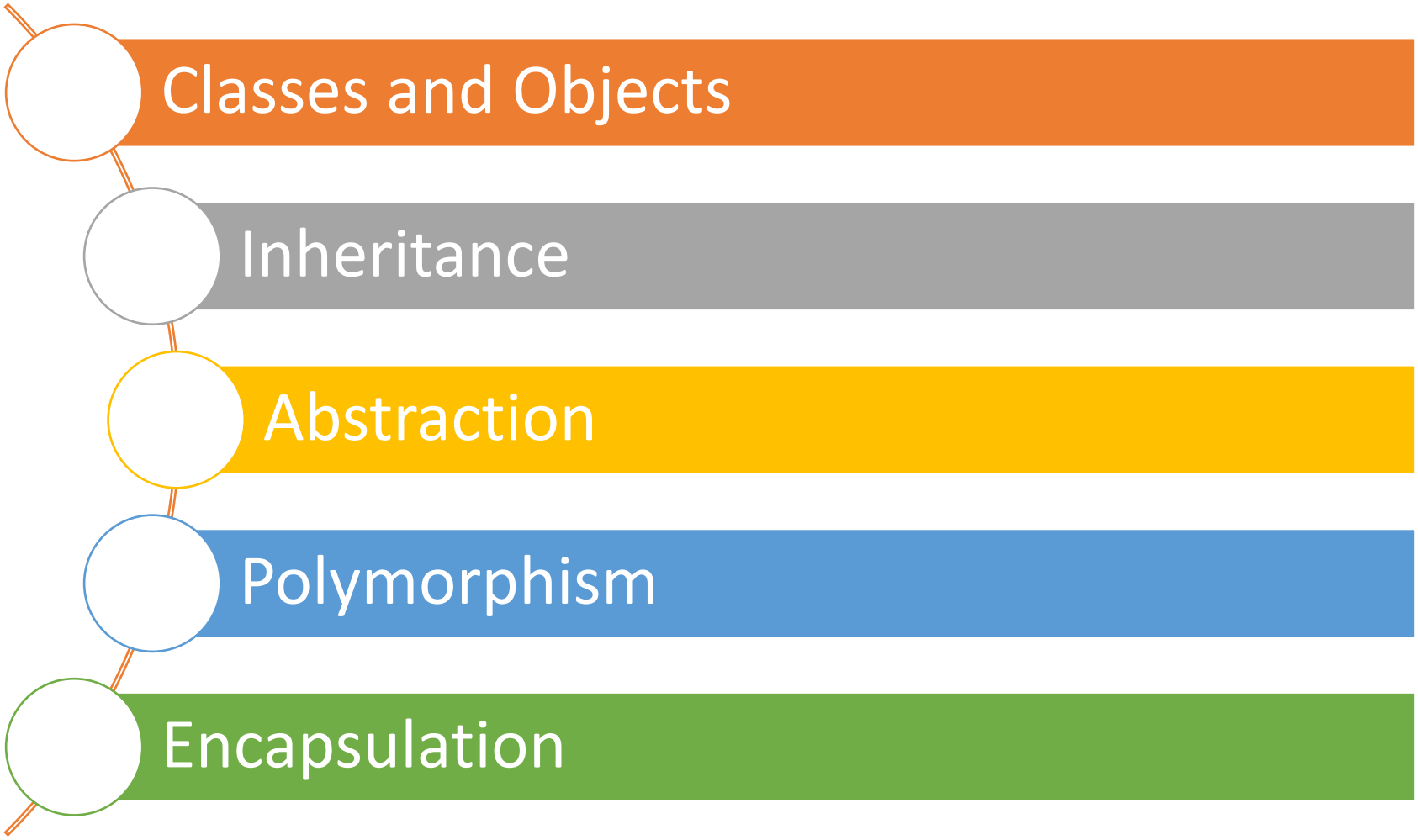
Session 4

Object-Oriented Programming in Dart

Session Overview

- Explain an overview of OOP concepts in Dart.
- Describe various types of methods in Dart.
- Elaborate on classes, objects, inheritance, polymorphism, interfaces, and abstraction.
- Explain constructors in Dart.
- Outline the purpose of the `this`, `static`, and `super` keywords in Dart.

OOP Approach



Classes and Objects

Classes are user-defined data types that describe the characteristics of an entity.

A class combines both data members and member functions.

An instance of a class is defined as an object.

Syntax for defining a class:

```
class ClassName  {
}
```

Syntax for creating objects:

```
ClassName ObjectName = ClassName();
```

Instance and Class Variables

Instance Variables	Class Variables
These are class variables that are accessed using instances of the class.	These are static variables that are declared inside the class.
Each instance of the class can have a different value of the instance variable.	A class variable has only one value and is shared by all objects of the class.
The value is retained till the object is active.	The value is retained till the program termination.

Example:
`class NewClass{`
`  int counter;`
`}`
	Example:
	`class NewClass{`
	`  static int counter;`
	`}`

Methods [1-2]

Class Methods

- They are declared using the `static` keyword.
- These methods are not accessed using objects.
- **Syntax:**

```
static Returntype methodname(arguments)
{
//statement
}
```

Instance Methods

- These methods are accessed only using objects of the class.
- They can be accessed within the class also using the `this` keyword.
- **Syntax:**

```
Returntype methodname(arguments)
{
//statement body
}
```

Methods [2-2]

Code Snippet 1:

```
class Student {
  var name = "Adelina";
  static void printName(name){
    print(name);
  }
}
void main(){
  Student.printName("Eliza");
}
```

Code Snippet 2:

```
class Student {
  var name = "Adelina";
  void printName(name){
    print(name);
  }
}
void main(){
  Student im = new Student();
  im.printName("Julia");
}
```

Getter and Setter Methods

Getter Method

- **Syntax:**

```
returntype get fieldname{
}
```

Setter Method

- **Syntax:**

```
set fieldname(value)
   {
}
```

Code Snippet 3:

```
class Employee {
  late String eName;
  String get ename {
    return eName;
  }
   set ename(String name) {
    this.eName = name;
  }
}
void main() {
  Employee emp = new Employee();
  emp.ename ='Jeff';
  print("Employee name Is : ${emp.ename}");
}
```

Inheritance

A class can inherit the properties and methods of another class.

The main class that has properties and methods is called the parent class.
The class that inherits the parent class is called the child class.

Syntax:

```
class ParentClass{
...
}
//single level inheritance
class ChildClass1 extends ParentClass
{
...
}
// multi level inheritance
class ChildClass2 extends ChildClass1
{
...
}
```

Polymorphism

- Polymorphism refers to one object having many forms.
- The existing features of a parent class are modified in the child class.
- Polymorphism is achieved using method overriding.

Syntax:

```
class ParentClass{
  void parentMethod(){
       //do something
  }
}
class ChildClass extends
ParentClass{
  @override
  void parentMethod(){
       //do something more
  }
}
```

Interfaces

- Abstract declaration of methods is provided in an interface.
- Every class implicitly defines an interface.
- All the methods of an inherited class must be re-defined.

```
class ClassName{
...
Returntype Method(){
//some functionality
}
}
```

```
class ClassName implements InterfaceClassName
{
...
@override
Returntype Method(){
//different functionality than that
//which was defined in parent class
}   }
```

Abstraction

Abstraction means hiding unnecessary information.

Abstract classes are created using the `abstract` keyword.

Syntax:

```
abstract class MainClass{
    void abstractPrintMethod();
}
class ImplementationClass implements MainClass{
    @override
    void abstractPrintMethod(){
            //do something
    }
}
```

Constructors [1-2]

- A constructor is a special method used to initialize objects. It has the same name as a class.
- A constructor is automatically called when an object is created.
- Default constructors have no parameters.
- Parameterized constructors take variables as arguments.
- A named constructor is used to create multiple constructors with different names in the same class.

Constructors [2-2]

Code Snippet 4:

```
class Student{
  Student(){
    print("Default Constructor");
  }

  Student(String name){
    print("Student name is: ${name}");
  }

  Student.namedConst(String branch){
    print("Branch name is: ${branch}");
```

```
void main() {

Student std1 = new Student();
Student std2 = new Student("Ray");
Student std3 = new
Student.namedConst("Computer
Science");
}
```

The `this` Keyword

The `this` keyword refers to the current class object.

It helps to remove ambiguity when the class variable and parameters have the same name.

Code Snippet 5:

```
void main()
{
  Student s1 = new Student('101');
}
```

```
class Student
{
  var stid;
  Student(var stid)
  {
    this.stid = stid;
    print("Dart this keyword Example");
    print("Student ID is : ${stid}");
  }
}
```

The `super` Keyword

The `super` keyword is used to refer to parent class properties or methods when both the parent and child class have the same method or property name.

Code Snippet 6:

```
class ParentClass {
  String subject = "Example of Super Keyword";
}
class SubClass extends ParentClass {
  String subject = "Science";
  void showMessage(){
    print(super.subject);
    print("$subject has ${subject.length} letters.");
  }
}

void main(){
  SubClass myClass = new SubClass();
  myClass.showMessage();
}
```

Summary

- Object-oriented programming is a programming approach based on four pillars: inheritance, polymorphism, abstraction, and encapsulation.
- In OOP, everything is treated as an object. OOP enforces security and generalization in a program.
- In the Dart programming language, a method is a block of code that is defined inside a class and runs only when called.
- Methods divide a large task into small parts and perform a specific program operation. This increases the reusability of code and enhances the program's approach.
- Getter and setter methods in Dart are used for getting and setting a value. The getter method is used to get, read, or retrieve the class fields whereas the setter method is used to set, write, or initialize the class fields.
- Constructors are used when an object is created in the program. They have the same name as the class and are often used to initialize values.
- Classes have their default constructor that is created by the compiler during the execution of the program, but the user can create custom constructors for a class. If the user creates his/her own constructor, the default constructor will be ignored. Constructors must not have a return type.