Das **Proxy-Pattern** (auf Deutsch: **Stellvertreter-Entwurfsmuster**) ermöglicht es, ein Objekt durch einen **Stellvertreter (Proxy)** zu ersetzen. Dieser Proxy **kontrolliert den Zugriff** auf das echte Objekt, kann den Zugriff überwachen, einschränken, verzögern oder erweitern, bevor er die eigentliche Anfrage an das reale Objekt weiterleitet.

Grundidee des Proxy-Patterns:

Das Proxy-Pattern ermöglicht es, ein **Objekt zu kapseln** und dabei einen Stellvertreter einzuschalten, der das eigentliche Objekt kontrolliert oder verändert, bevor auf das echte Objekt zugegriffen wird. Dies ist besonders nützlich, wenn:

- Der Zugriff auf das Objekt **kontrolliert** oder **eingeschränkt** werden muss.
- Das echte Objekt **ressourcenintensiv** zu erstellen ist und erst dann geladen werden soll, wenn es wirklich benötigt wird (**Lazy Loading**).
- **Sicherheitsüberprüfungen** oder **Logik** vor dem Zugriff auf das Objekt hinzugefügt werden sollen.

Wie funktioniert das Proxy-Pattern?

Das Proxy-Pattern besteht aus drei wichtigen Komponenten:

1. **Subject (Subjekt)**: Dies ist eine **abstrakte Schnittstelle** oder Klasse, die sowohl der Proxy als auch das echte Objekt implementieren. Sie definiert die gemeinsamen Methoden, auf die der Client zugreift.
2. **RealSubject (Echtes Subjekt)**: Das **echte Objekt**, das die eigentliche Arbeit verrichtet. Dieses Objekt wird durch den Proxy geschützt oder erweitert.
3. **Proxy**: Der **Stellvertreter** oder Proxy, der als **Mittelsmann** agiert und die Kontrolle über den Zugriff auf das reale Objekt übernimmt. Der Proxy kann zusätzliche Logik ausführen, bevor er die Anfrage an das echte Objekt weiterleitet.

Beispiel: Bankkonto und ein Schutzproxy:

Stell dir vor, du entwickelst eine Bankanwendung, und du möchtest sicherstellen, dass der Zugriff auf das **Bankkonto** nur mit einer Sicherheitsüberprüfung erfolgt. Anstatt die Sicherheitslogik direkt in die Bankkonto-Klasse einzubauen, kannst du einen **Proxy** verwenden, der den Zugriff auf das Konto **schützt**.

Schritt 1: Definiere das gemeinsame Interface (Subject)

Sowohl der **Proxy** als auch das **echte Bankkonto** müssen ein gemeinsames Interface oder eine abstrakte Klasse implementieren, um sicherzustellen, dass sie dieselbe Methode bereitstellen.

```
interface Bankkonto {
    void einzahlen(double betrag);
    void abheben(double betrag);
}
```

Schritt 2: Erstelle das echte Bankkonto (RealSubject)

Das echte Bankkonto führt die eigentliche Logik aus, wie das Einzahlen und Abheben von Geld.

```
class EchtesBankkonto implements Bankkonto {
    private double saldo;

    @Override
    public void einzahlen(double betrag) {
        saldo += betrag;
        System.out.println(betrag + " € eingezahlt. Neuer Kontostand: " +
    }

    @Override
    public void abheben(double betrag) {
        if (saldo >= betrag) {
            saldo -= betrag;
            System.out.println(betrag + " € abgehoben. Neuer Kontostand: "
        } else {
            System.out.println("Nicht genügend Guthaben.");
        }
    }
}
```

Schritt 3: Erstelle den Proxy
Der Proxy kontrolliert den Zugriff auf das echte Bankkonto, indem er z. B. eine **Sicherheitsüberprüfung** durchführt, bevor er Anfragen an das reale Bankkonto weiterleitet.

```
class BankkontoProxy implements Bankkonto {
    private EchtesBankkonto echtesBankkonto;
    private String zugangscode;

    public BankkontoProxy(String zugangscode) {
        this.echtesBankkonto = new EchtesBankkonto();  // Echtes Konto
        this.zugangscode = zugangscode;                // Zugriffskontroll
    }

    private boolean autorisieren(String eingegebenerCode) {
        return zugangscode.equals(eingegebenerCode);  // Vergleich des Si
    }

    @Override
    public void einzahlen(double betrag) {
        // Sicherheitsprüfung, bevor Zugriff gewährt wird
        System.out.println("Sicherheitsüberprüfung...");
        if (autorisieren("1234")) {  // Beispiel: Der Code ist "1234"
            echtesBankkonto.einzahlen(betrag);
        } else {
            System.out.println("Zugriff verweigert: Ungültiger Code.");
        }
    }

    @Override
    public void abheben(double betrag) {
        System.out.println("Sicherheitsüberprüfung...");
        if (autorisieren("1234")) {
            echtesBankkonto.abheben(betrag);
        } else {
            System.out.println("Zugriff verweigert: Ungültiger Code.");
        }
    }
}
```

Was passiert hier?

- Der **BankkontoProxy** überprüft den Zugangscode, bevor er die Methodenaufrufe (`einzahlen()` und `abheben()`) an das echte Bankkonto weiterleitet.
- Wenn die Sicherheitsüberprüfung erfolgreich ist, wird die Anfrage an das echte Bankkonto weitergeleitet. Wenn nicht, wird der Zugriff verweigert.

Verschiedene Arten von Proxies:

Es gibt verschiedene Arten von **Proxies**, die unterschiedliche Aufgaben haben. Hier sind einige gängige Typen:

1. **Schutz-Proxy (Protection Proxy)**:
    - Wie im obigen Beispiel schützt der Schutz-Proxy das Objekt vor **unerlaubtem Zugriff**. Das kann eine **Sicherheitsüberprüfung** sein, wie in einem Bankkonto-Beispiel.
2. **Virtueller Proxy (Virtual Proxy)**:
    - Der **virtuelle Proxy** wird verwendet, um das **Laden eines ressourcenintensiven Objekts** zu verzögern (auch bekannt als **Lazy Loading**). Das echte Objekt wird erst dann erstellt, wenn es wirklich benötigt wird.

    Beispiel: Ein Bildobjekt, das erst geladen wird, wenn es wirklich angezeigt werden muss.
3. **Remote Proxy (Fernzugriffs-Proxy)**:
    - Der **Remote Proxy** ermöglicht es, auf ein Objekt zuzugreifen, das sich **auf einem entfernten Rechner** (z. B. in einem anderen Netzwerk) befindet. Der Proxy stellt sicher, dass die Kommunikation mit dem entfernten Objekt korrekt erfolgt.
4. **Cache Proxy (Caching Proxy)**:
    - Ein **Cache-Proxy** speichert häufige Anfragen zwischenspeichert und reduziert so die **Ladezeiten** oder **Leistungskosten**. Das echte Objekt wird nur bei Bedarf verwendet, wenn die Daten nicht bereits im Cache vorhanden sind.
5. **Logging Proxy (Protokoll-Proxy)**:
    - Ein **Logging Proxy** wird verwendet, um **Anfragen zu protokollieren**, die an das Objekt gesendet werden. Dies ist besonders nützlich für **Debugging**, **Überwachung** und **Analyse**

Vorteile des Proxy-Patterns:

1. **Kontrolle über den Zugriff**: Das Proxy-Pattern bietet eine Möglichkeit, den Zugriff auf ein Objekt zu kontrollieren (z. B. durch Sicherheitsüberprüfungen).
2. **Lazy Loading**: Mit einem virtuellen Proxy kannst du **ressourcenintensive Objekte erst dann laden**, wenn sie tatsächlich benötigt werden. Dies verbessert die Leistung, insbesondere in Systemen, die mit vielen Ressourcen arbeiten.
3. **Zusätzliche Funktionalität**: Proxies können zusätzliche Logik wie **Caching**, **Logging** oder **Sicherheitsüberprüfungen** hinzufügen, ohne das ursprüngliche Objekt zu verändern.
4. **Vermeidung von Änderungen am Originalcode**: Der Proxy fügt dem Originalobjekt keine zusätzlichen Aufgaben hinzu, sondern arbeitet als separate Komponente, die den Zugriff steuert.

Wann sollte man das Proxy-Pattern verwenden?

- **Zugriffskontrolle**: Wenn du den Zugriff auf ein Objekt überwachen oder kontrollieren musst (z. B. durch Authentifizierung).
- **Lazy Loading**: Wenn du das **Laden eines Objekts verzögern** möchtest, bis es tatsächlich benötigt wird.
- **Remote-Zugriffe**: Wenn du ein Objekt hast, das sich **auf einem entfernten System** befindet, und du sicherstellen möchtest, dass die Kommunikation korrekt gehandhabt wird.
- **Caching**: Wenn du häufige Anfragen effizienter machen möchtest, indem du Ergebnisse zwischenspeicherst.