0) In 10 Sekunden: Compile-Fails zuerst

- Fehlende `;` / `)` / `}` / falsche Klammern
- Tippfehler in Typen/Namen (z.B. `Tocket` vs `Ticket`)

1) Polymorphie-Check (häufigster Punktverlust)

- **Virtueller Destruktor** in Basisklasse (wenn über Base-Pointer/Ref genutzt):
- **`override`** in abgeleiteten Klassen:
- **Slicing vermeiden:** keine Base-by-value Parameter/Returns; stattdessen `Base&`, `Base*`, `std::unique_pt`

2) Copy/Assign korrekt (Rule of 0/5)

- Wenn Copy/Assign **verboten** sein soll:

```
Base(Base const&) = delete;
Base& operator=(Base const&) = delete;
```

- Bei Ownership: **`std::unique_ptr`** bevorzugen (statt roher Pointer).

3) Ownership / Speicher (Factories & Container)

- Factory-Rückgabe (typisch):

```
std::unique_ptr<Base> Create(...);
```

- Container-Speicherung:

```
std::vector<std::unique_ptr<Base>> v;
```

- Nie: `return &local;` oder `new` ohne klaren Besitzer.
- Base-Pointer korrekt zuweisen:

```
Base* p = &derived;   // nicht: Base* p = derived;
```

4) Sichtbarkeit & `const`

- Interface-Methoden: `public`
- Member: fast immer `private`

Pattern-spezifische Mini-Checklisten

Template Method

- Template-Funktion definiert Ablauf, ruft Hooks:

```
void Print() const { header(); DoPrint(); footer(); }
protected: virtual void DoPrint() const = 0;
```

- Hook ist `virtual`, häufig `= 0`.

Factory Method

- Creator-Interface:

  ```
  virtual std::unique_ptr<Product> Create(...) = 0;
  ```

- Konkrete Creator erstellen konkrete Products.
- High-Level nutzt **Creator-Abstraktion** (nicht `new ConcreteProduct` im High-Level).

Strategy

- Strategy-Interface + austauschbar:

  ```
  struct Strategy { virtual ~Strategy()=default; virtual void Run()=0; };
  ```

- Kontext hält `unique_ptr<Strategy>` oder Referenz.

Observer

- `Attach / Detach / Notify`
- Bei `Notify`: Iterator-Invalidierung vermeiden (z.B. Liste kopieren)
- Ownership-Zyklen vermeiden (oft `weak_ptr` bei bidirektional)

Visitor

- `Accept(Visitor&)` in Elementen
- `Visit(Concrete&)` im Visitor
- Kein `dynamic_cast` nötig (Doppeldispatch korrekt)

Decorator

- Decorator erbt von Component und hält Component:

  ```
  std::unique_ptr<Component> inner_;
  ```

- Delegieren + erweitern (Aufruf weiterleiten nicht vergessen)

5) DIP-Schnellcheck (falls gefordert)

- High-Level soll nur **Abstraktionen** kennen, keine konkreten Klassen.
- Lösung: **Dependency Injection** (Ctor/Setter/Parameter) oder Factory.

30-Sekunden Final Scan (immer machen)

- Base hat `virtual ~Base()`?
- `override` überall korrekt?
- Pointer-Zuweisung mit `&`?
- `operator=` liefert `T&`?
- Ownership klar (`unique_ptr`)?
- Keine Base-by-value Übergaben/Returns?