Mohammed ElShafey

1. Inheritance

Inheritance allows a class (child/subclass) to acquire properties and methods from another class (parent/superclass). This is achieved using the `extends` keyword. A parent class cannot access the properties or methods defined in its child classes.

- Single Inheritance: A class inherits properties from only one other class.
- Multilevel Inheritance: A class inherits from a child class, which itself inherits from a parent class, forming a chain (e.g., Child2 extends Child, Child extends Parent). Properties are inherited down the chain, meaning Child2 also gains properties from Parent.
- Hierarchical Inheritance: Multiple classes inherit from a single parent class. Each child class can also have its own unique properties in addition to those inherited from the parent.

2. Inheritance of Constructors

When a child class object is created, the parent class's constructor is automatically invoked first.

- A child class can directly access and modify the parent's properties and methods.
- If the parent class's constructor requires parameters, the child class's constructor must pass these parameters to the parent using the `super` keyword. The `super` keyword is crucial for distinguishing between properties/methods belonging to the current child class and those inherited from the parent class.
- The `this` keyword refers to properties within the current class.

3. Polymorphism

Polymorphism, meaning "many shapes," allows objects to take on multiple forms. In Dart, this is primarily achieved through method overriding.

- Method Overriding: A child class redefines or provides a new implementation for a method that is already present in its parent class.
- The `override` keyword is recommended before an overridden method to clearly indicate that it's inheriting and modifying a parent's method. It's not required if the method is not inherited.
- Polymorphism can be applied in various inheritance structures, including hierarchical inheritance.

4. Static Variable

A static variable is declared using the `static` keyword.

- It is associated with the class itself, not with individual objects (instances). This means its value is shared across all instances of that class.
- Static variables cannot be accessed directly through an object instance using dot notation (e.g., `object.staticVar`). Instead, they must be accessed directly through the class name (e.g., `ClassName.staticVar`).

5. Static Function

A static function (or method) is declared using the `static` keyword before its definition.

- Similar to static variables, a static function belongs to the class itself, not to an object.
- It can be called directly using the class name without needing to create an instance of the class (e.g., `ClassName.staticFunction()`).
- Static functions can access static variables directly.

6. Enum

An enum is a collection of fixed, unchanging values or constants.

- Declared using the `enum` keyword, enums provide a way to define a set of named constants (e.g., days of the week, connection states).
- While not strictly mandatory for a program to function, using enums is highly recommended for code organization, readability, and maintainability, as it reduces the potential for errors like typos or case sensitivity issues that might arise with plain strings.

7. Abstract Class

An abstract class is declared with the `abstract` keyword.

- It cannot be instantiated directly; you cannot create objects from an abstract class.
- Its primary purpose is to be inherited from using `extends`. Abstract classes serve as a blueprint or contract for other classes.
- Abstract classes can contain abstract methods, which are declared without a body (implementation). Any non-abstract class that extends an abstract class must implement (override) all of its abstract methods.
- Abstract classes can also contain regular (non-abstract) methods with implementations. Subclasses can choose to override these or use them as they are.

8. Interface

In Dart, there is no explicit `interface` keyword. Any class can act as an interface.

- A class acts as an interface when another class implements it.
- When a class implements an "interface" class, it is forced to provide implementations (override) for all the properties and methods defined in the interface class. This differs from `extends`, where properties/methods are inherited and can be optionally overridden.
- The `super` keyword cannot be used with `implements` because it's

not a direct inheritance relationship for values.
- A key advantage is that a class can implement multiple classes, unlike `extends` which only allows single inheritance.
- `implements` works with both regular ("concrete") classes and abstract classes.

9. Mixin

Mixins are used for sharing and reusing code across multiple classes.

- A mixin is declared using the `mixin` keyword.
- Classes incorporate a mixin's functionalities using the `with` keyword.
- Mixins cannot be instantiated (no objects can be created from them) and cannot have constructors.
- Mixins allow a class to incorporate functionalities from multiple mixins, effectively bypassing Dart's single-inheritance limitation for `extends`.
- They can contain properties (regular or static), regular methods, static methods, and abstract methods.
- Mixins can be used alongside `extends` in a class definition. They are useful for common, reusable functionalities like sending notifications or implementing pagination.

10. Factory Constructor

A factory constructor is a special type of constructor declared with the `factory` keyword. Other constructors are typically referred to as "generative constructors".

- The main purpose of a factory constructor is to return an instance (object) of the class itself or a subclass.
- Unlike generative constructors, a factory constructor must always include a `return` statement.
- It allows for custom logic during object creation, such as:
  - Returning an existing instance rather than always creating a new one, promoting object reuse.

  - Implementing validation (e.g., ensuring input values meet certain criteria) before creating an instance.
  - Returning an instance of a subclass based on specific conditions or input parameters.
- Factory constructors can be named or unnamed.
- They cannot directly access instance members (non-static properties or methods of the current class) using `this` inside the factory constructor.