Abstract class

✓ abstract Method
abstract class

```
class Test
{
    p.s.v.main()
    {
        ✓ Super s1;
        ✗ s1=new Super();
        ✓ Sub s2=new Sub();
    }
}
```

```
abstract class Super
{
    Super()
    {
        s.o.p("super");
    }
    void meth1()
    {
        s.o.p("super meth1");
    }
    abstract void meth2();
}
```

```
class Sub extends Super
{
    void meth2()
    {
        s.o.p("Sub meth2");
    }
}
```

--> There are two types of classes: Abstract class and Concrete class.
--> "Abstraction" is a process of hiding the implementation or way of managing complexity by hiding unnecessary details and showing only functionality to the user.
--> If "abstract" keyword is used before the class then it is an abstract class but if nothing is written before class then it is a concrete class.
--> Object of an Abstract class cannot be created but object of concrete class can be created.
--> Reference of abstract class is allowed.
--> "Method" which is not having a body is known as Abstract method, the method must be declared as abstract. Mtlb hma "method" ka andar kuch bhi nahi likhna hai.
--> A class is "abstract class" if at least one of the methods is abstract.
--> If any "sub-class" inherits abstract class then that "sub-class" also becomes "abstract class"; lekin aagar ham "sub-class" ko "concrete class" mai change karna chahta hai to hma "undefined method of super-class" ko override karna hoga

Abstract class

Do and Dont

1.
```
abstract class Test
{
  =
}
```
✓ `Test t;`
✗ `t=new Test();`

2. ✗
```
abstract final class Test
{
  -
}
```

3. ✗
```
abstract static class Test
{
  -
}
```

4.
```
abstract class Test
{
  ✗ final abstract void display();
}
```

5.
```
abstract class Test1
{
  abstract void display();
}

class Test2 extends Test1
{
  ✗ void display()
  {
    =
  }
}
```

--> An Abstract class cannot be final because if it is made final then it cannot be extended whereas abstract class is meant for inheritance.

--> An Abstract method cannot be final because if it made final then it cannot be overridden whereas Abstract method is meant for overriding.

--> Abstract Class and method can neither be final nor static.

--> A Sub class must override an abstract method or else it will become abstract class.

Java

explain abstartct class with context of java.

Abstract Class in Java

An **abstract class** in Java is a class that cannot be instantiated directly. Instead, it is meant to be **subclassed**, and the subclasses provide specific implementations for its abstract methods. The purpose of abstract classes is to provide a common structure or blueprint for other classes while enforcing that certain methods must be implemented by any concrete subclass.

Key Features of an Abstract Class

1. **Abstract Methods**:
   - An abstract class can contain abstract methods. These are methods that do not have a body (implementation) in the abstract class itself.
   - Subclasses that inherit from the abstract class are required to provide concrete implementations of these abstract methods.

   Example:

```java
abstract class Animal {
    // Abstract method
    abstract void sound();

    // Concrete method with implementation
    void sleep() {
        System.out.println("Sleeping...");
    }
}
```

2. **Concrete Methods**:
   - An abstract class can also contain concrete methods—methods with full implementation. This allows the abstract class to share common functionality across all subclasses without having to reimplement those methods in each subclass.
3. **No Instantiation**:
   - You cannot create an object of an abstract class using the `new` keyword. However, you can create an object of a subclass that extends the abstract class.

   Example:

```java
Animal myAnimal = new Dog();  // Not new Animal(), because Animal is abstract
```

4. **Partial Implementation**:
    - Abstract classes allow partial implementation. This means some methods can have common behavior (implemented methods), and others are left for subclasses to define (abstract methods).

Example of an Abstract Class

```java
abstract class Animal {
    // Abstract method
    abstract void sound();

    // Concrete method
    void eat() {
        System.out.println("This animal is eating.");
    }
}

// Subclass implementing the abstract method
class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        // Creating an object of the subclass
        Animal myDog = new Dog();
        myDog.sound();  // Outputs: Dog barks
        myDog.eat();    // Outputs: This animal is eating.
    }
}
```

In this example, `**Animal**` is an abstract class with an abstract method `**sound()**` and a concrete method `**eat()**`. The `**Dog**` class extends `**Animal**` and provides its specific implementation for the `**sound()**` method, while inheriting the `**eat()**` method.

When to Use an Abstract Class

- **When you want to share code** among several closely related classes but also want to enforce certain behaviors by requiring subclasses to implement specific methods.
- **When a superclass should not be instantiated directly** but should serve as a base for other subclasses.
- **When you need both concrete and abstract methods** in a class.