

ChatGPT 5.2 ▾

Gemeinsam nutzen

...

Du bist ein Senior-Java-Engineer. Implementiere ein Java-Swing-Desktopprogramm namens "CodeVibrator" als Maven-Projekt mit Java 21 (LTS). Ziel: Ein lauffähiger MVP, der kompiliert und startet. Halte dich streng an die untenstehenden fachlichen Anforderungen. Wenn etwas unklar ist, triff eine konservative, sichere Entscheidung und dokumentiere sie kurz in einer README im Projekt.

WICHTIG: Keine Automatisierung der ChatGPT-Weboberfläche (kein DOM-Zugriff, kein Scraping). Human-in-the-loop: Prompt wird nur in die Zwischenablage kopiert und ChatGPT im Browser geöffnet.

1) Projekt/Build

- Maven-Projekt (pom.xml), Java 21.
- Erzeuge eine ausführbare App (z.B. maven-shade-plugin oder exec-maven-plugin), sodass mvn package ein startbares Artefakt erzeugt.
- Abhängigkeiten: Swing (JDK), JSON (Jackson oder Gson), optional eine Diff-Library (ansonsten selbst implementieren).
- Plattform: Windows/macOS/Linux.

2) Grund-Workflow / UI (MVP)

- Hauptfenster mit dreigeteiltem SplitPane:
 - A) Links: Directory-Tree, zeigt NUR Verzeichnisse ab Projekt-Root.
 - B) Rechts oben: Selektoren-Kontrollfeld für das ausgewählte Verzeichnis.
 - C) Rechts unten: Dateitabelle (Dateien des ausgewählten Verzeichnisses), sortierbar nach Name und Eigenschaften.
- In der Dateitabelle zeige pro Datei an, ob sie "im Zip" wäre, inkl. Markierung für Konflikte (Warnsymbol), Blockierungen etc. Farben: Neu grün, Geändert gelb, Blockiert rot, Konflikt Warnsymbol.
- Keine Interaktion im Import-Dialog (keine Datei-

Overrides); nur Anzeige + Bestätigen/Abbrechen.

3) Profile / Selektoren / Vererbung (Zip-Input)

- Es gibt beliebig viele Zip-Profile. MVP: mindestens "default" fix vorhanden, optional weitere.
- Zip-Input wird ausschließlich über "Aktiv"-Selektoren bestimmt, außer ein Verzeichnis ist "Exclude from Zip".
- Selektoren gelten pro Ebene nur für die aktuelle Ebene (keine Pfadpattern). Syntax: Semikolon-separierte einfache Globs wie *.java;*.md .
- Vererbung:
 - Jede Ebene hat ein Textfeld "selectorsText". Alle dort genannten Selektoren sind in dieser Ebene immer FORCE und setzen ihren Aktivzustand.
 - Selektoren aus Parent-Ebenen werden in Child-Ebenen "ausmultipliziert" und als einzelne Zeilen angezeigt.
 - Jede ausmultiplizierte Zeile hat zwei Checkboxen: FORCE und AKTIV.
 - Ohne FORCE übernimmt die Zeile den Parent-Aktivwert. Mit FORCE gilt der Child-Aktivwert.
- Konflikterkennung:
 - Wenn eine Datei mehrere Selektoren matcht, deren aufgelöste Aktivzustände widersprüchlich sind, markiere Konflikt (Warnsymbol).
 - Wenn der Nutzer die Warnung hinnimmt, gilt OR-Logik: mindestens ein aktiver matchender Selektor => Datei kommt ins Zip.
 - MVP: Warnung anzeigen, aber keine zusätzliche Interaktion nötig; OR-Logik anwenden.

4) Exclude-from-Zip

- Verzeichnisse können pro Profil als "Exclude from Zip" markiert werden (vererbend nach unten).
- Exclude hat Vorrang vor Selektoren: in excluded Bereichen keine Dateien ins Zip.
- UI: Excluded-Verzeichnisse im Tree markieren.

5) Readonly-Regeln (Importschutz)

- Readonly-Dateien per Pattern (analog zu Selektoren) je Ebene; Readonly-Verzeichnisse per Tree-Knoten

markierbar, vererbend.

- Readonly und Exclude wirken beim Import als Blockierung: niemals überschreiben; nur Warnung im Result-Log.
- Niemals löschen (auch wenn Dateien im Result-Zip fehlen).

6) Persistenz (.code.vibrator)

- Pro Verzeichnis optional eine `.code.vibrator` Datei (wenn dort nichts konfiguriert ist, kann sie fehlen).
- Alle Profile teilen sich in einem Verzeichnis dieselbe `.code.vibrator`.
- Projektweite Daten (`activeProfile`, `promptHistory`, `addOns`) im Projekt-Root in `.code.vibrator`.
- Home-Defaults: im User-Home existiert genau eine Datei `~/.code.vibrator` (keine Unterverzeichnisse). Darin stehen Defaults (Prompt-AddOns und Textfile-Listen).
- Textfile-Erkennung:
 - ausschließlich über Konfig in Home-Defaults: semikolon-separiert
 - * `textFileExtensions`: Endungen MIT führendem Punkt, case-insensitive (z.B. `.java`; `.xml`)
 - * `textFileExactNames`: exakte Dateinamen (z.B.

Makefile;Dockerfile;gradlew;gradlew.bat)

- Datei ist Text, wenn Name in exactNames ODER Endung in extensions. Sonst nicht-Text.

- Wenn Listen fehlen oder leer sind, initialisiere sie mit sinnvollen Default-Werten (Software-Dev typisch) und schreibe `~/.code.vibrator` .

7) Tages-Arbeitsverzeichnis / Logs / Zips

- Pro Tag ein Arbeitsverzeichnis direkt unter dem Projekt-Root: `./YYYY_MM_DD/` (relativ zum Projekt-Root).

- Es bleibt erhalten; keine automatische Aufräumlogik.

- Zeitstempel-Format für Dateien: `yyyy_MM_dd_HH_mm` .

- Export-Zip wird im Tagesverzeichnis abgelegt.

- Result-Zips (vom Nutzer heruntergeladen) sollen ebenfalls dort abgelegt/gesucht werden; MVP: Import-Dialog lässt Zip-Datei wählen; optional "neueste Zip im Tagesverzeichnis vorschlagen".

8) Zip-Struktur-Regeln

- Pfade im Zip sind relativ zum Projekt-Root.

- Ein zusätzlicher Root-Ordner im Zip (z.B. alles liegt unter `projectname/...`) ist fatal: Abbruch vor Schreiboperationen.

- Keine absoluten Pfade, keine ... Traversal. Solche Einträge: fatal.

9) Send to ChatGPT (Human in the Loop)

- Prominenter Button "Send to ChatGPT":

1) Zip erzeugen

2) optional Git add+commit (Default: AN; nach Zyklus wieder AN)

3) finalen Prompt zusammensetzen (Basis + AddOns + JSON-Kontext)

4) Prompt in Zwischenablage kopieren

5) ChatGPT im Browser öffnen/foreground

- Danach Statusdialog:

- Zip erstellt

- Prompt kopiert

- Hinweis: im Browser manuell einfügen (Strg+V/Cmd+V) und Zip hochladen

- Optionale Einstellung "Auto-Paste experimentell"
(Default OFF): versucht OS-abhängig Paste per Robot;
Fehler nicht kritisch.

10) Import / Diff (Working Directory Basis)

- Vergleich gegen aktuelles Working Directory (kein Git-Stand).
- Vor jeder Schreiboperation:
 - Vollständiger Vergleich
 - alle Checks (readonly/exclude, Struktur, Namenskonflikte, executables)
 - bei fatalen Konflikten: Abbruch vor Schreiboperationen
- Dateikategorien:
 - Neu: anlegen
 - Geändert: überschreiben
 - Identisch: nicht überschreiben und altes Dateidatum behalten
 - Readonly/Exclude: nicht überschreiben, Warnung im Result-Log
 - Binär: wie normal
 - Ausführbare Dateien: ignorieren + Warnung
 - Fehlend im Zip: nicht löschen
 - Namenskonflikte (Case): alte Schreibweise behalten + Warnung.
- Änderungserkennung:
 - Textdateien: Zeichenvergleich mit Whitespace-Normalisierung. Whitespace-Definition wie Java-Compiler/JLS; verwende `Character.isWhitespace`. Sequenzen werden zu genau einem Whitespace-Zeichen normalisiert.
 - Nicht-Textdateien: Hashvergleich.
- Diff-Anzeige:
 - Klick auf geänderte Textdatei => Side-by-Side Diff Popup.
 - Kein Diff für Binärdateien.
 - Encoding/BOM unverändert aus dem Zip übernehmen.

11) Git

- Commit nur vor Send.
- Default: eingeschaltet; Checkbox springt nach jedem Zyklus wieder auf AN.

- Git-Commit-Message wird aus dem Prompt abgeleitet; Nutzer kann final editieren (Dialog vor commit).
- Git add -A + commit. Bei Fehler (kein Repo etc.): klare Fehlermeldung, Zyklus abbrechen.

12) Logging

- Nach jedem Zyklus (Import oder Abbruch) ein Result-Log erzeugen, im Tagesverzeichnis speichern, und in der UI anzeigen.
- Log enthält Warnungen, geblockte Dateien, ignorierte Executables, Case-Konflikte, Statistiken (#neu/#geändert/ #identisch/#blocked/#ignored).

13) Deliverables

- Vollständiger Quellcode (Maven).
- README mit:
 - Build/Run
 - Projektstruktur
 - Kurze Erklärung der .code.vibrator JSON-Struktur (was gespeichert wird)
 - Hinweis: Human-in-the-loop, keine Web-Automatisierung
 - Implementiere so, dass es schnell erweiterbar ist (aber ohne Overengineering). Nutze klar getrennte Komponenten für: Persistenz, Zip, Selektorauflösung, Import/Diff, UI.

Bitte gib am Ende:

- Dateibaum des Projekts
- Hinweise, wie man es startet
- und kurze Notizen, welche Teile MVP sind und was als nächstes ausgebaut werden kann.

Nachgedacht für 5s >

Nachgedacht für 9s >

Download: [CodeVibrator Maven-Projekt \(MVP\)](#)

Projekt-Datebaum



```
CodeVibrator/
  README.md
  pom.xml
  src/
    main/
      java/
        dev/
          codevibrator/
            app/
              CodeVibratorApp.java
            config/
              DirectoryConfig.java
              HomeDefaults.java
              ProfileDirConfig.java
              ProjectConfig.java
              SelectorState.java
            git/
              GitService.java
            importer/
              ImportPlan.java
              ImportService.java
              TextClassifier.java
              WhitespaceNormalizer.java
            logging/
              ResultLog.java
              ResultLogService.java
            persist/
              ConfigService.java
              JsonIO.java
            selectors/
              EffectiveSelector.java
              FileDecider.java
              FileDecision.java
              GlobMatcher.java
              ResolvedDirRules.java
              SelectorResolver.java
            ui/
              DiffDialog.java
              DirTreeCellRenderer.java
              DirTreeModel.java
              ImportPlanDialog.java
              MainFrame.java
              PromptComposer.java
              package-info.java
            icons/
              Icons.java
            model/
              FileRow.java
```

```
FileTableModel.java  
SelectorRow.java  
SelectorTableModel.java  
zip/  
ZipExportService.java
```

Starten (Build/Run)

Voraussetzungen: **JDK 21 + Maven 3.9+**

«» Bash



```
cd CodeVibrator  
mvn -q clean package  
java -jar target/codevibrator-0.1.0-shaded.jar
```

Wichtig: Das **Projekt-Root** ist das aktuelle Arbeitsverzeichnis, in dem die App gestartet wird.

MVP-Umfang und was als Nächstes ausbaubar ist

Enthalten im MVP

- Swing-Hauptfenster mit 3-Split:
 - Links: Directory-Tree (nur Verzeichnisse ab Projekt-Root)
 - Rechts oben: Profil/Selektoren/Exclude/ Readonly-Controls + Selektor-Tabelle (FORCE/AKTIV)
 - Rechts unten: Dateitabelle je Verzeichnis, inkl. „in Zip“/Status und Markierungen (Warnsymbol bei Konflikt, Lock bei Blockierung)
- Selektoren (Glob, ; -separiert), Vererbung + FORCE/AKTIV, Konflikterkennung + OR-Logik
- Exclude-from-Zip (vererbend), Tree-Markierung
- Readonly-Verzeichnis + readonlyFilePatterns (vererbend) als Import-Blockierung
- Persistenz:
 - ~/.code.vibrator (Textfile-Listen; wird bei fehlenden/leerem Inhalt initialisiert)
 - Projekt-Root ./code.vibrator (activeProfile, promptBase, addOns, history)
 - pro Verzeichnis optional ./path/.code.vibrator (profiles-map)
- Tagesverzeichnis ./YYYY_MM_DD/, Export-Zips und Logs dort
- „Send to ChatGPT“ (human-in-the-loop):

- Export-Zip erzeugen
- optional Git add/commit (Default AN, wird nach Zyklus wieder AN)
- Prompt zusammensetzen + in Zwischenablage kopieren
- Browser öffnen (keine Web-Automatisierung)
- optional „Auto-Paste“ per Robot (Default OFF, Fehler non-fatal)
- Import:
 - Zip-Strukturchecks (keine absoluten Pfade, kein . . . , konservativer extra-root-folder Check)
 - Plan/Preview-Dialog ohne Overrides (nur Confirm/Cancel)
 - Kategorien: NEW/CHANGED/IDENTICAL/BLOCKED/IGNORED_EXECUTABLE
 - Diff-Anzeige: im Import-Preview per Doppelklick auf CHANGED+Text öffnet Side-by-Side + Delta-Info

Nächste sinnvolle Ausbauschritte

- Import-Diff auch direkt aus der File-Tabelle heraus (nicht nur im Import-Preview)
- Robustere Executable-Erkennung (POSIX-Mode-Bits aus Zip, platform-aware)
- Caching der aufgelösten Regeln pro Verzeichnis (Performance bei großen Projekten)
- UI für Prompt-AddOns/History-Management und Profile-Anlage/Lösung
- Striktere und vollständigere Case-Konfliktbehandlung (der MVP loggt/handhabt das noch nicht vollständig)

Hinweis zu konservativen Entscheidungen (und JSON-Struktur) steht im `README.md` im Projekt. ↵

