1. Abstract Class vs Normal Class:

Feature	Abstract Class	Normal Class
Object Instantiation	Cannot be instantiated directly.	Can be instantiated directly using new.
Method Implementation	Can have both abstract (no body) and concrete methods.	Must have complete method implementations.
Purpose	Used to provide a base for other classes, may contain some common behavior to share.	Defines concrete behavior; used to create actual objects.
Inheritance	Can only be extended (inherited) by subclasses.	Can be extended by other classes, or used directly.
Abstract Methods	Can have abstract methods that subclasses must implement.	Cannot have abstract methods.
Constructor	Can have constructors but can't be instantiated.	Can have constructors and can be instantiated.
Use Case	Used when you need a base class that shares some behavior but must be extended for full implementation.	Used when you need a fully functional class that can create objects.

2. Interface vs Normal Class:

Feature	Interface	Normal Class
Object Instantiation	Cannot be instantiated directly.	Can be instantiated directly using new.
Method Implementation	Cannot have method bodies (before Java 8). After Java 8, can have default and static methods with bodies.	Must have concrete method implementations.
Fields	Can only have public, static, and final fields (constants).	Can have fields of any type, and with any access modifier (private, protected, public).
Inheritance	A class can implement multiple interfaces.	Can be extended by subclasses, but can only extend one class.

Feature	Interface	Normal Class
Abstract Methods	All methods are abstract by default (before Java 8).	Must have concrete method implementations.
Constructor	No constructors, cannot manage state.	Can have constructors, and objects can be created from them.
Access Modifiers	Methods are public by default (before Java 9). Can have private methods since Java 9.	Methods can have any access modifier (public, private, protected).
Use Case	Used to define a contract that classes must implement (without enforcing how they do it).	Defines a concrete class with behavior and state, used to create objects.

3. Abstract Vs Interface

Feature	Abstract Class	Interface
Method Types	Can have both abstract and concrete methods.	Only abstract methods (before Java 8), default, and static methods (since Java 8).
Method Implementation	Can provide method implementations.	No method implementation (before Java 8). Default and static methods can have implementations (since Java 8).
Inheritance	Supports single inheritance; a class can extend only one abstract class.	Supports multiple inheritance; a class can implement multiple interfaces.
Fields	Can have fields of any type and with any access modifier (private, protected, public).	Fields are public, static, and final by default (i.e., constants).
Constructor	Can have constructors to initialize fields.	Cannot have constructors. Interfaces cannot be instantiated.
Access Modifiers for Methods	Methods can have any access modifier (public, protected, private).	Methods are public by default, but private methods are allowed (since Java 9).

Feature	Abstract Class	Interface
State Management	Can maintain state (instance variables).	Cannot maintain state (no instance variables, only constants).
Multiple Inheritance	Cannot extend more than one class.	A class can implement multiple interfaces, achieving multiple inheritance.
Use Case	Used when classes share common behavior, and some methods may have default implementations.	Used to define a contract that multiple classes can implement, often when unrelated classes share behavior.
Performance	Generally faster as method calls are resolved at compile-time (since it can have concrete methods).	Slower as interface methods are resolved at runtime (though this difference is minimal in modern Java).
Keyword for Usage	extends keyword is used to inherit an abstract class.	implements keyword is used to implement an interface.
Abstract vs Interface Comparison (since Java 8/9)	Even though Java 8 added default methods in interfaces, abstract classes are still better for scenarios where shared state or common functionality is needed.	Interface default methods provide a way to add new functionality to interfaces without breaking existing code, but they cannot store or manage stat