Abstract Methods in Java - Complete Details

1. What We Can Do With Abstract Methods

- Declare an abstract method inside an abstract class or an interface. - Use the 'abstract' keyword in the method declaration. - Leave the method without a body (only method signature). - Provide implementation of abstract methods in a concrete subclass. - Have multiple abstract methods inside an abstract class or interface. - Mix abstract and non-abstract methods in an abstract class. - Override abstract methods in subclasses to provide actual behavior. - Abstract methods can have any valid access modifier in an abstract class (public, protected, default). - Use abstract methods to enforce a contract or a "must-do" behavior for subclasses. - An abstract class can inherit another abstract class and choose not to implement all abstract methods (leaving the subclass abstract). - Interfaces can extend multiple interfaces containing abstract methods, inheriting their contracts. - Abstract methods participate in polymorphism (the subclass implementation is invoked at runtime).

2. What We Cannot Do With Abstract Methods

- ■ Cannot declare an abstract method in a non-abstract (concrete) class. - ■ Cannot instantiate an abstract class containing abstract methods. - ■ Cannot declare an abstract method with a body/implementation. - ■ Cannot mark an abstract method as static (static methods cannot be overridden). - ■ Cannot mark an abstract method as final (final methods cannot be overridden). - ■ Cannot mark an abstract method as private (private methods are not visible to subclasses). - ■ Cannot declare abstract constructors (constructors are always concrete). - ■ Cannot combine abstract with native, synchronized, or strictfp in method declaration (not allowed by design). - ■ Cannot directly call an abstract method (you must call through an implementation in a subclass). - ■ Cannot have abstract methods inside non-abstract inner classes (but can in abstract inner classes).

3. Special Notes & Edge Cases

- Abstract methods cannot be 'abstract synchronized' or 'abstract native' because they cannot have implementation. - Abstract methods cannot be declared inside enums (enums are concrete). - A subclass that does not provide implementation for all inherited abstract methods must also be declared abstract. - Interfaces implicitly make methods 'public abstract' (until Java 8, where default/static methods were introduced). - From Java 9 onwards, interfaces can have private methods, but they cannot be abstract. - An abstract class can have static methods, but they must be concrete (with implementation).

4. Summary

✔ Abstract methods are for defining 'what to do' without 'how to do it'. ✔ Subclasses are forced to provide implementation, enabling polymorphism. ✔ They cannot be static, final, or private. ✔ They must be inside an abstract class or interface. ✔ They are key in enforcing contracts and designing extensible OOP systems.