub — The Dart package manager. You can use the `pub` tool to manage Dart packages. The Flutter SDK has its own commands for managing and updating packages.

1.3. Dart VM

Dart Compiler is a tool that converts the source code we wrote in dart language to other intermediate languages or machine code that can run on a specific platform in a dart virtual machine. Like in the below figure



Dart Uses different compilers for different specific jobs. For example, the CPU architecture listed below is usually found on desktops and phones. Dart uses JIT(Just in Time Compiler) and AOT(Ahead Of Time Compiler) for native. On the web obviously, it uses a javascript compiler since it needs to translate the code into javascript language.


ARM32
ARM64
X86_64
Javascript
JIT – Just In Time Compiler
AOT – Ahead Of Time Compiler
dartdevc – Dart Development Compiler
dart2js – Dart to JavaScript

Now we are going to see the two stages of the development process,

- Development Phase and
- Production Phase.

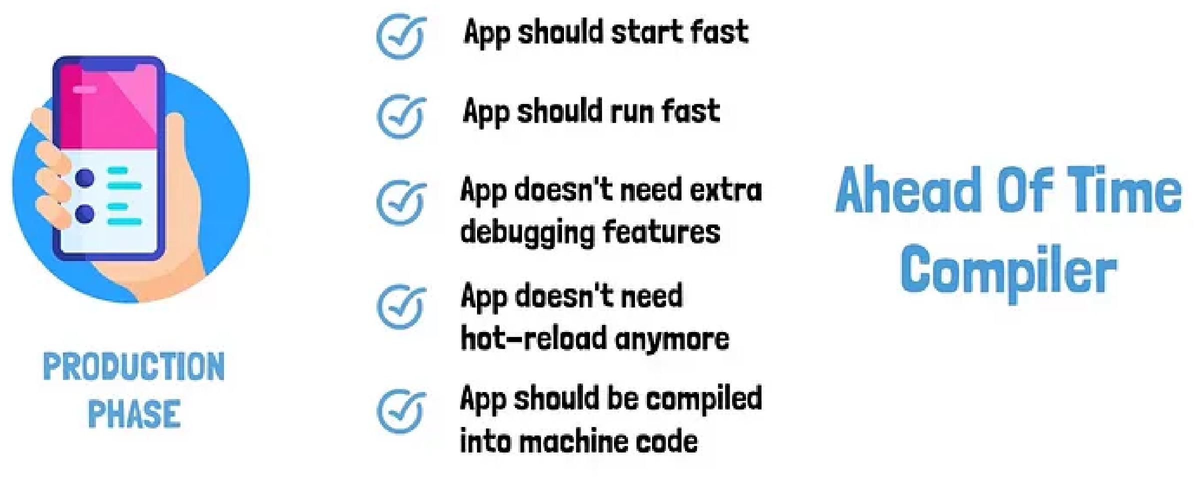
Development Phase - Just In time Compiler (JIT)

DEVELOPMENT PHASE
Easy to test the code
Easy to debug the code
Live metrics support
Fast coding cycles
Incremental recompilation
Just In Time Compiler

- JIT just compiles just the amount of code it needs.
- JIT also comes with incremental recompilation so that it only recompiles the modified part when needed.
- JIT compiler is the main star that enables the hot reload in Dart during development.

Production Phase (AOT - Ahead Of Time)

- The ahead-of-time compiler compiles the entire source code into the machine code supported natively by the platform. It does before the platform runs the program.
- When you decide to promote your code from development state to production state you need to use this compiler to do its specialized jobs.
- To benefit from the best and most optimized version of your code has drawbacks through compiling the same code from scratch over and over and it's not the best solution for developing an app into the development stage.


PRODUCTION PHASE
App should start fast
App should run fast
App doesn't need extra debugging features
App doesn't need hot-reload anymore
App should be compiled into machine code
Ahead Of Time Compiler

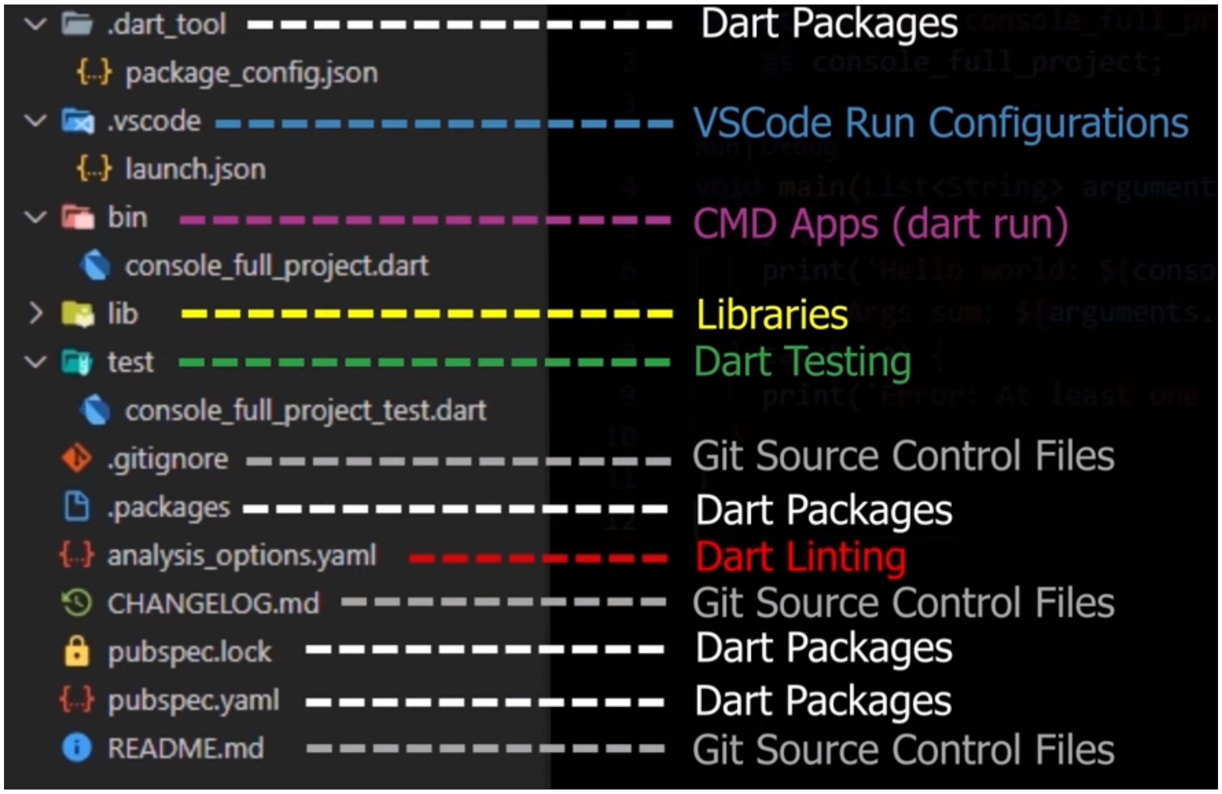
Dart VM provides an execution environment for the dart programming language. The code within the VM is running within the same isolate and it is also called an isolated dart universe with its own memory known as heap, its own thread of control called the mutator thread and its own helper thread.

Components of Dart:

- The Runtime System
- Development experience components debugging and hot reload
- JIT and AOT compilation pipelines

Dart VM can execute dart apps in 2 ways from source by using JIT and AOT compiler and from snapshots JIT, AOT, or kernel snapshots.

1.4. Dart Project Structure



1.5. Dart Variable and Types

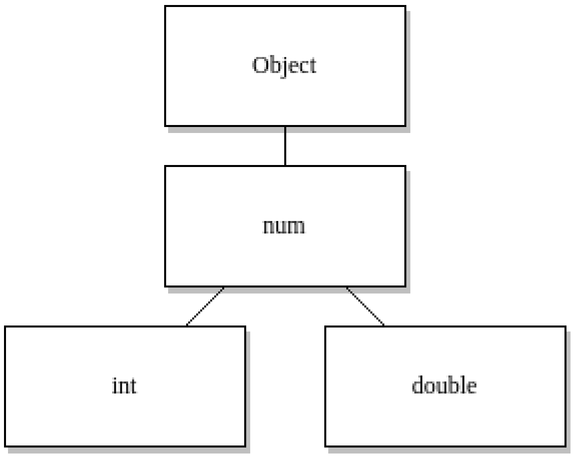
Data Types

The Dart language supports the following types:

- Numbers: **int, double**
- Strings: **String**. A Dart string is a sequence of **UTF-16** code units.
- Booleans: **bool**
- Lists, Maps: The data types list and map are used to represent a collection of objects. A List is an ordered group of objects. The List data type in Dart is synonymous to the concept of an array in other programming languages. The Map data type represents a set of values as key-value pairs. The dart: core library enables the creation and manipulation of these collections through the predefined List and Map classes respectively.

- The Dynamic Type: Dart is an optionally typed language. If the type of a variable is not explicitly specified, the variable's type is dynamic. The **dynamic** keyword can also be used as a type annotation explicitly.
- Sets:It is an unordered list of unique values of same types
- Runes: It represents Unicode values of String
- Null: It represents null value

In addition to the above types dart has the **num** type is inherited by the int and double types.



And also, In Dart, **var** automatically finds a data type. In simple terms, var says if you don't want to specify a data type, I will find a data type for you.

Note: If a variable is declared as a **dynamic**, its type can change over time. If you declare a variable as a **var**, once assigned type can not change.

Dart Variables

Variables are containers used to store value in the program. There are different types of variables where you can keep different kinds of values. Here is an example of creating a variable and initializing it.

Syntax

type variableName = value;

Rules For Creating Variables In Dart

- Variable names are case sensitive, i.e., a and A are different.
- A variable name can consist of letters and alphabets.
- A variable name cannot start with a number.
- Keywords are not allowed to be used as a variable name.
- Blank spaces are not allowed in a variable name.
- Special characters are not allowed except for the underscore (_) and the dollar ($) sign.

Const vs final

Const: If the value you have is computed at runtime (new DateTime.now(), for example), you can **not** use a const for it. However, if the value is known at compile time (const a = 1;), then you should use const over final. There are 2 other large differences between const and final. Firstly, if you're using const inside a class, you have to declare it as static const rather than just const. Secondly, if you have a const collection, everything inside of that is in const. If you have a final collection, everything inside of that is **not** final.

Final:final should be used over const if you don't know the value at compile time, and it will be calculated/grabbed at runtime. If you want an HTTP response that can't be changed, if you want to get something from a database, or if you want to read from a local file, use final. Anything that isn't known at compile time should be final over const.

1.6. Null Safety

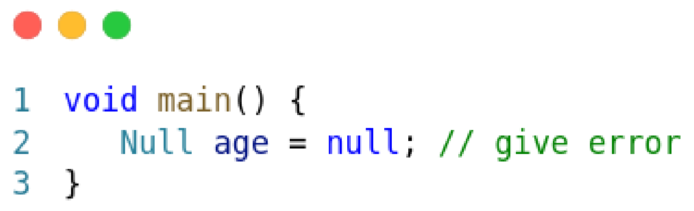
Null safety is a feature in the Dart programming language that helps developers to avoid null errors. This feature is called Sound Null Safety in dart. This allows developers to catch null errors at edit time.

Advantage Of Null Safety

- Write safe code.
- Reduce the chances of application crashes.
- Easy to find and fix bugs in code.

Using Null In Variables

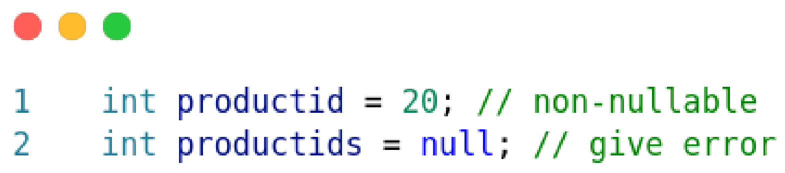
In the example below, the variable age is an int type. If you pass a null value to this variable, it will give an error instantly.



```dart
void main() {
   Null age = null; // give error
}
```

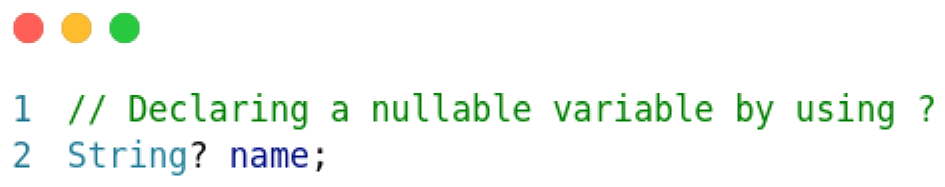
Programmers do have a lot of difficulties while handling null values. They forget that there are null values, so the program breaks. In the real world null mostly acts as a time bomb for programmers, which is ready to break the program.

In Dart, variables and fields are non-nullable by default, which means that they cannot have a value null unless you explicitly allow it.



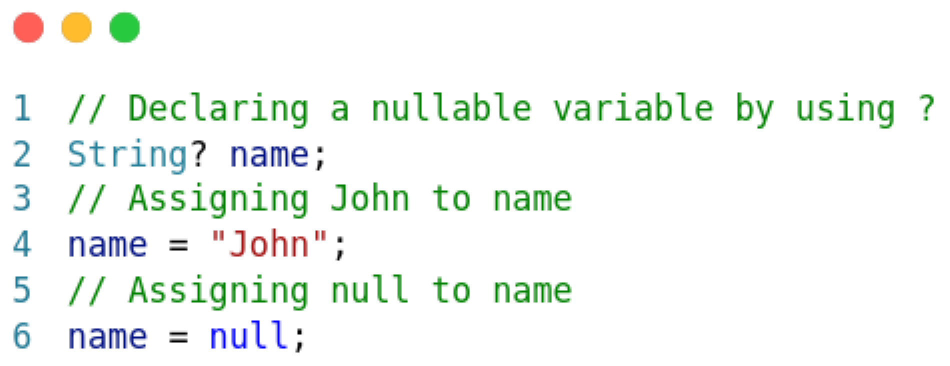
```dart
  int productid = 20; // non-nullable
  int productids = null; // give error
```

With dart sound null Safety, you cannot provide a null value by default. If you are 100% sure to use it, then you can use **?** operator after the type declaration.



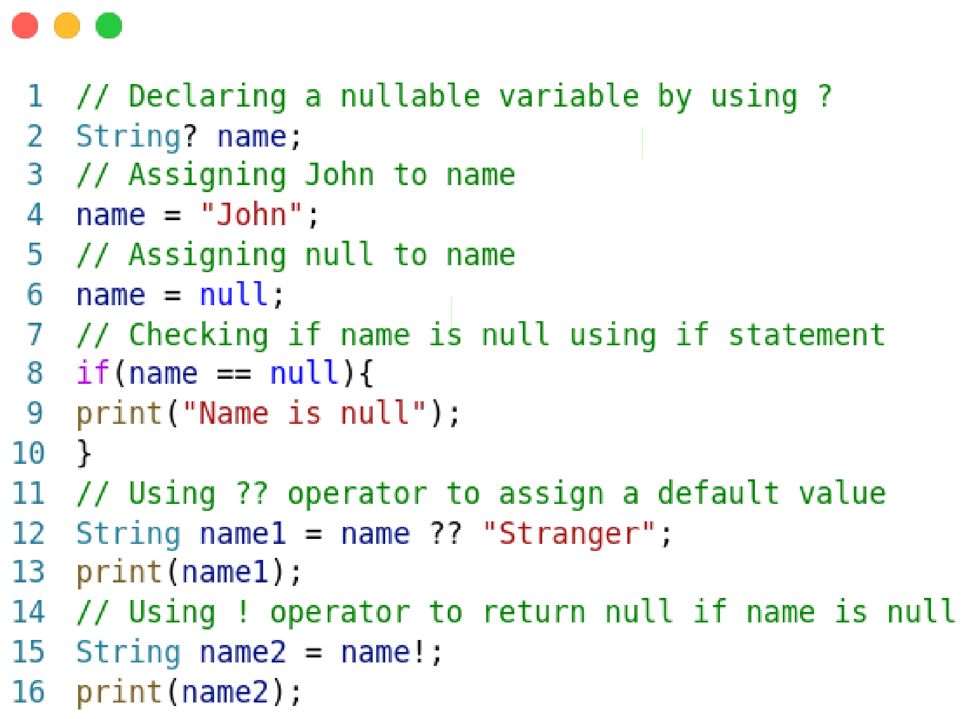
```dart
// Declaring a nullable variable by using ?
String? name;
```

You can assign a value to nullable variables just like any other variable. However, you can also assign null to them.


```
// Declaring a nullable variable by using ?
String? name;
// Assigning John to name
name = "John";
// Assigning null to name
name = null;
```

You can use nullable variables in many ways. Some of them are shown below:

- You can use **if** statement to check whether the variable is null or not.
- You can use **!** operator, which returns null if the variable is null.
- You can use **??** operator to assign a default value if the variable is null.


```
// Declaring a nullable variable by using ?
String? name;
// Assigning John to name
name = "John";
// Assigning null to name
name = null;
// Checking if name is null using if statement
if(name == null){
print("Name is null");
}
// Using ?? operator to assign a default value
String name1 = name ?? "Stranger";
print(name1);
// Using ! operator to return null if name is null
String name2 = name!;
print(name2);
```

Result

```
Name is null
Stranger
null
```

1.7. Dart Functions

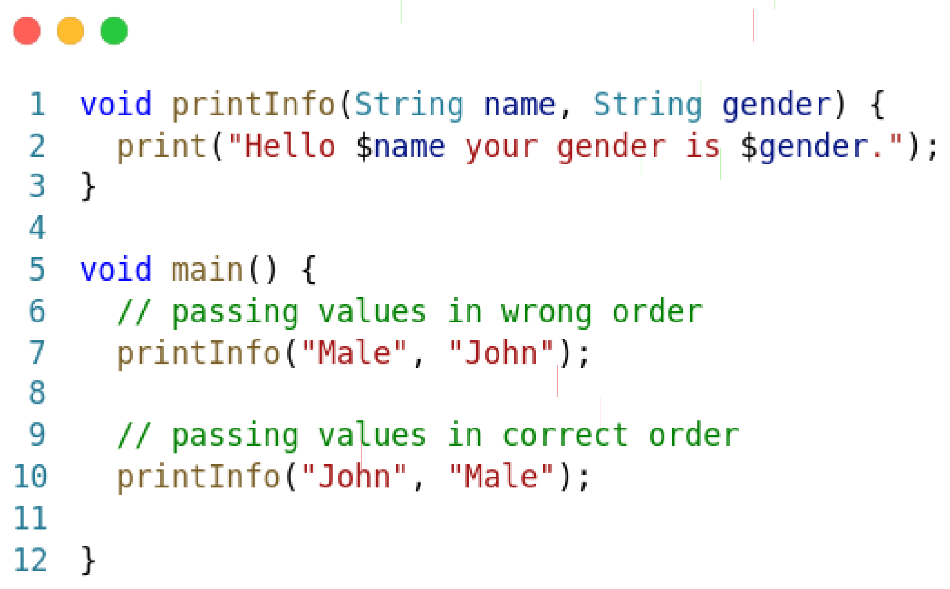
Function is a set of statements that take inputs, do some specific computation and produce output. Functions are created when certain statements are repeatedly occurring in the program and a function is created to replace them. Functions make it easy to divide the complex program into smaller sub-groups and increase the code reusability of the program.

Positional vs Named Parameter in dart function

Positional Parameter

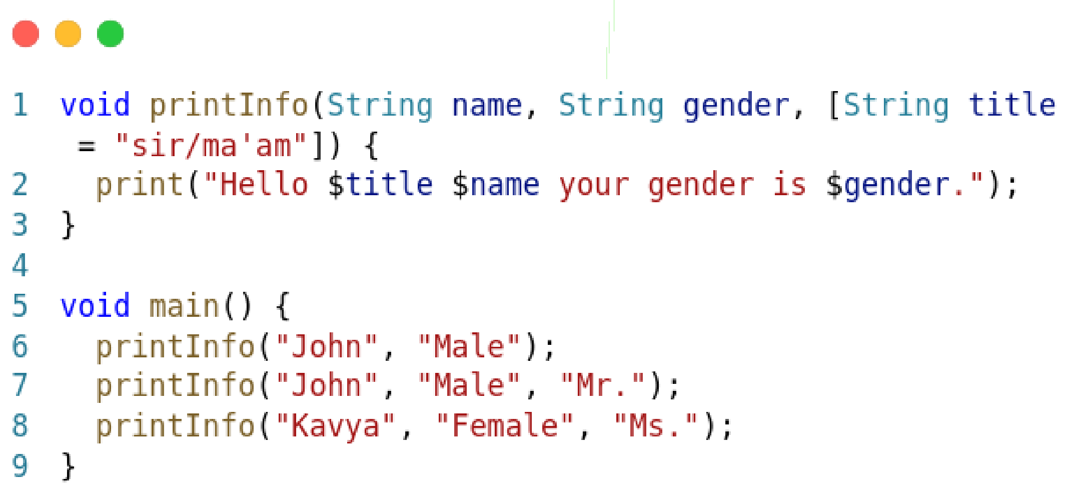
In positional parameters, you must supply the arguments in the same order as you defined on parameters when you wrote the function. If you call the function with the parameter in the wrong order, you will get the wrong result.

For example:-



```dart
void printInfo(String name, String gender) {
  print("Hello $name your gender is $gender.");
}

void main() {
  // passing values in wrong order
  printInfo("Male", "John");

  // passing values in correct order
  printInfo("John", "Male");

}
```

We can also provide a default value on positional parameter like following example:



```dart
void printInfo(String name, String gender, [String title = "sir/ma'am"]) {
  print("Hello $title $name your gender is $gender.");
}

void main() {
  printInfo("John", "Male");
  printInfo("John", "Male", "Mr.");
  printInfo("Kavya", "Female", "Ms.");
}
```

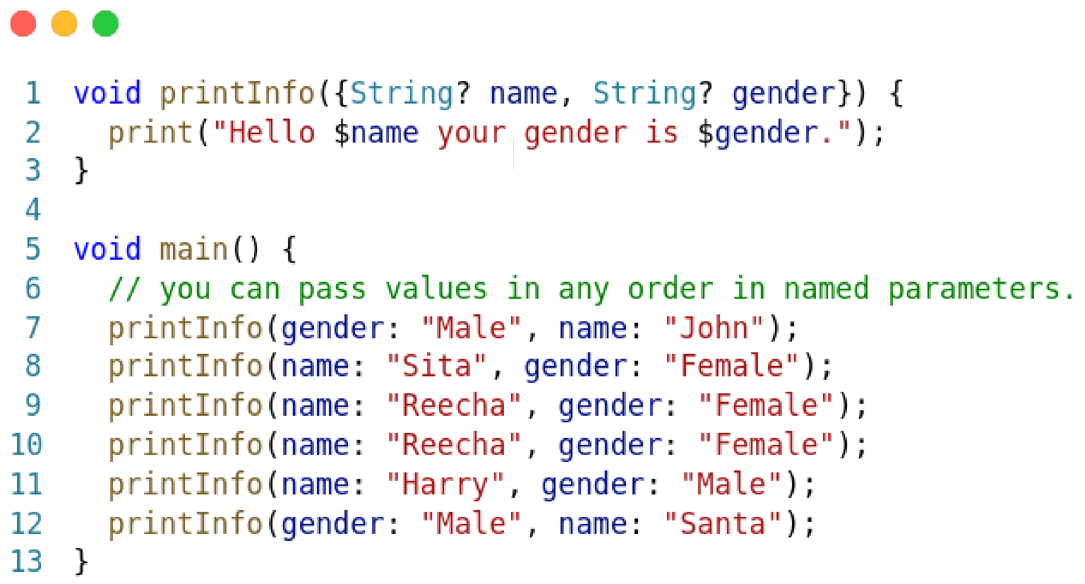
Result:

```
Hello sir/ma'am John your gender is Male.
Hello Mr. John your gender is Male.
Hello Ms. Kavya your gender is Female.
```

In the example above, function **printInfo** takes two positional parameters and one optional parameter. The title parameter is optional here. If the user doesn't pass the title, it will automatically set the title value to sir/ma'am.

Named Parameter

Dart allows you to use named parameters to clarify the parameter's meaning in function calls. Curly braces {} are used to specify named parameters.



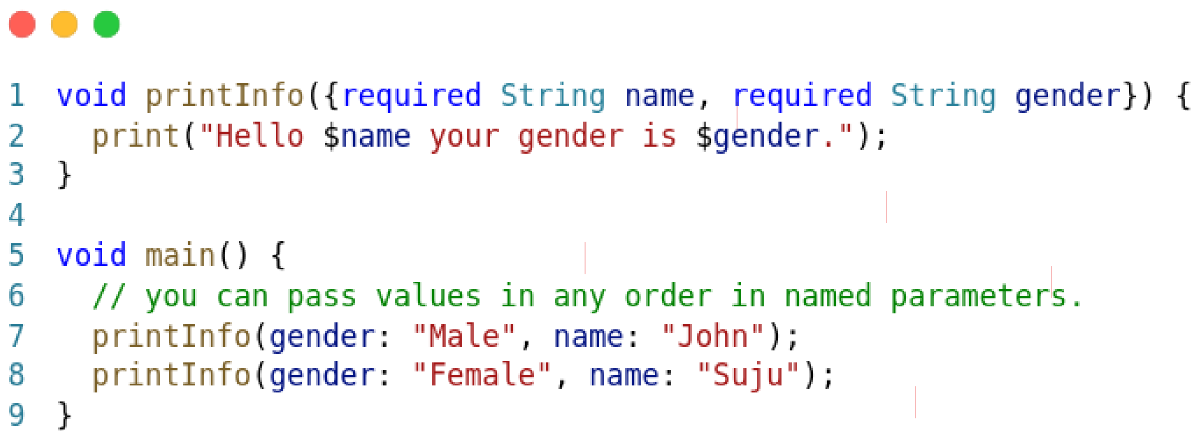
```dart
void printInfo({String? name, String? gender}) {
  print("Hello $name your gender is $gender.");
}

void main() {
  // you can pass values in any order in named parameters.
  printInfo(gender: "Male", name: "John");
  printInfo(name: "Sita", gender: "Female");
  printInfo(name: "Reecha", gender: "Female");
  printInfo(name: "Reecha", gender: "Female");
  printInfo(name: "Harry", gender: "Male");
  printInfo(gender: "Male", name: "Santa");
}
```

Result:

```
Hello John your gender is Male.
Hello Sita your gender is Female.
Hello Reecha your gender is Female.
Hello Reecha your gender is Female.
Hello Harry your gender is Male.
Hello Santa your gender is Male.
```

Use Of Required In Named Parameter

In the example below, function printInfo takes two named parameters. You can see a required keyword, which means you must pass the person's name and gender. If you don't pass it, it won't work.



```dart
void printInfo({required String name, required String gender}) {
  print("Hello $name your gender is $gender.");
}

void main() {
  // you can pass values in any order in named parameters.
  printInfo(gender: "Male", name: "John");
  printInfo(gender: "Female", name: "Suju");
}
```

Result:

```
Hello John your gender is Male.
Hello Suju your gender is Female.
```

Arrow Function

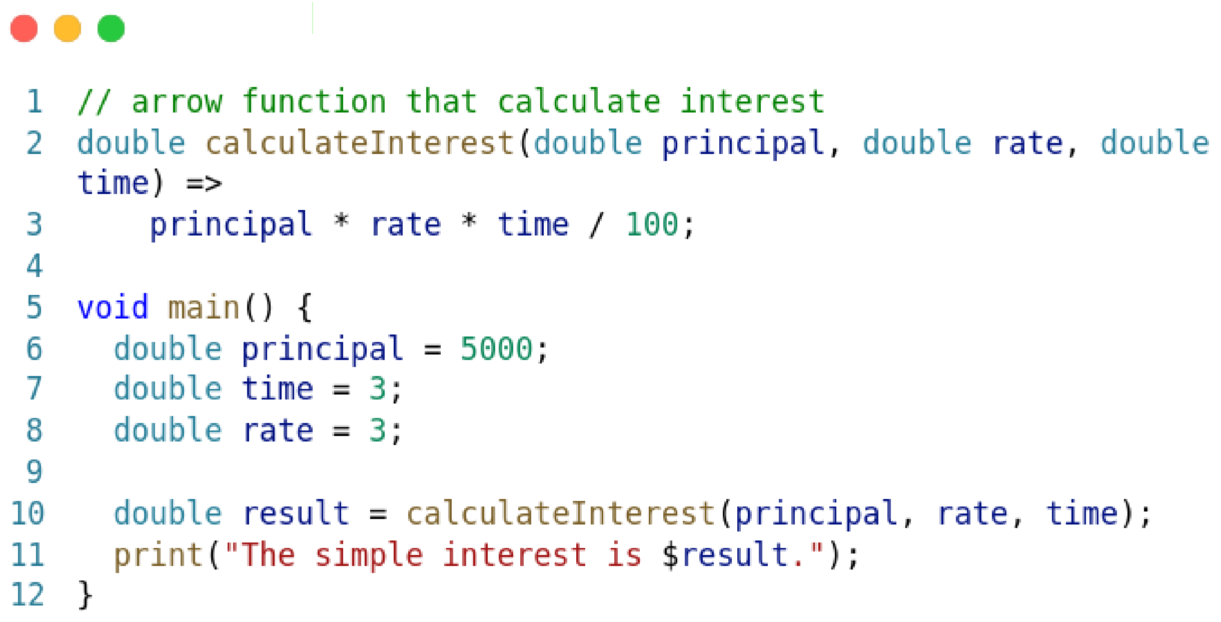
Dart has a special syntax for the function body, which is only one line. The arrow function is represented by => symbol. It is a shorthand syntax for any function that has only one expression.

Syntax

The syntax for the dart arrow function.

returnType functionName(parameters...) => expression;

Example:



```dart
// arrow function that calculate interest
double calculateInterest(double principal, double rate, double time) =>
    principal * rate * time / 100;

void main() {
  double principal = 5000;
  double time = 3;
  double rate = 3;

  double result = calculateInterest(principal, rate, time);
  print("The simple interest is $result.");
}
```

Result:

```
Simple interest is 450.0
```

1.8 OOP in Dart

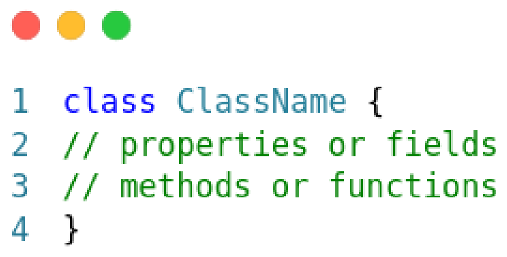
Object-oriented programming (OOP) is a programming method that uses objects and their interactions to design and program applications. It is one of the most popular programming paradigms and is used in many programming languages, such as Dart, Java, C++, Python, etc.

Advantages

- It is easy to understand and use.
- It increases reusability and decreases complexity.
- The productivity of programmers increases.
- It makes the code easier to maintain, modify and debug.
- It promotes teamwork and collaboration.
- It reduces the repetition of code.

Declaring Class In Dart

You can declare a class in dart using the class keyword followed by class name and braces {}. It's a good habit to write class name in PascalCase. For example, Employee, Student, QuizBrain, etc.

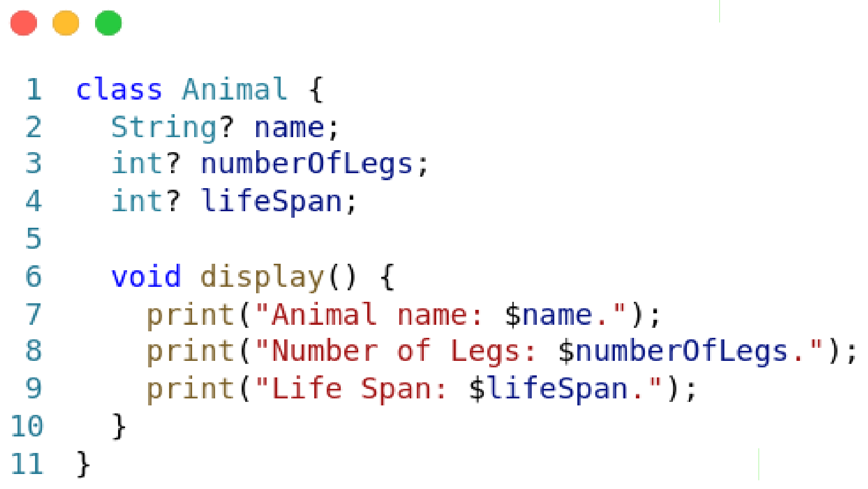


```
class ClassName {
// properties or fields
// methods or functions
}
```

In the above syntax:

- The class keyword is used for defining the class.
- ClassName is the name of the class and must start with capital letter.
- Body of the class consists of properties and functions.
- Properties are used to store the data. It is also known as fields or attributes.
- Functions are used to perform the operations. It is also known as methods.

Declaring A Class In Dart



```dart
class Animal {
  String? name;
  int? numberOfLegs;
  int? lifeSpan;

  void display() {
    print("Animal name: $name.");
    print("Number of Legs: $numberOfLegs.");
    print("Life Span: $lifeSpan.");
  }
}
```

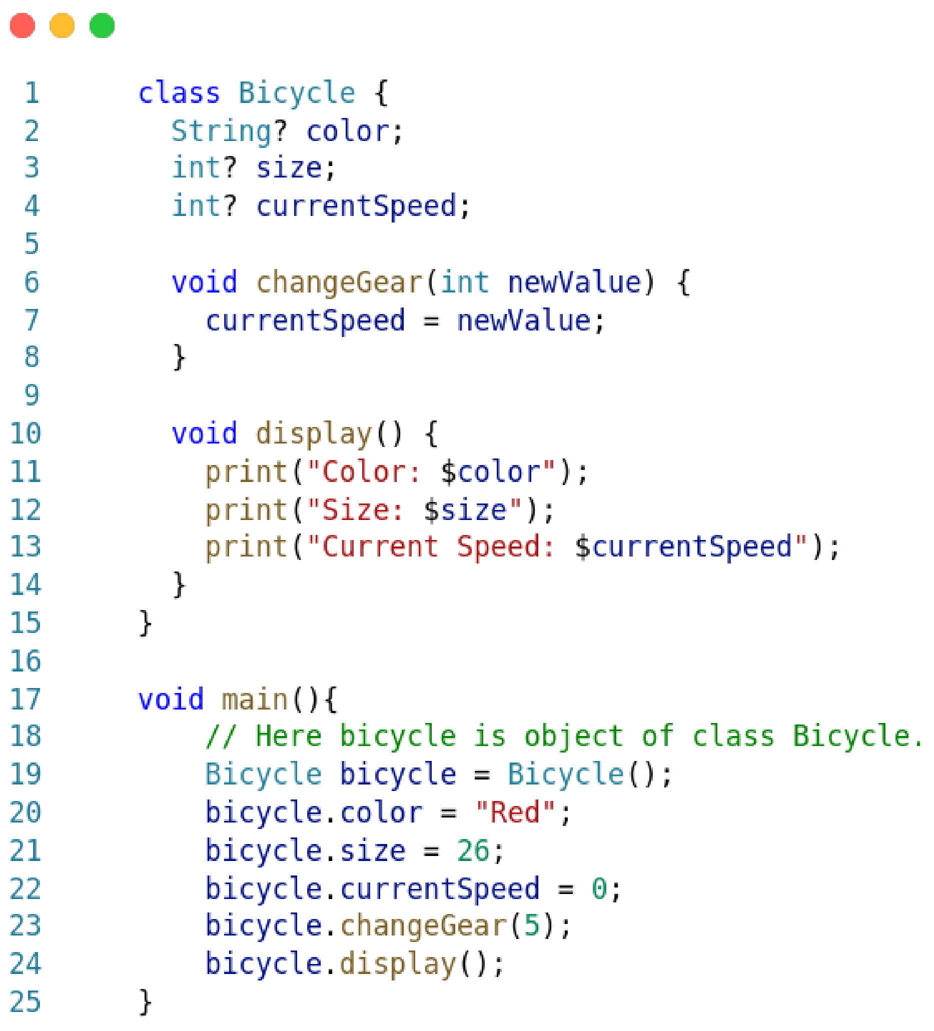
Object In Dart

In object-oriented programming, an object is a self-contained unit of code and data. Objects are created from templates called classes. An object is made up of properties(variables) and methods(functions). An object is an instance of a class.

Once you have created a class, it's time to declare the object. You can declare an object by the following syntax:

```dart
ClassName objectName = ClassName();
```

Example:



```dart
class Bicycle {
  String? color;
  int? size;
  int? currentSpeed;

  void changeGear(int newValue) {
    currentSpeed = newValue;
  }

  void display() {
    print("Color: $color");
    print("Size: $size");
    print("Current Speed: $currentSpeed");
  }
}

void main(){
    // Here bicycle is object of class Bicycle.
    Bicycle bicycle = Bicycle();
    bicycle.color = "Red";
    bicycle.size = 26;
    bicycle.currentSpeed = 0;
    bicycle.changeGear(5);
    bicycle.display();
}
```

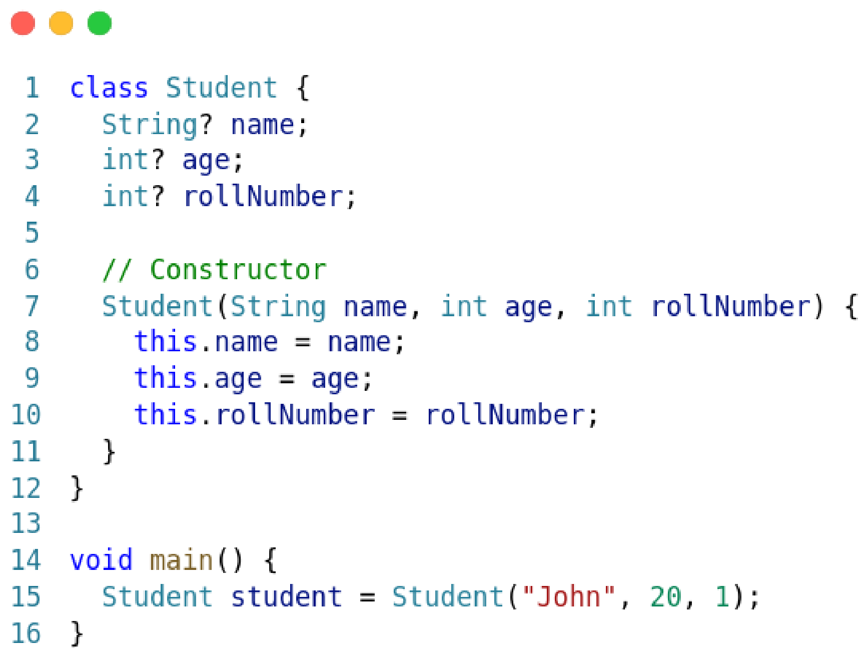
Result:

```
Result:

Color: Red
Size: 26
Current Speed: 5
```

Constructor In Dart

A constructor is a special method used to initialize an object. It is called automatically when an object is created, and it can be used to set the initial values for the object's properties.

Declare Constructor In Dart



```dart
class Student {
  String? name;
  int? age;
  int? rollNumber;

  // Constructor
  Student(String name, int age, int rollNumber) {
    this.name = name;
    this.age = age;
    this.rollNumber = rollNumber;
  }
}

void main() {
  Student student = Student("John", 20, 1);
}
```

Note: The **this** keyword is used to refer to the current instance of the class. It is used to access the current class properties. In the example above, parameter names and class properties of constructor Student are the same. Hence to avoid confusion, we use the **this** keyword.

Write Constructor Single Line

In the above section, you have written the constructor in long form. You can also write the constructor in short form. You can directly assign the values to the properties. For example, the following code is the short form of the constructor in one line.

```dart
class Person{
  String? name;
  int? age;
  String? subject;
  double? salary;

  // Constructor in short form
  Person(this.name, this.age, this.subject, this.salary);
}

void main(){
  Person person = Person("John", 30, "Maths", 50000.0);
}
```

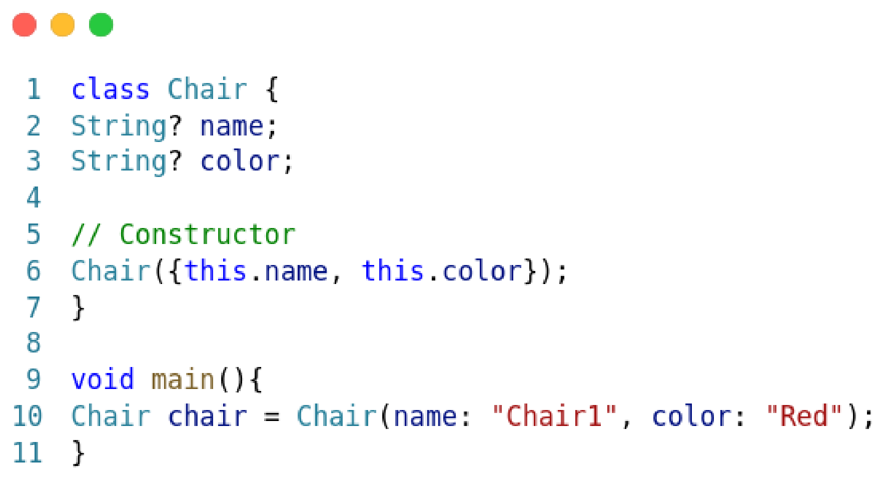
Constructor With Optional Parameters

In the example below, we have created a class Employee with four properties: name, age, subject, and salary. Class has one constructor for initializing all properties values. For subject and salary, we have used optional parameters.

```dart
class Employee {
  String? name;
  int? age;
  String? subject;
  double? salary;

  // Constructor
  Employee(this.name, this.age, [this.subject = "N/A", this.salary=0]);
}

void main(){
  Employee employee = Employee("John", 30);
}
```

Constructor With Named Parameters

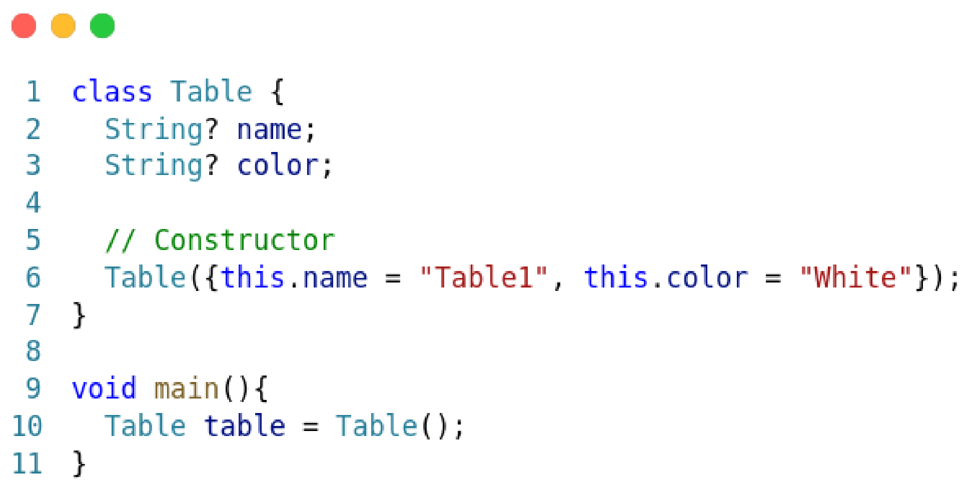
In the example below, we have created a class Chair with two properties: name and color. Class has one constructor for initializing the all properties values with named parameters.



```dart
class Chair {
String? name;
String? color;

// Constructor
Chair({this.name, this.color});
}

void main(){
Chair chair = Chair(name: "Chair1", color: "Red");
}
```

Constructor With Default Values

In the example below, we have created a class Table with two properties: name and color. Class has one constructor for initializing all properties values with default values.



```dart
class Table {
  String? name;
  String? color;

  // Constructor
  Table({this.name = "Table1", this.color = "White"});
}

void main(){
  Table table = Table();
}
```

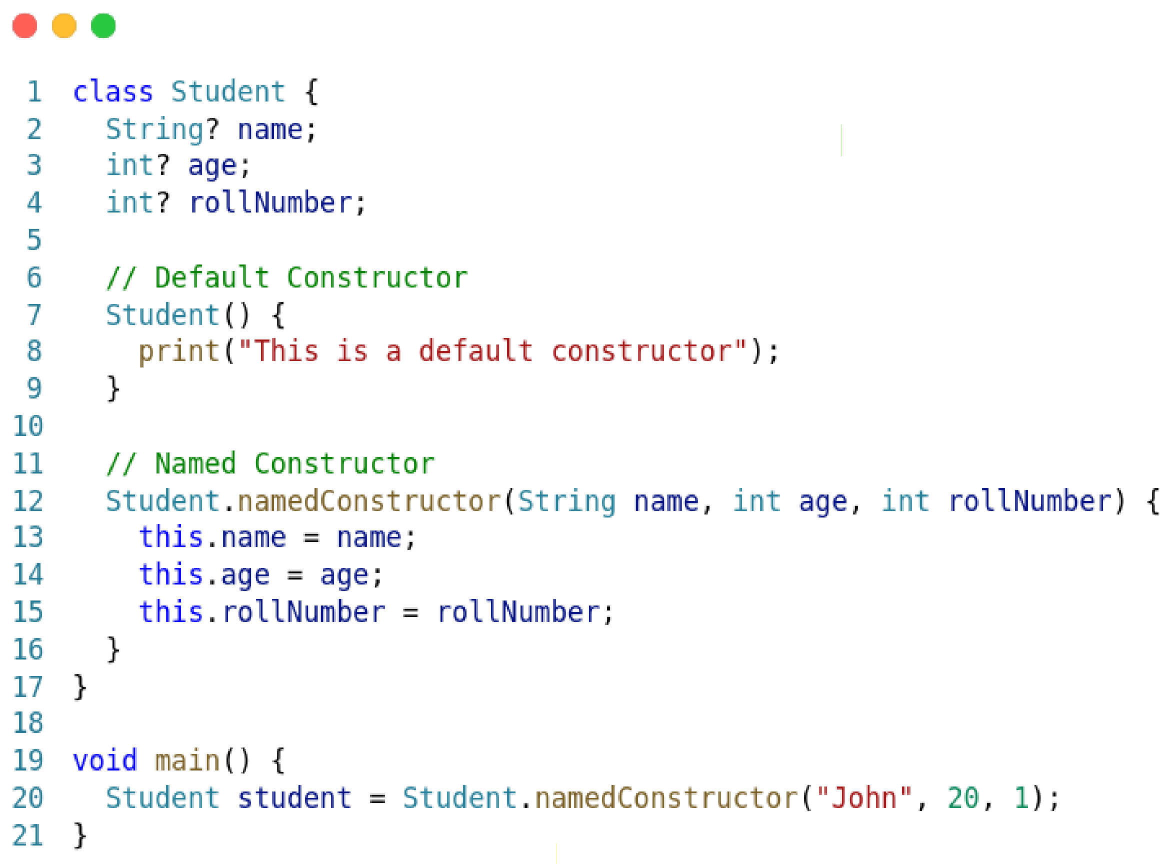
Named Constructor In Dart

In most programming languages like java, c++, c#, etc., we can create multiple constructors with the same name. But in Dart, this is not possible. Well, there is a way. We can create multiple constructors with the same name using named constructors.

Note: Named constructors improves code readability. It is useful when you want to create multiple constructors with the same name.

Example:

In this example below, there is a class Student with three properties: name, age, and rollNumber. The class has two constructors. The first constructor is a default constructor. The second constructor is a named constructor. The named constructor is used to initialize the values of the three properties. We also have an object of the class Student called student.



```dart
class Student {
  String? name;
  int? age;
  int? rollNumber;

  // Default Constructor
  Student() {
    print("This is a default constructor");
  }

  // Named Constructor
  Student.namedConstructor(String name, int age, int rollNumber) {
    this.name = name;
    this.age = age;
    this.rollNumber = rollNumber;
  }
}

void main() {
  Student student = Student.namedConstructor("John", 20, 1);
}
```

Constant Constructor In Dart

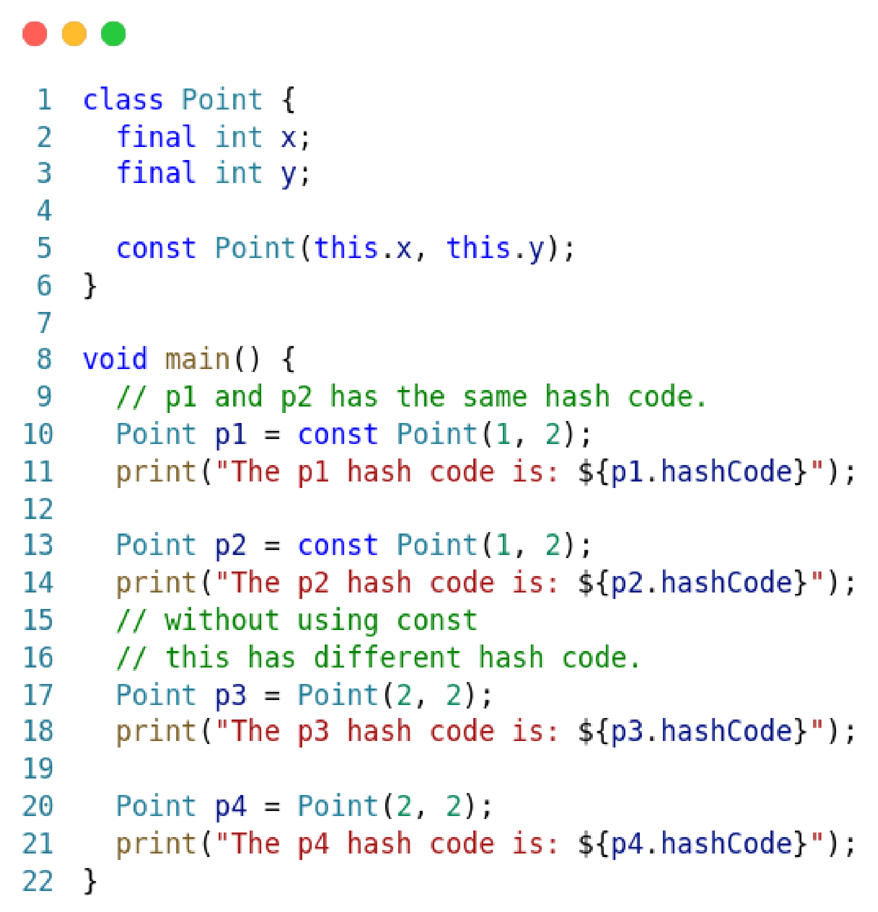
Constant constructor is a constructor that creates a constant object. A constant object is an object whose value cannot be changed. A constant constructor is declared using the keyword const.

Note: Constant Constructor is used to create an object whose value cannot be changed. It Improves the performance of the program.

Rule For Declaring Constant Constructor In Dart

- All properties of the class must be final.
- It does not have any body.
- Only class containing const constructor is initialized using the const keyword.

Example:



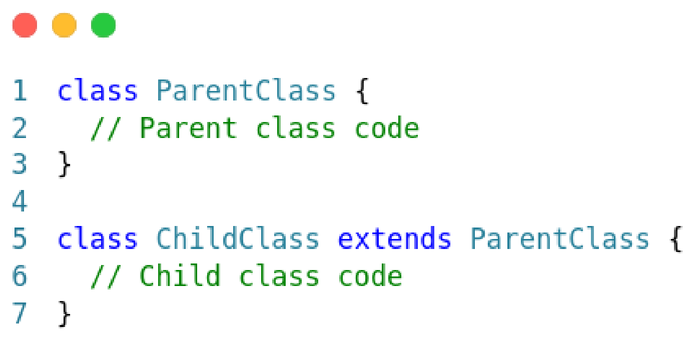
```dart
class Point {
  final int x;
  final int y;

  const Point(this.x, this.y);
}

void main() {
  // p1 and p2 has the same hash code.
  Point p1 = const Point(1, 2);
  print("The p1 hash code is: ${p1.hashCode}");

  Point p2 = const Point(1, 2);
  print("The p2 hash code is: ${p2.hashCode}");
  // without using const
  // this has different hash code.
  Point p3 = Point(2, 2);
  print("The p3 hash code is: ${p3.hashCode}");

  Point p4 = Point(2, 2);
  print("The p4 hash code is: ${p4.hashCode}");
}
```

Note: Here p1 and p2 has the same hash code. This is because p1 and p2 are constant objects. The hash code of a constant object is the same. This is because the hash code of a constant object is computed at compile time. The hash code of a non-constant object is computed at run time. This is why p3 and p4 have different hash code.

Inheritance In Dart

Inheritance is a sharing of behavior between two classes. It allows you to define a class that extends the functionality of another class. The extend keyword is used for inheriting from parent class.

Note: Whenever you use inheritance, it always create a is-a relation between the parent and child class like Student is a Person, Truck is a Vehicle, Cow is a Animal etc.

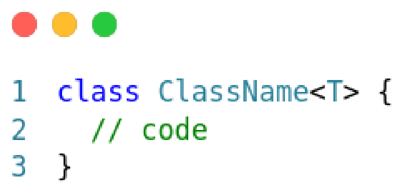


```dart
class ParentClass {
  // Parent class code
}

class ChildClass extends ParentClass {
  // Child class code
}
```

1.9. Generics in Dart

Generics is a way to create a class, or function that can work with different types of data (objects). If you look at the internal implementation of List class, it is a generic class. It can work with different data types like int, String, double, etc. For example, List<int> is a list of integers, List<String> is a list of strings, and List<double> is a list of double values.

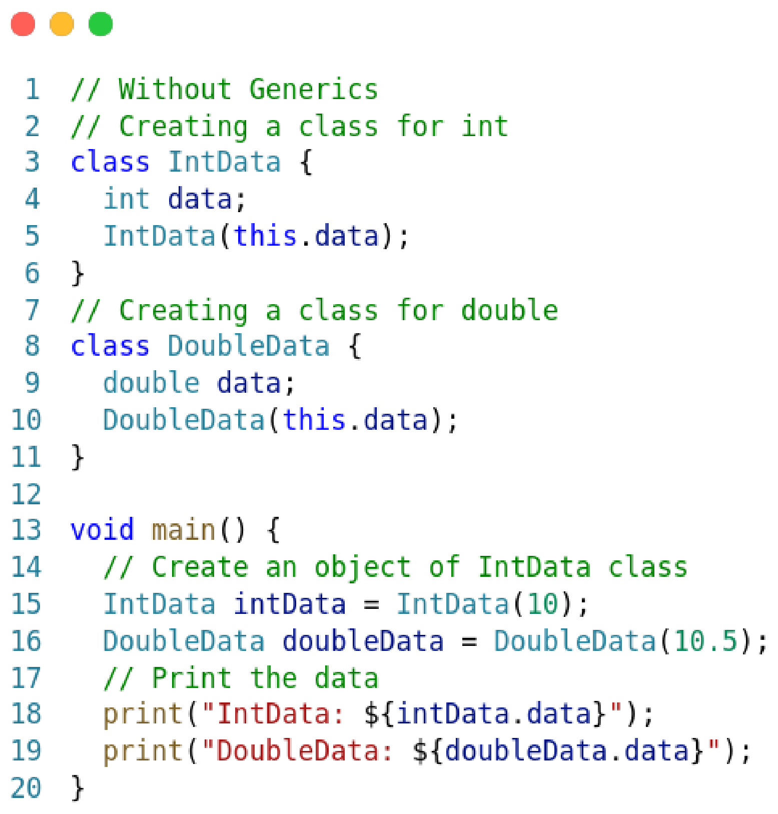
Syntax



```dart
class ClassName<T> {
  // code
}
```

Example: without generics

Suppose, you need to create a class that can work with both int and double data types. You can create two classes, one for int and another for double like this:



```dart
// Without Generics
// Creating a class for int
class IntData {
  int data;
  IntData(this.data);
}
// Creating a class for double
class DoubleData {
  double data;
  DoubleData(this.data);
}

void main() {
  // Create an object of IntData class
  IntData intData = IntData(10);
  DoubleData doubleData = DoubleData(10.5);
  // Print the data
  print("IntData: ${intData.data}");
  print("DoubleData: ${doubleData.data}");
}
```

This is not a good practice because both classes contain the same code. You can create one Generics class that can work with different data types. See the example below.

Example: with generics

In this example below, there is a single class that can work with int, double, and any other data types using Generics.

```dart
// Using Generics
class Data<T> {
  T data;
  Data(this.data);
}

void main() {
  // create an object of type int and double
  Data<int> intData = Data<int>(10);
  Data<double> doubleData = Data<double>(10.5);

  // print the data
  print("IntData: ${intData.data}");
  print("DoubleData: ${doubleData.data}");
}
```

Generics Methods

You can also create a generic method. For this, you need to use the <T> keyword before the method's return type. See the example below.

```dart
// Define generic method
T genericMethod<T>(T value) {
  return value;
}

void main() {
  // call the generic method
  print("Int: ${genericMethod<int>(10)}");
  print("Double: ${genericMethod<double>(10.5)}");
  print("String: ${genericMethod<String>("Hello")}");
}
```

Generic Method With Multiple Parameters

In this example below, you will learn to create a generic method with multiple parameters.

```dart
// Define generic method
T genericMethod<T, U>(T value1, U value2) {
  return value1;
}

void main() {
  // call the generic method
  print(genericMethod<int, String>(10, "Hello"));
  print(genericMethod<String, int>("Hello", 10));
}
```

Restricting the Type of Data

While implementing generics, you can restrict the type of data that can be used with the class or method. This is done by using the extends keyword. See the example below.

In this example below, there is a Data class that works only with int and double types. It will not work with other types..

```dart
// Define generic class with bounded type
class Data<T extends num> {
  T data;
  Data(this.data);
}

void main() {
  // create an object of type int and double
  Data<int> intData = Data<int>(10);
  Data<double> doubleData = Data<double>(10.5);
  // print the data
  print("IntData: ${intData.data}");
  print("DoubleData: ${doubleData.data}");
  // Not Possible
  // Data<String> stringData = Data<String>("Hello");
}
```

1.10. Dart Collection

Reading assignment

1.11. Dart Exception

Reading assignment

1.12. Dart Futures

Reading assignment