**Core 4 Pillars of OOP**

**1. Encapsulation**

* Bundling of data + methods in a class.
* Achieved using private fields + public getters/setters.

**2. Abstraction**

* Hides complexity shows only essential features.
* Achieved using:
  + abstract classes
  + interfaces

**3. Inheritance**

* Allows one class to inherit fields & methods of another.
* Promotes reusability.

**Types**:

* Single
* Multilevel
* Hierarchical
* Multiple (via interfaces in Java)
* Hybrid (combination of above)

**4. Polymorphism**

* Many forms: method behaves differently depending on context.

**Types**:

* Compile-time → **Overloading**
* Runtime → **Overriding**

**Extended OOP Concepts**

**5. Data Hiding**

* Part of encapsulation.
* Protects internal state by restricting direct access (private, protected).

**6. Generalization**

* Process of extracting shared features from multiple classes and placing them in a common superclass.

Example:

1. class Dog extends Animal {}

2. class Cat extends Animal {}

3. // Animal is the generalized form

**7. Specialization**

* The opposite of generalization.
* Subclass adds or customizes features of a parent class.

**8. Composition**

* A class contains another class object.
* **Strong association**: the contained object **cannot exist independently**.

1. class Engine {}

2. class Car {

3. private Engine engine = new Engine(); // Composition

4. }

**9. Aggregation**

* **Weaker association**: the contained object **can exist independently**.

1. class Engine {}

2. class Car {

3. private Engine engine; // Aggregation

4. Car(Engine engine) { this.engine = engine; }

5. }

6.

**Other Important OOP Concepts**

**10. Association**

* General relationship between two classes.
* Can be one-to-one, one-to-many, etc.

1. class Teacher {

2. Student student; // Associated with

3. }

**11. Dependency**

* One class depends on another temporarily (uses it in a method).

1. class Printer {

2. void print(Document doc) { System.out.println(doc.getText()); }

3. }

4.

**12. Cohesion**

* How strongly-related and focused a class is.
* **High cohesion = better design** (one class = one responsibility).

**13. Coupling**

* Degree of dependency between classes.
* **Low coupling = better design** (classes are independent).

**14. Delegation**

* One object delegates work to another.

1. class Printer {

2. void print() { System.out.println("Printing..."); }

3. }

4.

5. class Manager {

6. private Printer printer = new Printer();

7. void startPrinting() {

8. printer.print(); // Delegation

9. }

10. }

**Bonus: Java-Specific Notes**

* Java doesn’t support **multiple inheritance** with classes, but does support it via **interfaces**.
* Java interfaces can now have **default methods** (since Java 8).
* Java supports **inner classes**, **anonymous classes**, **static classes**, etc.—useful for advanced OOP design.

**Summary Table (Cheat Sheet)**

| **Concept** | **Category** | **Key Idea** |
| --- | --- | --- |
| Encapsulation | Pillar | Data + methods + access control |
| Abstraction | Pillar | Hides complexity (abstract class/interface) |
| Inheritance | Pillar | "Is-a" relationship |
| Polymorphism | Pillar | One method, many forms (overload/override) |
| Generalization | Design Principle | Extract shared features |
| Specialization | Design Principle | Customize behavior from a parent |
| Composition | Relationship | "Has-a", strong ownership |
| Aggregation | Relationship | "Has-a", weak ownership |
| Association | Relationship | General connection between classes |
| Dependency | Relationship | Temporary use of another class |
| Coupling | Principle | Dependency degree between classes |
| Cohesion | Principle | Focus and unity of purpose in a class |
| Delegation | Principle | One object does work on behalf of another |

Would you like this exported as a PDF or a diagram flowchart as well?