Projektplan – OOP mit Java: Monte-Carlo-Simulation

# 1. Zusammenfassung der Anforderungen

- Kontext: Wahlpflichtmodul „OOP mit Java“  
- Kernfunktionalität: Monte-Carlo-Simulation von Finanzprodukten (z. B. ETFs)  
- Dateninput: CSV mit historischen Kursdaten  
- Datenverarbeitung: Bereinigung (Missing Values, Ausreißer) und statistische Analyse  
- Simulationstypen: Sparplan (netto, inflations- und steuerbereinigt) und Auszahlplan  
- GUI: JavaFX (SceneBuilder) mit CSV-Import, Einstellungen und Ergebnisdarstellung  
- Packaging: Image + Installer  
- Dokumentation: Projektbericht und Präsentation  
- Team & Zeit: 6 Studierende, 4 Wochen  
- Prozess: Agil mit wöchentlichen Statusmeetings

# 2. Epics & User Stories (ergänzt & verfeinert)

## Epic 1: Datenmanagement

* US 1.1: CSV-Import mit Datei-Chooser und Fehlermeldung bei falschem Format
* US 1.2: Automatische Auffüllung fehlender Werte (Interpolation)
* US 1.3: Hinweis und Entfernung von Ausreißern
* US 1.4: Benutzer kann Separator auswählen
* US 1.5: Benutzer wählt Spalten für Datum und Schlusskurs
* US 1.6: Benutzer kann entscheiden, ob fehlende Werte interpoliert oder gelöscht werden

## Epic 2: Statistische Analyse

* US 2.1: Anzeige der historischen Rendite-Verteilung
* US 2.2: Anzeige wesentlicher Kennzahlen (Mittelwert, Standardabweichung, Sharpe-Ratio)
* US 2.3: Auswahl eines Analysezeitraums (z. B. 6 Monate, 1 Jahr, 5 Jahre)
* US 2.4: Darstellung der Renditeverteilung als Gauß-Glocke
* US 2.5: Export der Grafiken als Bilddateien

## Epic 3: Monte-Carlo-Simulation

* US 3.1: Starten der Simulation mit Parameter-Dialog (Iterationen, Anlagehorizont, Startkapital)
* US 3.2: Darstellung der Ergebnis-Verteilung als Histogramm und Kennzahlen
* US 3.3: Berechnung und Darstellung von Konfidenzintervallen
* US 3.4: Anzeige von Wahrscheinlichkeiten für Szenarien (z. B. Kapitalverlust bei Entnahme)
* US 3.5: Export der Simulationsergebnisse als Grafik

## Epic 4: Sparplan-Simulation

* US 4.1: Definition regelmäßiger Einzahlungen
* US 4.2: Netto-Ergebnis inflations- und steuerbereinigt
* US 4.3: Chart der Wahrscheinlichkeitsverteilung des Endkapitals

## Epic 5: Auszahlplan-Simulation

* US 5.1: Simulation der Kapitalhaltedauer bei jährlichen Entnahmen
* US 5.2: Darstellung der Nachhaltigkeits-Wahrscheinlichkeiten

## Epic 6: GUI & Packaging

* US 6.1: Übersichtliche Navigation aller Funktionen
* US 6.2: Bereitstellung als installierbares Paket

## Epic 7: Dokumentation & Präsentation

* US 7.1: Automatisierte Screenshots für Bericht
* US 7.2: Präsentation mit Architektur- und Workflow-Diagrammen

# 3. Technische Modulstruktur

- CsvImporter: Datei laden, Separator & Spaltenauswahl, Validierung  
- DataCleaner: Erkennung & Behandlung fehlender Daten (Interpolieren oder Entfernen)  
- StatisticsAnalyzer: Analysekennzahlen, Zeitfilterung, Histogramm mit Gauß-Kurve  
- ChartGenerator: Erstellung & Export von Diagrammen (PNG)  
- MonteCarloEngine: Simulation mit Parameter-Input, Konfidenzintervall-Berechnung  
- ExportService: Export von Charts & Simulationsergebnissen

# 4. Sprint-Plan (4 Wochen)

|  |  |  |  |
| --- | --- | --- | --- |
| Woche | Fokus | Deliverables | Meeting |
| 1 | Kick-off & Design | Pflichtenheft, Architektur, Wireframes | Status-Meeting |
| 2 | Datenmanagement & Analyse | CSV-Import, Cleaning, Statistikmodule | Demo Cleaning |
| 3 | Simulationskern | Monte-Carlo-Engine, Spar-/Auszahlplan-Logik | Demo Simulation |
| 4 | GUI-Integration & Packaging | JavaFX-Oberfläche, Installer, Tests, Dokumentation | Abschl. Demo & Review |

# 5. Nächste Schritte

1. Review und Ergänzung der Epics/User Stories (inkl. Separator, Spaltenwahl, Zeitintervallwahl etc.)  
2. Priorisierung für MVP: CSV-Import, Datenbereinigung, Kennzahlen, einfache Simulation  
3. Sprint 1 Kick-off: Pflichtenheft finalisieren, Architektur skizzieren, Verantwortlichkeiten zuweisen  
4. Definition of Done (DoD) festlegen  
5. Taskboard aufsetzen (Trello, GitHub Projects o. ä.) und Tickets erstellen  
6. Teststrategie & erste Testspezifikationen festlegen

# 6. UML-Klassendiagramm (PlantUML-Code)

@startuml  
class CsvImporter {  
 - separator: String  
 - csvFile: File  
 - hasHeader: boolean  
 + loadFile(path: String): List<String[]>  
 + parseColumns(dateIndex: int, priceIndex: int): List<PriceEntry>  
}  
class PriceEntry {  
 - date: LocalDate  
 - closePrice: double  
}  
class DataCleaner {  
 + interpolateMissingValues(data: List<PriceEntry>): List<PriceEntry>  
 + removeOutliers(data: List<PriceEntry>): List<PriceEntry>  
}  
class StatisticsAnalyzer {  
 + calculateReturns(data: List<PriceEntry>): List<Double>  
 + calculateMean(returns: List<Double>): double  
 + calculateStdDev(returns: List<Double>): double  
 + calculateSharpeRatio(): double  
}  
class MonteCarloEngine {  
 - iterations: int  
 - horizonYears: int  
 - startCapital: double  
 - monthlyContribution: double  
 + runSimulation(returns: List<Double>): List<Double>  
 + simulateWithCashFlows(): List<Double>  
 + calculateConfidenceIntervals(): ConfidenceInterval  
}  
class ConfidenceInterval {  
 - lowerBound: double  
 - upperBound: double  
}  
class ChartGenerator {  
 + generateHistogram(data: List<Double>): Image  
 + generateGaussianCurve(data: List<Double>): Image  
 + exportChart(chart: Image, filename: String): void  
}  
CsvImporter --> PriceEntry : parseColumns()  
DataCleaner --> CsvImporter : verwendet bereinigte Daten  
StatisticsAnalyzer --> DataCleaner : analysiert bereinigte Daten  
MonteCarloEngine --> StatisticsAnalyzer : nutzt Renditen  
MonteCarloEngine --> ConfidenceInterval : gibt Intervall aus  
ChartGenerator --> MonteCarloEngine : visualisiert Ergebnisse  
ChartGenerator --> List<Double> : visualisiert Datenreihen  
@enduml