SILVER OAK UNIVERSITY

EDUCATION TO INNOVATION

Chapter 2: Dart Programming

Unit-name : Dart Programming

Topics : Introduction to Dart Programming
Characteristics of Dart .
Variables and Data types
Decision Making and Loops.
Functions
Dart Object-Oriented Concept

Introduction to Dart Programming

- Dart is a general-purpose, high-level modern programming language which is originally developed by Google. It is the new programming language which is emerged in 2011, but its stable version was released in June 2017. Dart is not so popular at that time, but It gains popularity when it is used by the Flutter.
- Dart is a dynamic, class-based, object-oriented programming language. Syntactically, it is quite similar to Java, C, and JavaScript. If you know any of these programming languages, you can easily learn the Dart programming language.
- It also supports a few advance concepts such as interfaces, mixins, abstract classes, refield generics, and type interface.
- Dart is an open-source programming language which is widely used to develop the mobile application, modern web-applications, desktop application, and the Internet of Things (IoT) using by Flutter framework.
- It is a compiled language and supports two types of compilation techniques.
- **AOT (Ahead of Time) -** It converts the Dart code in the optimized JavaScript code with the help of the dar2js compiler and runs on all modern web-browser. It compiles the code at build time.
- **JOT (Just-In-Time) -** It converts the in the machine code (native code), code that is necessary.byte code but only

Characteristics of Dart

- Dart is a platform-independent language and supports all operating systems such as Windows, Mac, Linux, etc.
- It is an open-source language, which means it available free for everyone. It comes with a BSD license and recognized by the ECMA standard.
- It is an object-oriented programming language and supports all features of oops such as inheritance, interfaces, and optional type features.
- Dart is very useful in building real-time applications because of its stability.
- Dart comes with the dar2js compiler which transmits the Dart code into JavaScript code that runs on all modern web browser.
- The stand-alone Dart VM permits Dart code to run in a command-line interface environment.

Variables and Data types

- The variables are used to store values and reserve the memory location.
- In Dart, the data type of the variable is defined by its value.
- The data-type specifies what type of value will be stored by the variable. Each variable has its data-type. The Dart is a static type of language, which means that the variables cannot modify.
- Dart supports the following built-in Data types. • Number
  - Strings
  - Boolean
  - Lists
  - Maps
  - Symbols

- **Dart Number**
- The Darts Number is used to store the numeric values. The number can be two types - integer and double.
- **Integer -** Integer values represent the whole number or non-fractional values. An integer data type represents the 64-bit non-decimal numbers between -2^{63} to 2^{63}. A variable can store an unsigned or signed integer value. The example is given below
- int marks = 80;
- **Double -** Double value represents the 64-bit of information (double-precision) for floating number or number with the large decimal points. The double keyword is used to declare the double type variable.
- double pi = 3.14;

- **Dart Strings**
- A string is the sequence of the character. If we store the data like - name, address, special character, etc. It is signified by using either single quotes or double quotes. A Dart string is a sequence of UTF-16 code units.
- var msg = "Welcome to JavaTpoint";

- **Dart Boolean :** The Boolean type represents the two values - true and false. The bool keyword uses to denote Boolean Type. The numeric values 1 and 0 cannot be used to represent the true or false value.

  bool isValid = true;

- **Dart Lists**
- In Dart, The list is a collection of the ordered objects (value). The concept of list is similar to an array. An array is defined as a collection of the multiple elements in a single variable. The elements in the list are separated by the comma enclosed in the square bracket[]. The sample list is given below.

- var list = [1,2,3]
- **Dart Maps**
- The maps type is used to store values in key-value pairs. Each key is associated with its value. The key and value can be any type. In Map, the key must be unique, but a value can occur multiple times. The Map is defined by using curly braces ({}), and comma separates each pair.
- var student = {'name': 'Joseph', 'age':25, 'Bran ch': 'Computer Science'}
- **Dart Runes:** As we know that, the strings are the sequence of Unicode UTF-16 code units. Unicode is a technique which is used to describe a unique numeric value for each digit, letter, and symbol. Since Dart Runes are the special string of Unicode UTF-32 units. It is used to represent the special syntax.
- For example - The special heart character ♥ is equivalent to Unicode code \u2665, where \u means Unicode, and the numbers are hexadecimal integer. If the hex value is less or greater than 4 digits, it places in a curly bracket ({}). For example - An emoji 怀 is represented as \u{1f600}. The example is given below.

```
void main(){
        var heart_symbol = '\u2665';
        var laugh_symbol = '\u{1f600}';
        print(heart_symbol);
        print(laugh_symbol);   }
```

o/p • ♥

Dart Symbols

- The Dart Symbols are the objects which are used to refer an operator or identifier that declare in a Dart program. It is commonly used in APIs that refers to identifiers by name because an identifier name can changes but not identifier symbols.

- **Dart Dynamic Type**
- Dart is an optionally typed language. If the variable type is not specified explicitly, then the variable type is dynamic. The dynamic keyword is used for type annotation explicitly.

Dart Variable

- Variable is used to store the value and refer the memory location in computer memory. When we create a variable, the Dart compiler allocates some space in memory. The size of the memory block of memory is depended upon the type of variable. To create a variable, we should follow certain rules. Here is an example of a creating variable and assigning value to it.

- var name = 'Devansh';
- **Rule to Create Variable**
- The variable cannot contain special characters such as whitespace, mathematical symbol, runes, Unicode character,

  and keywords.

- The first character of the variable should be an alphabet([A to Z],[a to z]). Digits are not allowed as the first character.
- Variables are case sensitive. For example, - variable age and AGE are treated differently.
- The special character such as #, @, ^, &, * are not allowed expect the underscore(_) and the dollar sign($).

- The variable name should be retable to the program and readable.

- **How to Declare Variable in Dart**
- We need to declare a variable before using it in a program. In Dart, The **var** keyword is used to declare a variable. The Dart compiler automatically knows the type of data based on the assigned to the variable because Dart is an infer type language. The syntax is given below.

  Syntax - var <variable_name> = <value>; • or
  var <variable_name>;
- **Example -** var name = 'Andrew' • **Type Annotations**
- While declaring the variable, it suggests the type of the value that variable can store. In the type annotation, we add the data type as a prefix before the variable's name that ensures that the variable can store specific data type. The syntax is given below.

Syntax -
<type> <variable_name>; • or
<type> <name> = <expression>; • **Example -** int age;
String msg = "Welcome to JavaTpoint";

- Dart provides the facility to declare multiple values of the same type to the variables. We can do this in a single statement, and

  each value is separated by commas.
- **Example –**int i,j,k;
- **Default Value:** While declaring the variable without initializing the value then the Dart compiler provides default value (Null) to the variable.

int count;
assert(count == null);
- **Final and const**

- When we do not want to change a variable in the future then we use final and const. It can be used in place of **var** or in addition to a type. A final variable can be set only one time where the variable is a compile-time constant. The example of creating a final variable is given below.

- **Example -**

final name = 'Ricky'; // final variable without ty pe annotation.

final String msg = 'How are you?'; // final variable with type annotation.

we try to change these values then it will throw an error.

name = 'Roger'; // Error: Final v ariable can't be changed.

- The **const** is used to create compile-time constants. We can declare a value to compile-time constant such as number, string literal, a const variable, etc.

const a = 1000;
The const keyword is also used to create a constant value that cannot be changed after its creation.

var f = const[];

- If we try to change it, then it will throw an error.

f = [12]; //Error, The const variable cannot b e change

Decision Making and Loops.

- In Dart, Control flow statement can be categorized mainly in three following ways.

  -Decision-making statements -Looping statements

  -Jump statements
- Dart Decision-Making Statements
- Decision-making statements are also known as the Selection statements.
- In Dart program, single or multiple test expression (or condition) can be existed, which evaluates Boolean TRUE and FALSE. These results of the expression/condition helps to decide which block of statement (s) will execute if the given condition is TRUE or FALSE.
- Dart provides following types of Decision-making statement.
- If Statement
- If-else Statements • If else if Statement
- Switch Case Statement
- Dart if Statements: If statement allows us to a block of code execute when the given condition returns true. In Dart programming, we have a scenario where we want to execute a block of code when it satisfies the given condition. The condition evaluates Boolean values TRUE or FALSE and the decision is made based on these Boolean values.
- **Syntax -** If (condition) {

  //statement(s)

  }

- Example-

```
void main () {
  // define a variable which holds a numeric value
  var age = 16;

  // if statement to check the given condition
  if (age > 18) {
    print("You are eligible for voting");
  };

  print("You are not eligible for voting");
}
```

//o/p You are not eligible for voting

- **Syntax-if**(condition) {
  // statement(s);}
  else {
  // statement(s);}
- Example:

```
 void main() {
var num = 20;
print("if-else statement example");
if(num%2 == 0){
print("The given number is even"); }
   else {
print("The given number is odd"); };
```

// o/p I f-else statement example The given // //number is even

- Dart if else-if Statement

Dart if else-if statement provides the facility to check a set of test expressions and execute the different statements. It is used when we have to make a decision from more than two possibilities.

Decision Making and Loops.

- Syntax

if (condition1) { //statement(s)}
else if(condition2) {
// statement(s)}
else if (conditionN) { // statement(s)
}
else {
// statement(s) }

```
void main() {
  var marks = 74;

  if (marks > 85) {
    print("Excellent");
  }
  else if (marks > 75) {
    print("Very Good");
  }
  else if (marks > 65) {
    print("Good");
  }
  else {
    print("Average");
  }
}
```

- Nested If else Statement
  nested if else statement means one if-else inside another one.

```
void main() {
  var a = 10;
  var b = 20;
  var c = 30;

  if (a > b) {
    if (a > c) {
      print("a is greater");
    } else {
      print("c is greater");
    }
  } else if (b > c) {
    print("b is greater");
  } else {
    print("c is greater");
  }
}
```

- **Dart Switch Case Statement** :Dart Switch case statement is used to avoid the long chain of the if-else statement. It is the simplified form of nested if-else statement. The value of the variable compares with the multiple cases, and if a match is found, then it executes a block of statement associated with that particular case.

Syntax:

switch(expression) {
  case value-1:{
  //statement(s) Block-1;}
  break;

case value-2:{
//statement(s) Block-2;}
break;
case value-N:{
//statement(s) Block-N;}
break;
default: {
//statement(s);}}

```
void main() {
  // Declaring an integer variable
  int Roll_num = 90014;

  // Evaluate the test-expression to find the match
  switch (Roll_num) {
    case 90009:
      print("My name is Joseph");
      break;
    case 90010:
      print("My name is Peter");
      break;
    case 90011: // The correct format should be without the leading zero.
      print("My name is Devansh");
      break;
    default:
      print("Roll number is not found");
  }
```

- Dart Loops
- Dart Loop is used to run a block of code repetitively for a given number of times or u
- until matches the specified condition. Dart supports the following type of loops.
- Dart for loop
- Dart for…in loop • Dart while loop
- Dart do-while loop
- Dart for loop
- The for loop is used when we know how many times a block of code will execute.
- Syntex:

  for(Initialization; condition; incr/decr) {

  // loop body}

- Example

```
void main() {
  for(int i=1;i<=10;i++){
    print(i);
  }
}
```

o/p 1 2 3 4 5 6 7 8 9 10(in new line)

- Dart for… in Loop

  The for…in loop is slightly different from the for loop. It only takes dart object or expression as an iterator and iterates the element one at a time. The value of the element is bound to var, which is and valid and available for the loop body. The loop will execute until no element left in the iterator.

- **Syntax**

  for (var in expression) { //statement(s)}

Example:

```
void main() {
  var list1 = [10,20,30,40,50];
for(var i in list1){
    print(i);
  }
}
```

o/p 10 20 30 40 50

- Dart while loop

  The while loop executes a block of code until the given expression is false

Syntax:

while(condition) {
// loop body}

Example:

```
void main() {
  var a = 1;
  var maxnum = 10;
  while(a<maxnum){  // it will print until the expres //sion return false
  a = a+1;  // increase value 1 aft //er each iteration
  print(a);
}
}
```

O/P 1 2 3 4 5 6 7 8 9(IN NEW LINE

- Dart do…while Loop:

  The do…while loop is similar to the while loop but only difference is that, it executes the loop statement and then check the given condition.

Syntax: **do** {

// loop body

} **while**(condition);

```
void main() {
  var a = 1;
  var maxnum = 10; do
{
     print("The value is: ${a}");
     a = a+1;}
     while(a<maxnum);
}
```

- o/p The value is: 1 The value is: 2 The value is: 3 The value is: 4 The value is: 5 The value is: 6 The value is: 7 The value is: 8 The value is: 9

- prime or not

```
void main(){
  int i, flag = 0;
  int num = 5;
  for(i = 2; i < num; i++) {
    if(num % i == 0) {
      print('$num is not a prime number');
```

```
      flag = 1;
      break;
    }
  }
  if(flag == 0) {
    print('$num is prime number');
  }
}
```

Functions

- Dart function is a set of codes that together perform a specific task. It is used to break the large code into smaller modules and reuse it when needed. Functions make the program more readable and easy to debug. It improves the modular approach and enhances the code reusability.
- The function provides the flexibility to run a code several times with different values. A function can be called anytime as its parameter and returns some value to where it called.
- Advantages of Functions
  It increases the module approach to solve the problems.
  It enhances the re-usability of the program. We can do the coupling of the programs.
  It optimizes the code.
  It makes debugging easier.
  It makes development easy and creates less complexity.

- Defining a Function

  A function can be defined by providing the name of the function with the appropriate parameter and return type. A function

  contains a set of statements which are called function body.

  Syntax return_type func_name (parameter_list): {
  //statement(s) **return** value;
  }

- **return_type -** It can be any data type such as void, integer, float, etc. The return type must be matched with the returned value of the function.
- **func_name -** It should be an appropriate and valid identifier.
- **parameter_list -** It denotes the list of the parameters, which is necessary when we called a function.
- **return value -** A function returns a value after complete its execution.

Calling a Function

```
void main() {
    print('Mul.. is${mul(12,45)}');
}

int mul(int a, int b){
return a*b;
}
```

After creating a function, we can call or invoke the defined function inside the main() function body. A function is invoked simply by its name with a parameter list, if any.

- **Syntax: f**un_name(<argument_list>); or
  variable = function_name(argument); mul(10,20);

- Passing Arguments to Function and Return a Value from Function :

When a function is called, it may have some information as per the function prototype is known as a parameter (argument).

A function always returns some value as a result to the point where it is called. The **return** keyword is used to return a value. The return statement is optional. A function can have only one return statement

- Dart Function with parameter and return value

```
void main() {
  double radius;
  // Prompt the user to enter the radius
  stdout.write("Enter the radius of Circle: ");
  radius = double.parse(stdin.readLineSync()!);

  // Check if the radius is valid
  if (radius <= 0.0) {
    print("Error: Please enter a valid value");
    return;
  }
// Print the result
  print("Area of Circle with radius $radius is: ${getArea(radius)}");
}
```

- Dart Anonymous Function

  We have learned the Dart Function, which is defined by using a user-define name. Dart also provides the facility to specify a nameless function or function without a name. This type of function is known as an **anonymous function, lambda, or closure**.

  An anonymous function behaves the same as a regular function, but it does not have a name with it. It can have zero or any number of arguments with an optional type annotation. We can assign the anonymous function to a variable, and then we can retrieve or access the value of the closure based on our requirement.

  An Anonymous function contains an independent block of the code, and that can be passed around in our code as function parameters. The syntax is as follows.

- Syntax : (parameter_list) {
  statement(s) }

```
void main() {
var list = ["James","Patrick","Mathew","Tom"];
print("Example of anonymous function");
list.forEach((item) {
print('${list.indexOf(item)}: $item');
 });
}
```

- output

Example of anonymous function

0 : James

1: Patrick

2: Mathew

3: Tom

- The main() function:
- The main() function is the top-level function of the Dart. It is the most important and vital function of the Dart programming language. The execution of the programming starts with the **main()** function. The **main()** function can be used only once in a program.that's also known as entry point function.
- The main function returns void and can have an optional List<String> parameter as arguments.

Syntax

void main() {
    // main function body }

Example

```
void main() {
  print("Welcome To Dart programming");
  }
```

Dart Object-Oriented Concepts

- Dart is an object-oriented programming language, and it supports all the concepts of object-oriented programming such as classes, object, inheritance and abstract classes.
- It focuses on the object and objects are the real-life entities. The Object-oriented programming approach is used to implement the concept like polymorphism, data-hiding, etc.
- The main goal of oops is to reduce programming complexity and do several tasks simultaneously. The oops concepts are given below.

- Class
- Object
- Inheritance
- Polymorphism • Interfaces
- Abstract class

- **Class**

  Dart classes are defined as the blueprint of the associated objects. A Class is a user-defined data type that describes the characteristics and behavior of it. To get all properties of the class, we must create an object of that class.

Syntax

```
class ClassName { <fields>
    <getter/setter>
    <constructor> <functions>
}
```

- **Object**: An object is a real-life entity such as a table, human, car, etc
- We can access the class properties by creating an object of that class. In Dart, The object can be created by using a new keyword followed by class name.

Example

```
class Student {
  // Fields
  var id;
  var name;
  var ram;

  // Constructor (optional)
  Student(this.id, this.name, this.ram); // parameterise

  // Class Function
  void showStdInfo() {
    print("Student ID is: $id");
    print("Student Name is: $name");
    print("Student RAM is: $ram GB");
  }
}

void main() {
  // Creating Object called 'std'
  var std = Student(001, "Dell", 32);

  // Accessing class Function
  std.showStdInfo();
}
```

- **What is constructor**?

A constructor is a different type of function which is created with same name as its class name. The constructor is used to ini tialize an object when it is created. When we create the object of class, then constructor is automatically called. It is quite sim ilar to class unction but it has no explicit return type.

- **Syntax:**

class ClassName {

ClassName() {}}

- **Types of Constructors**

- There are three types of constructors in Dart as given below.
- Default Constructor or no-arg Constructor
- Parameter Constructor
- Named Constructor

- **Dart this Keyword**

The this keyword is used to refer the current class object. It indicates the current instance of the class, methods, or constructor. It can be also used to call the current class methods or constructors.

```
// Mobile Class with Constructor
class Mobile {
  String modelname;
  int man_year;

  // Constructor
  Mobile(this.modelname, this.man_year);

  void showMobileInfo() {
    print("Mobile's model name is: $modelname, and the manufacture year is: $man_year");
  }
```

```
}

void main() {
  // Creating an object of the Mobile class
  Mobile mob = Mobile("iPhone 11", 2020);
  mob.showMobileInfo();
}

// Output:
// Mobile's model name is: iPhone 11, and the manufacture year is: 2020
```

- **Dart Object-Oriented Concepts**
- **Class Variable**: Class variable is also known as the **static member** variable, which is used to declare using the static keyword. It is declared in the class, but outside a constructor, method or a block. All instances share one copy of the class variable or we can say that class variables are common to all instances of that class.
- The static variable is also identified as a class variable.
- Single copy of the static variable is shared among the instance of a class.
- It can be accessed using the class name. We don't need to create an object of that class they belong to.
- The static variables can be accessed directly in the static methods.
- The static methods are the member class instead of its object.

```
void main() {
  Student std1 = Student(); // Creating instances of Student class
  Student std2 = Student();
  // Assigning value to static variable using class name
  Student.stdBranch = "Computer Science";
```

```
  std1.stdName = "Ben Cutting";
  std1.roll_num = 90013;
  std1.showStdInfo();

  std2.stdName = "Peter Handscomb";
  std2.roll_num = 90014;
  std2.showStdInfo();
}

// Output:
// Student's name is: Ben Cutting
// Student's roll number is: 90013
// Student's branch name is: Computer Science
//
// Student's name is: Peter Handscomb
// Student's roll number is: 90014
// Student's branch name is: Computer Science
```

- **Instance Variable :** Instance variable is also known as the non-static, variable which is used to declare without the static keyword. The instance variables are specific by an object. These variables can be accessed using the instance of that class.

- **Dart Inheritance**: Dart inheritance is defined as the process of deriving the properties and characteristics of another class. It provides the ability to create a new class from an existing class. It is the most essential concept of the oops(Object-Oriented programming approach). We can reuse the all the behavior and characteristics of the previous class in the new class.

- **Parent Class -** A class which is inherited by the other class is called **superclass** or **parent class**. It is also known as a **base class**.

- **Child Class -** A class which inherits properties from other class is called the child class. It is also known as the **derived class** or **subclass**.

- **Syntax class** child_class **extends** parent_class {
//body of child class
}

- Types of Inheritance 1.Single Inheritance
2. Multiple Inheritance 3. Multilevel Inheritance
4.Hierarchical Inheritance

- **Single Level Inheritance**: In the single inheritance, a class is inherited by a single class or subclass is inherited by one parent class.

```
class Bird {
  void fly() {
    print("The bird can fly");
  }
}

class Parrot extends Bird {
  void speak() {
    print("The parrot can speak");
  }
}

void main() {
  Parrot p = Parrot();
  p.speak();
```

```
  p.fly();
}

// Output:
// The parrot can speak
// The bird can fly
```

- **Multilevel Inheritance**: In the multiple inheritance, a subclass is inherited by another subclass or creates the chaining of inheritance.

```
class Bird {
  void fly() {
    print("The bird can fly");
  }
}

class Parrot extends Bird {
  void speak() {
    print("The parrot can speak");
  }
}

class Eagle extends Parrot {
  void vision() {
    print("The eagle has a sharp vision");
  }
```

```
}

void main() {
  Eagle e = Eagle();
  e.speak();
  e.fly();
  e.vision();
}

// Output:
// The parrot can speak
// The bird can fly
// The eagle has a sharp vision
```

- **Hierarchical Inheritance** :In the hierarchical inherence, two or more classes inherit a single class.

```
class Person {
  void dispName(String name) {
    print(name);
  }

  void dispAge(int age) {
    print(age);
  }
}
```

```
class Peter extends Person {
  void dispBranch(String branch) {
    print(branch);
}
}
class James extends Person {
  void result(String result) {
    print(result);
  }
}

void main() {
  James j = James();
  j.dispName("James");
  j.dispAge(24);
  j.result("Passed");

  Peter p = Peter();
  p.dispName("Peter");
  p.dispAge(21);
  p.dispBranch("Computer Science");
}

// Output:
// James
// 24
// Passed
// Peter
// 21
// Computer Science
```

- **Dart Super Constructor**: The child class can inherit all properties (methods, variables) and behavior of parent expect parent class constructor.& The superclass constructor can be invoke in sub class by using the **super()** constructor.

```
class Superclass {
  Superclass() {
    print("This is a superclass constructor");
  }
}

class Subclass extends Superclass {
  Subclass() {
    print("This is a subclass constructor");
  }

  void display() {
    print("Welcome to javatpoint");
  }
}

void main() {
  print("Dart Implicit Superclass constructor example");
  Subclass s = Subclass();
  s.display();
}

// Output:
// Dart Implicit Superclass constructor example
// This is a superclass constructor
// This is a subclass constructor
// Welcome to javatpoint
```

- **What is Polymorphism**?

  The polymorphism is a combination of the two Greek words **poly,** which means **many** and morph means **morphing into different forms or shapes**.

- **Method Overriding**

  When we declare the same method in the subclass, which is previously defined in the superclass is known as the method overriding. The subclass can define the same method by providing its own implementation, which is already exists in the superclass. The method in the superclass is called **method overridden,** and method in the subclass is **called method overriding**.

```
class Human {
  // Overridden method
  void run() {
    print("Human is running");
  }
}

class Man extends Human {
  // Overriding method
  void run() {
    print("Boy is running");
  }
}
void main() {
  Man m = Man();
  // This will call the child class version of run()
  m.run();
}
// Output:
//Boy is running
```

- **Dart Abstract Classes**: Abstract classes are the classes in Dart that has one or more abstract method. Abstraction is a part of the data encapsulation where the actual internal working of the function hides from the users.

- **Dart Interfaces :**To work with interface methods, the interface must be implemented by another class using the **implements** keyword. A class which is implemented the interface must provide a full implementation of all the methods that belongs to the interface. Following is the syntax of the implementing interface.

- **Example**

```
class Employee {
  void display() {
    print("I am working as an engineer");
  }
}

class Engineer implements Employee {
  @override
  void display() {
    print("I am an engineer in this company");
  }
}

void main() {
  Engineer eng = Engineer();
  eng.display();
}
// Output:
// I am an engineer in this company
```