Aufgabe:
In einzelnen Schritten soll eine rudimentäre Mitarbeiterverwaltung aufgebaut werden.

Schritt 1:
Erstellen Sie eine Basisversion der Mitarbeiterklasse.
Der Mitarbeiter sollte folgende Eigenschaften haben:

- Personalnummer
- Vorname
- Nachname
- Geburtsdatum
- Einstellungsdatum
- Gehalt

Wählen Sie geeignete Datentypen und realisieren Sie einen oder auch mehrere entsprechende Konstruktoren, sowie get/set-Pärchen

Schritt 2:
Ergänzen Sie die Lösung aus Schritt 1 um ein Enum für das Geschlecht sowie Standardmethoden (equals, hashcode, toString)

Schritt 3:
Ergänzen Sie die Lösung um einen rudimentären JUnit-Test statt main-Methode.

Schritt 4 Vererbung:
Ergänzen Sie die Lösung um eine Arbeiterklasse. Das Gehalt des Arbeiters setzt sich aus dem Produkt von Stundenlohn und Stundensatz zusammen.
Ist Mitarbeiter eine geeignete Elternklasse für Arbeiter?
Wenn ja, realisieren Sie diese Vererbungsbeziehung, ansonsten setzen Sie eine alternative Vererbungsbeziehung um. (Sicherlich haben Arbeiter und Mitarbeiter einige Gemeinsamkeiten!)
Selbstverständlich sollte anschließend das Abfragen und Setzen des Gehalts für den Arbeiter sauber funktionieren.

Schritt 5 Collections:
Ergänzen Sie die bisherige Lösung um eine Verwaltungsklasse, die 100 Mitarbeiter, jeweils zur Hälfte Mitarbeiter und Arbeiter im Speicher hält. Für spätere Zwecke sollen die Attribute der Mitarbeiter bzw. Arbeiter wertemäßig möglichst variieren. (Tipp: Random-Generator aus der Klassenbibliothek nutzen und Zähler der Füllschleife)
Als Datenstruktur kann wahlweise ein Array oder eine ArrayList verwendet werden.

Prüfen Sie mittels main-Methode, ob das Ganze funktioniert.

Schritt 6 Abstrakte Basisklasse:
Ergänzen Sie die bisherige Verwaltungsklasse um eine Methode, die die Gehaltssumme aller Mitarbeiter liefert.
Dazu sollte eine abstrakte Basisklasse die getGehalt-Methode vorschreiben!

Prüfen Sie mittels main-Methode, ob das Ganze funktioniert.

Schritt 7 Strategy Pattern:
Verwenden Sie ein Interface Gehaltsmodell, welches die getGehalt-Methode vorschreibt und machen Sie aus der Vererbungsbeziehung der Mitarbeiter eine Has-A-Beziehung, sprich der Mitarbeiter hat ein Gehaltsmodell.
Vom Gehaltsmodell kann es jetzt verschiedene Varianten für konkrete Gehaltsmodelle wie den Arbeiter geben.

Prüfen Sie mittels main-Methode, ob das Ganze funktioniert.

Schritt 8 Interfaces und Lambdas:

- Mitarbeiter sollen nach ihrem Gehalt sortierbar sein (natürliche Ordnung via Comparable)
- Optional: Ergänzen Sie die Mitarbeiterverwaltung um eine Methode, die ein Sortierkriterium (Comparator) als Baustein erwartet und eine entsprechend sortierte Liste der Miarbeiter liefert.
  Verwenden Sie nach Möglichkeit Lambda-Schreibweisen für die Sortierkriterien.
- Optional:Stellen Sie die Mitarbeiterverwaltung auf eine Map um und ergänzen Sie die Methodik, um einzelne Mitarbeiter zu bearbeiten.
- Optional: Factory für Gehaltsmodelle

Prüfen Sie mittels main-Methode, ob das Ganze funktioniert.

Schritt 9 GUI via JTable:
Ergänzen Sie die Lösung um eine minimale Swing-GUI, die via JTable die Mitarbeiter anzeigt.
Tipp: Erstellen Sie ein entsprechendes TableModel.
Starten Sie das Ganze mittels main-Methode.

Schritt 10 DB-Zugriff via JDBC:
Ergänzen Sie die Lösung um einen rudimentären Datenbankzugriff via JDBC, um die Mitarbeiterdaten aus der vorhanden Datenbank (Tabelle employees) einzulesen und darzustellen. Jeweils nicht vorhandene Attribute bei den Spalten/Klassen können ignoriert werden.