ABSTRACT VS INTERFACE (Java) Quick Notes

1) Definitions

- Abstract class: may have abstract + concrete methods; can hold state; can have constructors.
- Interface: contract of behaviors; methods are abstract by default; supports default/static (Java 8+) and private (Java 9+) helpers; fields are constants.

2) Syntax

```
abstract class Vehicle { int wheels; Vehicle(int w){ this.wheels = w; } abstract void move();
void horn(){ } }
interface Payable { void pay(double amt); default void receipt(){ } static boolean valid(){
return true; } }
class Car extends Vehicle implements Payable { Car(){ super(4); } void move(){ /*...*/ } public
 void pay(double a){ } }
```

3) Key Differences

- Inheritance: class extends ONE abstract class; class can implement MULTIPLE interfaces.
- State: abstract class can have instance fields; interface fields are public static final (constants).
- Constructors: abstract class YES; interface NO.
- Access: abstract members can be private/protected/package/public; interface methods are public (private allowed only for helpers).
- Use case: abstract class for shared state + partial impl; interface for capability/role across unrelated classes.

4) When to Use

- Prefer INTERFACE for APIs, plugins, and multiple inheritance of type.
- Use ABSTRACT CLASS when sharing common state/logic, default fields, or protected helpers.

5) Gotchas

- Default method conflict: override and disambiguate with InterfaceName.super.method().
- Avoid state in interfaces (not allowed, only constants).
- You can put default methods in interfaces, but keep them small to avoid tight coupling.