ChatGPT

Java OOP Concepts: When and Why to Use Abstraction, Inheritance, and Extends

---

1. Summary Table: Abstraction vs Inheritance vs Extends

Concept	What is it?	When to Use	Why to Use	Example Keyword
Abstraction	Hiding internal details, exposing only necessary features	When you want to define a blueprint for multiple classes (but cannot make objects of this class)	To force subclasses to implement methods	`abstract` class / `interface`
Inheritance	Acquiring fields & methods of another class	When multiple classes share common properties/behavior	To reduce code duplication, enhance reusability	`extends`
Extends	Keyword used to implement inheritance or extend abstract class	When one class derives from another	To indicate parent-child relationship	`extends`
Implements	Used to implement an interface	When a class must provide implementation of an interface	To achieve full abstraction, multiple inheritances of types	`implements`

---

2. Where to Put Functions?

Class Type	Functions to Put Here	Example
Abstract Class	- Partially implemented methods<br>- Common logic<br>- Constructors<br>- Fields	`abstract void area();` <br> `void display() { }`
Interface	- Only method signatures (no body unless default)<br>- Constants (final static)	`void print();`

Class Type	Functions to Put Here	Example
Base (Parent) Class	- General behavior & properties shared by child classes - Methods that can be overridden or used as-is	`void sound() { }`
Child (Derived) Class	- Specific implementation - Overriding parent/abstract methods - Own unique properties/methods	`@Override void sound() { }`

---

3. Decision Table: How to Choose?

Requirement	Choose	Reason
You want to define a standard but let derived classes decide exact implementation	**Abstract Class / Interface**	Enforces design, no direct object creation
You want to share code and properties across multiple classes	**Inheritance (extends)**	Code reuse, common behavior
You want full abstraction and multiple "type" inheritance	**Interface (implements)**	Java allows multiple interfaces but only single class inheritance
You want to implement optional or partial method logic and common fields	**Abstract Class**	Allows partial implementation with variables
You want to fully override behavior in child class	**Inheritance with extends**	Customize base class behavior

---

4. Example: Correct Placement of Functions

```java
// Interface (Only Signatures)
interface Printable {
    void print();  // No body
}

// Abstract Class (Partial Implementation)
abstract class Document {
    String title;
    Document(String title) {
        this.title = title;
    }
    abstract void open();   // Must be implemented by child
    void displayTitle() {   // Already implemented
        System.out.println("Title: " + title);
    }
}

// Base Class
class Animal {
    void sound() {
        System.out.println("Generic sound");
    }
}

// Child Class
class Dog extends Animal implements Printable {
    @Override
    void sound() {
        System.out.println("Bark");
    }
    public void print() {
        System.out.println("Printing Dog Info");
    }
}
```

---

5. Summary Notes

- Use `abstract` **class** when you want to share code + force some method implementation.
- Use `interface` when you want to define "what should be done" but not "how".
- Use `extends` to derive classes & reuse common logic.
- Use `implements` to guarantee that a class follows an interface.

For implementing OOP designs correctly in Java LeetCode and Projects.