Ausgangspunkt ist die Modellierung eines Mitarbeiters als Mitarbeiterklasse mit Datenattribut Gehaltsmodell(Has-A-Beziehung) und Gehaltsmodelle setzen das Interface Gehaltsmodell um. Kann natürlich zukünftig flexibel um weitere Gehaltsmodelle ergänzt werden.

Aufgaben (Es sollte natürlich immer ein Aufrufer (main-Methode oder optional JUnit-Test) geschrieben werden.

1) Ergänze den vorliegenden Code um vernünftige Implementierungen der Standard-Methoden der Klasse Object
(toString(), equals() und hashcode()).

2) Ergänze den vorliegenden Code um eine Mitarbeiterverwaltungs-klasse, die intern ein Array von 100 Mitarbeitern hält. Beim Instantiieren dieser Klasse wird das Array intern zur Hälfte mit Arbeitern (Mitarbeiter mit Gehaltsmodell Arbeiter) und Fixgehalt-Mitarbeitern (Mitarbeiter mit Gehaltsmodell Fixgehalt)befüllt. D.h. die Klasse bietet einen parameterlosen Konstruktor an, in dem die eigentliche Arbeit passiert.

Die Gehaltswerte, Datumsangaben (Einstellungsdatum und/oder Geburtsdatum) sollten variieren
(Hinweis: java.lang.Math.random() oder java.util.Random-Klasse) ebenso wie die Namen (z.B. Laufindex der Füllschleife anfügen).

Die Klasse sollte zumindest eine Methode anbieten, die die Gehaltssumme aller Mitarbeiter liefert und
eine Methode, die das gefüllte Mitarbeiter-Array liefert.

Optional kann diese Klasse natürlich auch um Implementierungen der Standard-Methoden aus der Klasse Object
erweitert werden.

3) Ergänze den vorhanden Code um mimimales Exception-Handling: Zum Beispiel sollte bei Übergabe eines nichtakzeptablen Nachnamens (leer oder null) für Mitarbeiter
eine Exception geworfen werden. Idealerweise in Form einer eigenen Exceptionklasse.

4) Ergänzung um eine Factory-Klasse, die Gehaltsmodelle mit default-Werten (frei festlegbar) liefert.

5) Umsetzen der natürlichen Ordnung in der Mitarbeiterklasse anhand des Gehaltswertes (Implementierung des Interfaces Comparable) und mal Sortierung des von der Mitarbeiterverwaltung gelieferten Arrays.

6) Ergänzung um weitere externe Sortierkriterien (Comparator-Interface), z.B. nach Namen und Datumswerten und entsprechender Sortierung des von der Mitarbeiterverwaltung gelieferten Arrays.

7) Umstellung der Mitarbeiterverwaltung auf interne Nutzung einer Map<String, Mitarbeiter> statt des Arrays.
Ergänzung dieser Klasse mit Methoden um Mitarbeiter zu entfernen bzw. neu aufzunehmen.
Ergänzung um eine Methode, die das Sortierkriterium als Lambda erwartet und eine entsprechend sortierte Liste von Mitarbeitern zurückgibt.

8) Ergänzung der Factory-Klasse, um die default-Werte aus einer Properties-Datei zu lesen.

9) Ergänzung der Mitarbeiterverwaltung um Methoden zum Schreiben der Mitarbeiterliste im Textformat in eine Datei, sowie per ObjektSerialisierung zum Speichern und Wiedereinlesen der Liste.

10) Ergänzung um eine kleine GUI (Swing), so dass die Mitarbeiter mit Hilfe von JTable in Form einer Tabelle grafisch dargestellt werden.

11) Umstellung der Mitarbeiterverwaltung auf Datenbankzugriff: Mitarbeiter werden aus der Datenbank eingelesen statt mit Testwerten zu arbeiten.