PROO/SEW III - Test: Wetterfee

Ziel

Implementierung einer Lösung zur erweiterbaren Verwaltung von Wetterstationen und -Sensoren.

Lernziele

- Einsatz des *Observer*-Entwurfsmusters
- Einsatz des *Abstract Factory*-Entwurfsmusters
- Verwenden verschiedener *Java-Collections*
- Schreiben von *Log*-Einträgen

Materialien

- *IntelliJ*

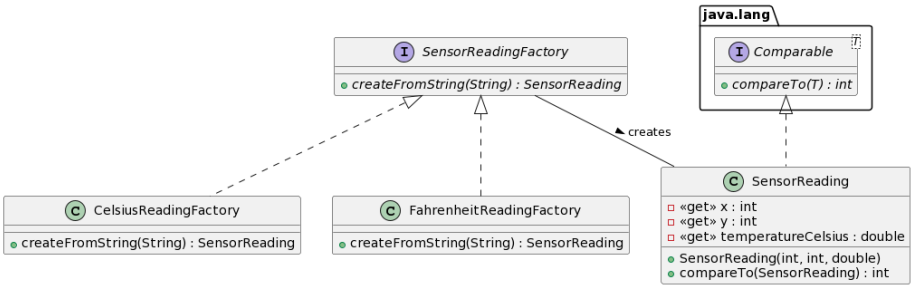
Abgaberichtlinien

- Die fertige *IntelliJ*-Lösung ist in einem Ordner mit ihrem **Nachnamen (ohne Sonderzeichen)** auf dem `U:`-Laufwerk abzugeben.

Aufgabe

Nach einem mehrstufigen Bewerbungsverfahren, in dem alle Ihre Talente auf die Probe gestellt wurden, dürfen Sie endlich Ihren Traumjob im *ORF Wetter*-Team antreten. Ihre erste Aufgabe: Die Überarbeitung der Software-Lösung zur Verwaltung von Wetterstationen und -Sensoren. Nachdem die bisherige Lösung nur wenig erweiterbar war, beschließen Sie *Wetterfee* Christa Kummer mit Ihrem umfangreichen Wissen von Entwicklungsmustern zu beeindrucken und wollen dabei mit dem *Abstract Factory*- und dem *Observer-Pattern* punkten.

Teil I: *Abstract Factory*



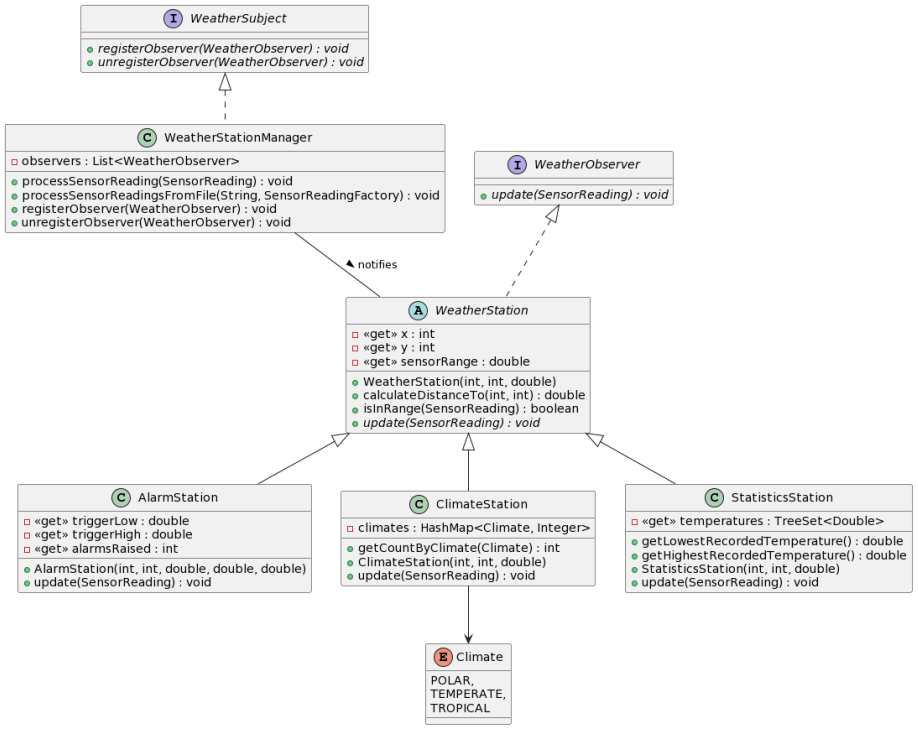
1. Die von Wettersensoren verschickten Daten werden durch die Klasse `SensorReading` dargestellt.

- Jeder Sensorwert besteht dabei aus X und Y Koordinaten und der gemessenen Temperatur in Celsius.
- Die im Konstruktor angegebene Temperatur darf nicht **unter** den *absoluten Nullpunkt* $-273,15$ sinken. Ansonsten ist eine `IllegalArgumentException` zu werfen.
- Sensorwerte können anhand der Temperatur *verglichen* werden.

2. Nachdem Sensordaten in verschiedenen Formaten gesendet werden, wird die Schnittstelle *SensorReadingFactory* bereitgestellt um einfache Erweiterbarkeit zu gewährleisten.
   - Alle konkreten Implementierung sollen eine `WeatherException` werfen, wenn die verarbeiteten Sensordaten nicht korrekt formatiert (e.g. falsche Länge) sind.
   - Die konkrete Implementierung `CelsiusReadingFactory` verarbeitet Sensordaten in Celsius. Hier sind keine weiteren Vorverarbeitungsschritte notwendig - außer natürlich das Entfernen von Leerzeichen, die sich vielleicht eingeschlichen haben.
   - Manche Sensorgeräte liefern ihre Daten in Fahrenheit, obwohl wir sämtliche Daten in Celsius speichern wollen. Dafür wird in der `FahrenheitReadingFactory` die folgende Formel zur Umrechnung von Fahrenheit (T_F) in Celsius (T_C) angewandt. **Runden Sie anschließend das Ergebnis mittels `Math.round()`.**

$$T_C = \frac{5}{9} * (T_F - 32)$$

Teil II: *Observer*



1. Die Wetterstationen, die Sensordaten empfangen, werden in der Klasse `WeatherStationManager` verwaltet.
   - Hierbei handelt es sich um das klassische *Subject* des *Observer*-Musters - i.e. eine Liste von Beobachtern und Methoden zum Anmelden/Abmelden.
   - Allen Sensordaten die empfangen werden (i.e. in `processSensorReading` und `processSensorReadingsFromFile`) werden einfach an die *Observer* weitergegeben. Diese entscheiden selbstständig, ob die Daten für sie relevant sind.
   - Treten beim Laden einer Datei Fehler auf, so ist die `Exception` in eine `WeatherException` zu *verpacken*.
2. Die Standard-Funktionalität von Wetterstationen wird in der abstrakten Klasse `WeatherStation` bereitgestellt:
   - Eine Wetterstation hat ebenfalls X und Y Koordinaten sowie der nicht-negativen Reichweite in der sie empfangen können.
   - Mithilfe der `isInRange`-Methode wird überprüft ob übergebene Sensordaten sich in Reichweite befinden.
   - Hierfür ist die Methode `calculateDistanceTo` zu verwenden, die die Distanz zwischen zwei Punkten anhand des einfachen *Euklidischen Abstands* (s.u.) berechnet. Die Methoden `Math.sqrt` und `Math.pow` (zweiter Parameter bestimmt den Exponenten) könnten hilfreich sein.

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

Auf Basis von `WeatherStation` sind nun 3 konkrete *Observer* zu implementieren. Alle Wetterstationen bearbeiten natürlich **nur Sensordaten, die innerhalb ihrer jeweiligen Reichweite** sind.

1. Die `AlarmStation` wird im Konstruktor mit einer Unter- und Obergrenze konfiguriert. Werden diese Temperaturen über- bzw. unterschritten, so wird ein Zähler (`alarmsRaised`) erhöht.
2. Die `ClimateStation` verwendet eine `HashMap` um die Vorkommen von bestimmten Temperatur-Bereichen je nach Klima (siehe Aufzählung `Climate`) mitzuzählen.
   - Temperaturen *kleiner gleich* 0.0 erhöhen den `POLAR`-Zähler.
   - Temperaturen *größer gleich* 25.0 erhöhen den `TROPICAL`-Zähler.
   - *Ansonsten* ist der Zähler für `TEMPERATE` (auf Deutsch *gemäßigt*) zu erhöhen.
   - Anmerkung: Sie können Werte in einer *Map* einfach durch Überschreiben erhöhen.
3. Die `StatisticsStation` verwendet ein `TreeSet` um alle eindeutig vorkommenden Temperaturen sortiert abzuspeichern.
   - Die vom *ORF* gewünschte höchste/tiefste Temperatur können Sie einfach durch Verwendung von `first/last` abfragen.
   - Nachdem das `Set` öffentlich abgerufen werden kann, müssen Sie es unbedingt vor dem Zurückgeben `clone()`n, damit die originalen Werte nicht verändert werden können!

Bonus: *Logging*

Verwenden Sie *log4j2* um das Laden und Verarbeiten von Dateien zu protokollieren. Ein minimalistisches Beispiel für die `FahrenheitReadingFactory` befindet sich in der Vorlage.

Christa Kummer wünscht gutes Gelingen!