Name: Sutariya dhruvik v.

Subject: dart

21. What is inheritance?

Ans: [In object-oriented programming, **inheritance** is a mechanism that allows a class to inherit properties and methods from another class](https://www.analyticsvidhya.com/blog/2020/10/inheritance-object-oriented-programming/) . [In other words, it is a way to create a new class based on an existing class](https://www.analyticsvidhya.com/blog/2020/10/inheritance-object-oriented-programming/) . [The existing class is called the **parent class** or **superclass**, and the new class is called the **child class** or **subclass**](https://www.codecademy.com/resources/blog/what-is-inheritance/) . [The child class can access all the properties and methods of the parent class, and it can also add its own properties and methods](https://www.codecademy.com/resources/blog/what-is-inheritance/) .

[There are three types of inheritance in object-oriented programming: single inheritance, multiple inheritance, and multilevel inheritance](https://www.analyticsvidhya.com/blog/2020/10/inheritance-object-oriented-programming/) . [In single inheritance, a class inherits only one parent class](https://www.analyticsvidhya.com/blog/2020/10/inheritance-object-oriented-programming/) . [In multiple inheritance, a class inherits more than one parent class](https://www.analyticsvidhya.com/blog/2020/10/inheritance-object-oriented-programming/) . [In multilevel inheritance, a subclass inherits from a parent class, which in turn inherits from another parent class](https://www.analyticsvidhya.com/blog/2020/10/inheritance-object-oriented-programming/) .

22. Which inheritance is not supported by Dart? Why? B3. What is advantage of inheritance?

Ans: [Dart supports **single inheritance**](https://www.darttutorial.org/dart-tutorial/dart-inheritance/) . [This means that a class can inherit from only one parent class](https://www.darttutorial.org/dart-tutorial/dart-inheritance/) . [Dart does not support **multiple inheritance**](https://www.darttutorial.org/dart-tutorial/dart-inheritance/) . [Multiple inheritance is a feature in object-oriented programming where a class can inherit from more than one parent class](https://www.geeksforgeeks.org/dart-concept-of-inheritance/) .

The reason why Dart does not support multiple inheritance is to avoid the **diamond problem**. The diamond problem occurs when two classes inherit from the same superclass, and a third class inherits from both of these classes. [This creates an ambiguity in the inheritance hierarchy, as the third class has two copies of the same superclass](https://www.darttutorial.org/dart-tutorial/dart-inheritance/) .

The advantages of inheritance include:

1. [**Code reuse**: Inheritance allows you to reuse code from existing classes, which can save time and effort in developing new classes](https://www.darttutorial.org/dart-tutorial/dart-inheritance/) .
2. [**Flexibility**: Inheritance makes your code more flexible by allowing you to modify the behavior of existing classes without changing their code directly](https://www.darttutorial.org/dart-tutorial/dart-inheritance/) .
3. [**Overriding**: Inheritance allows you to override methods of the parent class in the child class, which can help you customize the behavior of your code](https://www.darttutorial.org/dart-tutorial/dart-inheritance/) .
4. [**Data hiding**: Inheritance allows you to hide data members of the parent class from the child class, which can help you protect sensitive data and maintain data integrity](https://www.geeksforgeeks.org/dart-concept-of-inheritance/) .

23. Difference between inheritance and encapsulation. B5. Difference between inheritance and abstraction.

Ans: **Inheritance** and **encapsulation** are two fundamental concepts in object-oriented programming. [Inheritance is a mechanism that allows a class to inherit properties and methods from another class](https://www.developer.com/design/encapsulation-vs-inheritance/) . [In other words, it is a way to create a new class based on an existing class](https://www.developer.com/design/encapsulation-vs-inheritance/) . [The existing class is called the **parent class** or **superclass**, and the new class is called the **child class** or **subclass**](https://www.developer.com/design/encapsulation-vs-inheritance/) . [The child class can access all the properties and methods of the parent class, and it can also add its own properties and methods](https://www.developer.com/design/encapsulation-vs-inheritance/) .

[On the other hand, **encapsulation** is a technique that hides the internal details of a class from the outside world](https://www.developer.com/design/encapsulation-vs-inheritance/) . [It is a way to protect the data and behavior of an object from being accessed by other objects](https://www.developer.com/design/encapsulation-vs-inheritance/) . [Encapsulation allows you to define a public interface for your class, which specifies how other objects can interact with it](https://www.developer.com/design/encapsulation-vs-inheritance/) . [The implementation details of the class are hidden from other objects, which helps to maintain data integrity and prevent unwanted modifications](https://www.developer.com/design/encapsulation-vs-inheritance/) .

[The main difference between inheritance and encapsulation is that inheritance is about creating new classes based on existing classes, while encapsulation is about hiding the implementation details of a class from other objects](https://www.developer.com/design/encapsulation-vs-inheritance/) .

**Abstraction**, on the other hand, is another fundamental concept in object-oriented programming. [Abstraction is a way to focus on the essential features of an object while ignoring its non-essential details](https://www.developer.com/design/encapsulation-vs-inheritance/) . [It allows you to create abstract classes or interfaces that define a set of methods without providing any implementation details](https://www.developer.com/design/encapsulation-vs-inheritance/) . [The implementation details are left to the subclasses that inherit from these abstract classes or interfaces](https://www.developer.com/design/encapsulation-vs-inheritance/) .

[The main difference between inheritance and abstraction is that inheritance allows you to create new classes based on existing classes, while abstraction allows you to define abstract classes or interfaces that provide a common set of methods for multiple classes to implement](https://www.developer.com/design/encapsulation-vs-inheritance/) .

24. Difference between inheritance and polymorphism

Ans: **Inheritance** and **polymorphism** are two fundamental concepts in object-oriented programming.

[**Inheritance** is a mechanism that allows a class to inherit properties and methods from another class](https://www.geeksforgeeks.org/difference-between-inheritance-and-polymorphism/) . [In other words, it is a way to create a new class based on an existing class](https://pediaa.com/difference-between-inheritance-and-polymorphism/) . [The existing class is called the **parent class** or **superclass**, and the new class is called the **child class** or **subclass**](https://www.geeksforgeeks.org/difference-between-inheritance-and-polymorphism/) . [The child class can access all the properties and methods of the parent class, and it can also add its own properties and methods](https://www.geeksforgeeks.org/difference-between-inheritance-and-polymorphism/) .

[On the other hand, **polymorphism** is a concept that allows objects of different classes to be treated as if they were objects of the same class](https://www.geeksforgeeks.org/difference-between-inheritance-and-polymorphism/) . [It allows you to write code that can work with objects of different classes in a uniform way](https://www.geeksforgeeks.org/difference-between-inheritance-and-polymorphism/) . [Polymorphism can be achieved through method overloading and method overriding](https://www.geeksforgeeks.org/difference-between-inheritance-and-polymorphism/) .

[The main difference between inheritance and polymorphism is that inheritance is about creating new classes based on existing classes, while polymorphism is about treating objects of different classes as if they were objects of the same class](https://www.geeksforgeeks.org/difference-between-inheritance-and-polymorphism/) .

25. Can we override static method in Dart?

Ans: [No, it is not possible to override a method that has been declared static in the parent class in Dart](https://stackoverflow.com/questions/71471938/how-can-i-inherit-static-methods-in-dart-flutter) . [Static methods are not part of the class interface, and they are just a scope-restricted form of global functions](https://stackoverflow.com/questions/71471938/how-can-i-inherit-static-methods-in-dart-flutter) . [Therefore, it is not possible to inherit or override static methods in Dart](https://stackoverflow.com/questions/71471938/how-can-i-inherit-static-methods-in-dart-flutter) .

26. Can we overload static method in Dart?

Ans: [No, it is not possible to overload a method that has been declared static in Dart](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) . [Static methods are not part of the class interface, and they are just a scope-restricted form of global functions](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) . [Therefore, it is not possible to overload static methods in Dart](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) .

27. Can a class implement more than one interface? B10. Can a class extend more than one class in Dart?

Ans: [Yes, a class can implement more than one interface in Dart](https://stackoverflow.com/questions/71486826/does-dart-support-multiple-inheritance) . [An interface is a collection of abstract methods that define a set of behaviors that a class can implement](https://stackoverflow.com/questions/71486826/does-dart-support-multiple-inheritance) . [A class can implement multiple interfaces by separating them with commas in the implements clause](https://www.geeksforgeeks.org/dart-concept-of-inheritance/) . Here is an example:

abstract class InterfaceA {

void methodA();

}

abstract class InterfaceB {

void methodB();

}

class MyClass implements InterfaceA, InterfaceB {

@override

void methodA() {

print('Method A');

}

@override

void methodB() {

print('Method B');

}

}

[On the other hand, Dart does not support multiple inheritance](https://stackoverflow.com/questions/71486826/does-dart-support-multiple-inheritance) . Multiple inheritance is a feature in object-oriented programming where a class can inherit from more than one parent class . In Dart, a class can only extend one parent class at a time . However, Dart has mixins, which allow you to reuse code across multiple classes without using inheritance . Mixins are a way to add functionality to a class without creating a new subclass

28. Can an interface extend more than one interface in Dart?

Ans: [Yes, an interface can extend more than one interface in Dart](https://stackoverflow.com/questions/62011905/is-it-possible-to-implement-multiple-interfaces-in-dart-if-so-is-it-possible-t) . This is known as **interface inheritance** . Interface inheritance allows you to create a new interface that includes all the methods and properties of multiple existing interfaces . You can use the extends keyword to specify the parent interfaces that your new interface should inherit from . Here is an example:

abstract class InterfaceA {

void methodA();

}

abstract class InterfaceB {

void methodB();

}

abstract class InterfaceC {

void methodC();

}

abstract class InterfaceD extends InterfaceA implements InterfaceB, InterfaceC {

void methodD();

}

In this example, InterfaceD extends InterfaceA and implements InterfaceB and InterfaceC. Therefore, InterfaceD includes all the methods and properties of InterfaceA, InterfaceB, and InterfaceC.

29. What will happen if a class implements two interfaces and they both have a method with same name and signature?

Ans: [If a class implements two interfaces, and each interface defines a method that has identical signature, then in effect there is only one method, and they are not distinguishable](https://stackoverflow.com/questions/2801878/implementing-two-interfaces-in-a-class-with-same-method-which-interface-method) . [If the two methods have conflicting return types, then it will be a compilation error](https://stackoverflow.com/questions/2801878/implementing-two-interfaces-in-a-class-with-same-method-which-interface-method) . [This is the general rule of inheritance, method overriding, hiding, and declarations, and applies also to possible conflicts not only between 2 inherited interface methods but also an interface and a superclass method or even just conflicts due to type erasure of generics](https://stackoverflow.com/questions/2801878/implementing-two-interfaces-in-a-class-with-same-method-which-interface-method) .

30. Can we pass an object of a subclass to a method expecting an object of the super class? B14. Are static members inherited to sub classes?

Ans: [Yes, it is possible to pass an object of a subclass to a method that expects an object of the superclass in Dart](https://stackoverflow.com/questions/71471938/how-can-i-inherit-static-methods-in-dart-flutter) . This is known as **polymorphism**. [Polymorphism allows objects of different classes to be treated as if they were objects of the same class](https://stackoverflow.com/questions/71471938/how-can-i-inherit-static-methods-in-dart-flutter) . [In Dart, you can pass an object of a subclass to a method that expects an object of the superclass, and the method will treat the object as if it were an object of the superclass](https://stackoverflow.com/questions/71471938/how-can-i-inherit-static-methods-in-dart-flutter) .

[On the other hand, static members are not inherited to sub classes in Dart](https://stackoverflow.com/questions/71471938/how-can-i-inherit-static-methods-in-dart-flutter) . [Static members are not part of the class interface, and they are just a scope-restricted form of global functions](https://stackoverflow.com/questions/71471938/how-can-i-inherit-static-methods-in-dart-flutter) . [Therefore, it is not possible to inherit or override static members in Dart](https://stackoverflow.com/questions/71471938/how-can-i-inherit-static-methods-in-dart-flutter)

31. What happens if the parent and the child class have a field with same identifier? B16. Are constructors and initializers also inherited to sub classes?

Ans: [If a parent and a child class have a field with the same identifier, then the child class field will **hide** the parent class field 1](https://howtodoinjava.com/java/oops/java-inheritance/) . [This means that when you access the field using an object of the child class, you will get the value of the child class field, even if the object was created using the parent class constructor](https://howtodoinjava.com/java/oops/java-inheritance/) [1](https://howtodoinjava.com/java/oops/java-inheritance/) . [However, you can still access the parent class field using the super keyword](https://howtodoinjava.com/java/oops/java-inheritance/) [1](https://howtodoinjava.com/java/oops/java-inheritance/) .

For example:

class Parent {

int x = 1;

}

class Child extends Parent {

int x = 2;

}

void main() {

var obj = Child();

print(obj.x); // Output: 2

print((obj as Parent).x); // Output: 1

}

In this example, Child is a subclass of Parent, and both classes have a field named x. When you create an object of Child, it will have two fields named x: one from the Parent class and one from the Child class. When you access the x field using an object of Child, you will get the value of the Child class field, which is However, when you cast the object to Parent and access the x field, you will get the value of the Parent class field, which is.

[Constructors and initializers are inherited to subclasses in Dart](https://howtodoinjava.com/java/oops/java-inheritance/) . [When you create an object of a subclass, it will call the constructor of its superclass to initialize its inherited fields](https://howtodoinjava.com/java/oops/java-inheritance/) . [If you don’t define a constructor for your subclass, it will use the default constructor provided by its superclass](https://howtodoinjava.com/java/oops/java-inheritance/) . [You can also override constructors and initializers in your subclass to provide custom initialization logic](https://howtodoinjava.com/java/oops/java-inheritance/) .

32. How do you restrict a member of a class from inheriting by its sub classes?

Ans: To restrict a member of a class from being inherited by its subclasses in Dart, you can declare the member as **private**. [Private members are not inherited by subclasses](https://stackoverflow.com/questions/20959118/how-to-create-private-methods-that-can-be-redefined-in-subclasses-in-dart) . Here is an example:

class Student {

String name = "Jai";

int \_rollNo = 11;

}

class Main extends Student {

void show () {

print(name);

print(\_rollNo); // This will result in an error

}

}

void main () {

Main object = new Main ();

object.show();

}

In the above example, \_rollNo is declared as private in the Student class. Therefore, it cannot be inherited by the Main class, which extends Student. [When we try to access \_rollNo in the show() method of the Main class, it results in a compile-time error](https://stackoverflow.com/questions/20959118/how-to-create-private-methods-that-can-be-redefined-in-subclasses-in-dart) .

33. How do you implement multiple inheritance in Dart?

Ans: [Dart does not support multiple inheritance](https://www.geeksforgeeks.org/dart-concept-of-inheritance/) . However, you can achieve similar functionality using **mixins**. [A mixin is a class that contains methods for use by other classes without having to be the parent class of those other classes](https://stackoverflow.com/questions/10502649/when-to-use-interfaces-in-dart) . Here is an example:

class A {

void showA() {

print("A");

}

}

class B {

void showB() {

print("B");

}

}

mixin C on A, B {

void showC() {

print("C");

}

}

class D with C {}

void main() {

D object = new D();

object.showA();

object.showB();

object.showC();

}

In the above example, A and B are two classes that contain methods showA() and showB() respectively. The mixin keyword is used to define a mixin named C that uses both A and B. The D class uses the C mixin using the with keyword. [Therefore, the D class has access to all three methods: showA(), showB(), and showC()](https://www.geeksforgeeks.org/dart-concept-of-inheritance/) .

34. Can a class extend by itself in Dart?

Ans: [No, a class cannot extend itself in Dart](https://stackoverflow.com/questions/56879606/how-to-extend-a-class-in-dart-flutter) . Doing so would result in an infinite loop of inheritance. However, a class can extend another class that is derived from itself, creating a **circular inheritance**. Here is an example:

class A extends B {}

class B extends A {}

void main () {

A object = new A ();

}

In the above example, A extends B, which in turn extends A. Therefore, A and B are derived from each other, creating a circular inheritance. [When we create an object of the A class, it will inherit all the properties and methods of both A and B](https://stackoverflow.com/questions/56879606/how-to-extend-a-class-in-dart-flutter) .

35. How do you override a private method in Dart?

Ans: [To override a private method in Dart, you can’t use the @override annotation because private members are not inherited by subclasses](https://stackoverflow.com/questions/66277583/how-to-override-private-fields-and-methods-in-dart) . However, you can create a new method with the same name as the private method in the subclass. This new method will not override the private method, but it will have the same name and signature. Here is an example:

class A {

void \_show() {

print("A");

}

}

class B extends A {

void show() {

\_show();

}

}

void main() {

B object = new B();

object.show(); // Output: A

}

In the above example, A is a class that contains a private method \_show(). [The B class extends A and creates a new public method show() that calls \_show()](https://stackoverflow.com/questions/66277583/how-to-override-private-fields-and-methods-in-dart) .

36. When to overload a method in Dart and when to override it?

Ans: [In Dart, **method overloading is not supported**](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) . Therefore, you cannot have two or more methods in the same class with the same name but different arguments. [Instead, you can use optional named or unnamed parameters to achieve similar functionality](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) .

On the other hand, **method overriding** is supported in Dart. Method overriding means having two methods with the same name and arguments, but different implementations. [The subclass can override the implementation of a method that it has inherited from its superclass](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) . Here is an example:

class A {

void show() {

print("A");

}

}

class B extends A {

@override

void show() {

print("B");

}

}

void main() {

B object = new B();

object.show(); // Output: B

}

In the above example, A is a class that contains a method show(). [The B class extends A and overrides the implementation of show() using the @override annotation](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) .

37. What the order is of extends and implements keyword on Dart class declaration?

Ans: [In Dart, the extends keyword is used to create a subclass that inherits properties and characteristics from another class](https://www.geeksforgeeks.org/dart-extends-vs-with-vs-implements/) . [The implements keyword is used to implement an interface by forcing the redefinition of the functions](https://www.geeksforgeeks.org/dart-extends-vs-with-vs-implements/) .

The order of extends and implements keywords in a Dart class declaration is as follows:

class ClassName extends SuperClassName implements InterfaceName {

// Class members

}

[Here, ClassName is the name of the class being declared, SuperClassName is the name of the superclass being extended, and InterfaceName is the name of the interface being implemented](https://www.geeksforgeeks.org/dart-extends-vs-with-vs-implements/) .

[If a class extends a superclass and implements one or more interfaces, then the extends keyword must come before the implements keyword](https://www.geeksforgeeks.org/dart-extends-vs-with-vs-implements/) .

38. How do you prevent overriding a Dart method without using the final modifier?

Ans: [To prevent overriding a Dart method without using the final modifier, you can use the @nonVirtual annotation provided by the package:meta library](https://stackoverflow.com/questions/63984838/is-it-possible-to-declare-a-method-as-final-in-a-class-in-dart) . This annotation disallows overriding methods in subclasses. Here is an example:

import 'package:meta/meta.dart';

class A {

@nonVirtual

void foo() {

print('A');

}

}

class B extends A {

void foo() {

print('B');

}

}

void main() {

B().foo(); // Output: A

}

In the above example, A is a class that contains a method foo(). The @nonVirtual annotation is used to disallow overriding of foo() in subclasses. The B class extends A and tries to override the implementation of foo(). However, since foo() is marked as non-virtual, it cannot be overridden by the subclass. [Therefore, when we call foo() on an object of the B class, it calls the implementation of foo() in the superclass A](https://stackoverflow.com/questions/63984838/is-it-possible-to-declare-a-method-as-final-in-a-class-in-dart) .

39. What are the rules of method overriding in Dart?

Ans: In Dart, method overriding occurs when a child class tries to override the parent class’s method. When a child class extends a parent class, it gets full access to the methods of the parent class and thus it overrides the methods of the parent class. It is achieved by re-defining the same method present in the parent class. This method is helpful when you have to perform different functions for a different child class, so we can simply re-define the content by overriding it. Here are some rules of method overriding in Dart:

1. A method can be overridden only in the child class, not in the parent class itself.
2. Both the methods defined in the child and the parent class should be the exact copy, from name to argument list except the content present inside the method i.e. it can and can’t be the same.
3. A method declared final or static inside the parent class can’t be overridden by the child class.
4. Constructors of the parent class can’t be inherited, so they can’t be overridden by the child class.

Here is an example of method overriding in Dart:

class SuperClass {

void show() {

print("This is SuperClass");

}

}

class SubClass extends SuperClass {

@override

void show() {

print("This is SubClass");

}

}

void main() {

SubClass object = new SubClass();

object.show(); // Output: This is SubClass

}

In this example, SuperClass is a parent class that contains a method show(). [The SubClass extends SuperClass and overrides the implementation of show() using the @override annotation](https://www.geeksforgeeks.org/method-overriding-in-dart/) .

40. Difference between method overriding and overloading in Dart.

Ans: [In Dart, **method overloading is not supported**](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) . Therefore, you cannot have two or more methods in the same class with the same name but different arguments. [Instead, you can use optional named or unnamed parameters to achieve similar functionality](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) .

On the other hand, **method overriding** is supported in Dart. Method overriding means having two methods with the same name and arguments, but different implementations. [The subclass can override the implementation of a method that it has inherited from its superclass](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) . Here is an example:

class A {

void show() {

print("A");

}

}

class B extends A {

@override

void show() {

print("B");

}

}

void main() {

B object = new B();

object.show(); // Output: B

}

In the above example, A is a class that contains a method show(). [The B class extends A and overrides the implementation of show() using the @override annotation](https://stackoverflow.com/questions/62626833/why-does-dart-not-allow-method-overloading) .

41. What happens when a class implements two interfaces and both declare field (variable) with same name?

Ans: [When a class implements two interfaces and both declare a field (variable) with the same name, the class which implements those interfaces can’t identify which variable to access and throws an error](https://stackoverflow.com/questions/2801878/implementing-two-interfaces-in-a-class-with-same-method-which-interface-method) . [Therefore, to resolve this issue, you can access the variable using the interface name as the reference](https://stackoverflow.com/questions/2801878/implementing-two-interfaces-in-a-class-with-same-method-which-interface-method) . Here is an example:

abstract class A {

int x = 10;

}

abstract class B {

int x = 20;

}

class C implements A, B {

void show() {

print(A.x); // Output: 10

print(B.x); // Output: 20

}

}

void main() {

C object = new C();

object.show();

}

In the above example, A and B are two interfaces that declare a field x. The C class implements both A and B. To access the field x of interface A, we use A.x. [Similarly, to access the field x of interface B, we use B.x](https://stackoverflow.com/questions/2801878/implementing-two-interfaces-in-a-class-with-same-method-which-interface-method) .

42. Can a subclass instance method override a superclass static method?

Ans: [No, a subclass instance method cannot override a superclass static method in Dart](https://www.codingninjas.com/codestudio/library/dart-methods-and-method-overriding) . A static method belongs to the class and not to any instance of the class. [Therefore, it cannot be overridden by an instance method of a subclass](https://dart.dev/language/extend) . Here is an example:

class A {

static void show() {

print("This is A");

}

}

class B extends A {

void show() {

print("This is B");

}

}

void main() {

B object = new B();

object.show(); // Output: This is B

A.show(); // Output: This is A

}

In the above example, A is a class that contains a static method show(). The B class extends A and creates an instance method show() that tries to override the implementation of show() in the superclass. However, since show() is a static method, it cannot be overridden by an instance method of the subclass. Therefore, when we call show() on an object of the B class, it calls the implementation of show() in the subclass. [When we call show() on the class itself, it calls the implementation of show() in the superclass](https://www.codingninjas.com/codestudio/library/dart-methods-and-method-overriding) .

43. Can a subclass static method hide superclass instance method?

Ans:  [a subclass instance method cannot override a superclass static method in Dart](https://stackoverflow.com/questions/7420212/hiding-methods-of-superclass) . A static method belongs to the class and not to any instance of the class. [Therefore, it cannot be overridden by an instance method of a subclass](https://stackoverflow.com/questions/7420212/hiding-methods-of-superclass) . Here is an example:

class A {

static void show() {

print("This is A");

}

}

class B extends A {

void show() {

print("This is B");

}

}

void main() {

B object = new B();

object.show(); // Output: This is B

A.show(); // Output: This is A

}

In the above example, A is a class that contains a static method show(). The B class extends A and creates an instance method show() that tries to override the implementation of show() in the superclass. However, since show() is a static method, it cannot be overridden by an instance method of the subclass. Therefore, when we call show() on an object of the B class, it calls the implementation of show() in the subclass. [When we call show() on the class itself, it calls the implementation of show() in the superclass](https://stackoverflow.com/questions/7420212/hiding-methods-of-superclass) .

44. Can a superclass access subclass member?

Ans: [a superclass cannot access a subclass member in Dart](https://stackoverflow.com/questions/47897536/accessing-subclass-members-through-super-class-reference-variables) . A reference variable of a superclass can be assigned a reference to an object of any subclass derived from that superclass. [However, when a reference to a subclass object is assigned to a superclass reference variable, you will have access only to those parts of the object defined by the superclass](https://www.scientecheasy.com/2020/07/java-superclass-subclass.html/) . Therefore, you cannot access the members of the subclass using the superclass reference variable. Here is an example:

class A {

void show() {

print("This is A");

}

}

class B extends A {

void display() {

print("This is B");

}

}

void main() {

A object = new B();

object.show(); // Output: This is A

object.display(); // Error: The method 'display' isn't defined for the class 'A'.

}

In the above example, A is a class that contains a method show(). The B class extends A and creates a method display(). An object of the B class is created and assigned to a reference variable of the A class. When we call show() on this object, it calls the implementation of show() in the superclass. [However, when we try to call display() on this object, it results in an error because display() is not defined in the superclass](https://stackoverflow.com/questions/47897536/accessing-subclass-members-through-super-class-reference-variables) .

45. Difference between object oriented and object based language.

Ans:

|  |  |
| --- | --- |
| Object-oriented Programming Language | Object-based Programming Language |
| All the characteristics and features of object-oriented programming are supported. | All characteristics and features of object-oriented programming, such as inheritance and polymorphism are not supported. |
| These types of programming languages don't have a built-in object. Example: C++. | These types of programming languages have built-in objects. Example: JavaScript has a window object. |
| Java is an example of object-oriented programing language which supports creating and inheriting (which is reusing of code) one class from another. | VB is another example of object-based language as you can create and use classes and objects, but inheriting classes is not supported |

46. Create a program using List

Ans: void main() {

// Creating a list

List<int> numbers = [1, 2, 3, 4, 5];

// Accessing elements of the list

print(numbers[0]); // Output: 1

print(numbers[2]); // Output: 3

// Adding elements to the list

numbers.add(6);

numbers.addAll([7, 8]);

// Removing elements from the list

numbers.remove(1);

numbers.removeAt(3);

// Updating elements of the list

numbers[0] = 0;

// Printing the final list

print(numbers); // Output: [0, 2, 3, 5, 7]

}

47. Create a program using Set

Ans: void main() {

// Creating a set

Set<int> numbers = {1, 2, 3, 4, 5};

// Accessing elements of the set

print(numbers.contains(3)); // Output: true

print(numbers.contains(6)); // Output: false

// Adding elements to the set

numbers.add(6);

numbers.addAll([7, 8]);

// Removing elements from the set

numbers.remove(1);

// Printing the final set

print(numbers); // Output: {2, 3, 4, 5, 6, 7, 8}

}

48. Create a program using Map

Ans: void main() {

// Creating a map using map literals

Map<String, int> ages = {'John': 25, 'Jane': 30, 'Bob': 35};

// Accessing elements of the map

print(ages['John']); // Output: 25

print(ages['Jane']); // Output: 30

// Adding elements to the map

ages['Alice'] = 20;

ages.addAll({'Tom': 40, 'Jerry': 45});

// Removing elements from the map

ages.remove('Bob');

// Updating elements of the map

ages['John'] = 26;

// Printing the final map

print(ages); // Output: {John: 26, Jane: 30, Alice: 20, Tom: 40, Jerry: 45}

}