Dart Building Blocks

1. Data Types in Dart

In Dart, there are several built-in data types. Here are some of the most commonly used ones:

1. **Numbers**:
    - int: Represents integer values.
    - double: Represents floating-point values.
2. **Strings**:
    - String: Represents a sequence of characters.
3. **Booleans**:
    - bool: Represents true or false values.
4. **Lists**:
    - List: Represents an ordered collection of objects.
5. **Maps**:
    - Map: Represents a collection of key-value pairs.
6. **Sets**:
    - Set: Represents an unordered collection of unique items.
7. **Runes**:
    - Runes: Represents a sequence of Unicode code points.
8. **Symbols**:
    - Symbol: Represents an operator or identifier declared in a Dart program.

Operators in Dart

Dart supports a variety of operators. Here are some of the most commonly used ones:

1. **Arithmetic Operators**:
    - Addition: +
    - Subtraction: -
    - Multiplication: *
    - Division: /
    - Integer Division: ~/
    - Modulus: %

2. **Equality and Relational Operators**:
    - Equal to: ==
    - Not equal to: !=
    - Greater than: >
    - Less than: <
    - Greater than or equal to: >=
    - Less than or equal to: <=
3. **Type Test Operators**:
    - as: Typecast
    - is: True if the object has the specified type
    - is!: True if the object does not have the specified type
4. **Bitwise Operators**:
    - Bitwise AND: &
    - Bitwise OR: |
    - Bitwise XOR: ^
    - Bitwise NOT: ~
    - Left shift: <<
    - Right shift: >>
5. **Assignment Operators**:
    - Assign: =
    - Add and assign: +=
    - Subtract and assign: -=
    - Multiply and assign: *=
    - Divide and assign: /=
    - Integer divide and assign: ~/=
    - Modulus and assign: %=
6. **Logical Operators**:
    - Logical AND: &&
    - Logical OR: ||
    - Logical NOT: !
7. **Conditional Operators**:

- Conditional: condition ? expr1 : expr2
- If null: expr1 ?? expr2

8. **Cascade Notation**:
    - programs: Allows you to make a sequence of operations on the same object.

2. Functions

In Dart, functions are a fundamental building block. They allow you to encapsulate code for reuse and organization. Here are some key points about functions in Dart:

1. **Defining a Function:**
    - **Functions are defined using the returnType functionName(parameters) { ... } syntax.**
2. **Optional Parameters:**
    - **Dart supports optional positional parameters and named parameters.**
3. **Anonymous Functions:**
    - **Functions can be assigned to variables or passed as arguments.**
4. **Arrow Functions:**
    - **For short functions, you can use the arrow syntax =>.**

3. Class and objects

Key Points:

1. **Class Definition:**
    - **Use the class keyword to define a class.**
    - **Define instance variables and methods inside the class.**
2. **Constructor:**
    - **Use a constructor to initialize instance variables.**
    - **Dart provides a shorthand syntax for constructors.**
3. **Creating Objects:**
    - **Use the new keyword (optional) followed by the class name and constructor parameters to create an object.**
4. **Accessing Members:**

- **Use the dot (.) notation to access instance variables and methods.**

Inheritance is a fundamental concept in object-oriented programming that allows a class to inherit properties and methods from another class. In Dart, you use the extends keyword to create a subclass that inherits from a superclass.

Key Points:

1. **Superclass:**
   - **The base class that provides properties and methods to be inherited.**
2. **Subclass:**
   - **The derived class that inherits from the superclass using the extends keyword.**
3. **Constructor:**
   - **The subclass constructor calls the superclass constructor using the super keyword.**
4. **Accessing Inherited Members:**
   - **The subclass can access the instance variables and methods of the superclass.**

4. Interfaces

In Dart, interfaces are implemented using classes. Any class can be used as an interface, and you can implement multiple interfaces in a single class. To implement an interface, you use the implements keyword.

Key Points:

1. **Interface Definition**:
   - Any class can act as an interface by defining methods that other classes can implement.
2. **Implementing Interfaces**:
   - Use the implements keyword to implement an interface.
   - A class that implements an interface must provide concrete implementations for all the methods defined in the interface.
3. **Multiple Interfaces**:

- A class can implement multiple interfaces by separating them with commas.