Am Ende des Blatts finden Sie wieder eine Alternativ-/Minimalaufgabe. **Gesamtpunkte: 17**

Vorbereitung

Als Basis für dieses Arbeitsblatt sollten Sie eine Lösung zum vorigen Blatt, sowie das Observer-Pattern verstehen. **Punkte: 3**

Observer verstehen

In der Vorlesung zu KW49 wurde Ihnen das Observer-Pattern vorgestellt. Erweitern Sie Ihr Verständnis dieses Softwaredesigns mit Hilfe des entsprechenden Kapitels aus *Goll 2013*[1] (online: `http://dx.doi.org/10.1007/978-3-658-05532-5`).

Codebasis verstehen

Im Ilias finden Sie eine mögliche Implementierung zum vorigen Blatt, inkl. UML.

- Erstellen Sie ein Projekt aus diesem Code.
- Wie wird die Positionierung und Größe des Plots definiert?
- Machen Sie sich klar, wie die Datenverarbeitung in dieser Implementierung funktioniert. Sie sollten den logischen Ablauf eines `draw`-Zyklus erklären können. Was bedeutet *Polling* in diesem Zusammenhang (siehe Buchkapitel)?
- Warum aggregiert `Plot` weder `Sine` noch `RandomProvider`?
- Was ist der *statische* Typ der in `PlotMain` verwalteten `Plot`-Instanz?

Push Datenverarbeitung: Observer-Pattern

Wir wollen ein Observer-Pattern implementieren, so das kein Polling mehr notwendig ist (s.o.). **Punkte: 5**

- Schreiben Sie das `Provider`-Interface um. Da wir nicht mehr pollen wollen, sind die Methoden `hasData` und `nextDataPoint` unnötig. Ein `Provider` soll einem `Beobachtbar` entsprechen (siehe Buchkapitel).
- Ergänzen Sie ein `Receiver`-Interface. Ein `Receiver` soll einem `Beobachter` entsprechen.
- Beide Datengeneratoren (`Sine` und `RandomProvider`) sollen das `Provider`-Interface implementieren. Da die Registrierungsmethoden (an-/abmelden) hierbei identisch sind, sollten Sie dafür möglichst eine abstrakte Basisklasse definieren. Definieren Sie eine abstrakte Methode `update`, sowie eine implementierte Methode `publish` (siehe Diagramm unten).
- `publish` soll den übergebenen Datenpunkt an alle registrierten `Receiver` verteilen.

[1] Goll und Dausemann: Architektur und Entwurfsmuster der Softwaretechnik, Kapitel 4.11

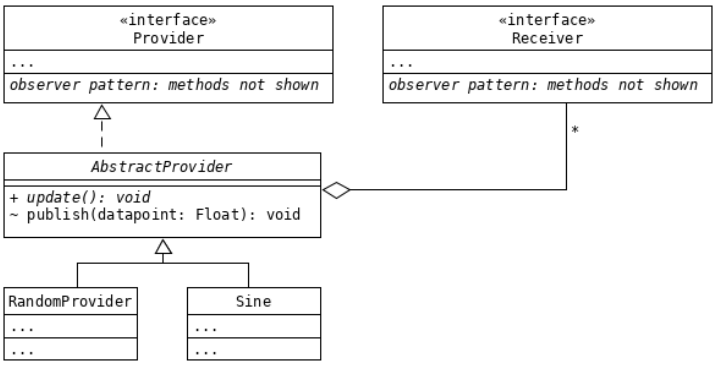

Abbildung 1: Auszug UML zum Observer

Push Datenverarbeitung: Inbetriebnahme

Passen Sie den Code an das implementierte Observermuster an, so das Ihr Programm wieder funktioniert. **Punkte: 4**

- Sie benötigen in `Plot` kein `Provider`-Attribut mehr (warum?).
- Implementieren Sie `Receiver` mit einer anonymen Klasse innerhalb von `PlotMain`. Den über die Interface-Methode erhaltenen Datenpunkt müssen Sie an die Liste `screenValues` in `Plot` übergeben. Entscheiden Sie selbst, wie.
- Leiten Sie `Sine` und `RandomProvider` von Ihrer abstrakten Basisklasse ab – damit implementieren Sie dort implizit das `Provider`-Interface.

Ergänzung durch ein "Display"

Erstellen Sie ein GUI-Objekt, das den aktuellsten Plotwert anzeigt. **Punkte: 5**

- Erstellen Sie eine Klasse `Display` und implementieren dort `Drawable`.
- Mit der Processing-Methode
  `text(txt: String, x: float, y: float, width: float, height: float)`
  können Sie Text in einer Box definierter Dimension an Position x/y anzeigen.
- Positionieren Sie das `Display` neben dem `Plot` (siehe `PlotMain`).
- Aktualisieren Sie das Display mit jedem neuen dargestellten Datenpunkt.

Alternative Aufgabe: Minimales Beispiel

Diese Aufgabe ist eine Alternative oder Ergänzung zu den obigen. Sie können damit eine Mindestpunktzahl erreichen, oder sie bspw. als Ergänzung zu Ihrem Softwareprojekt betrachten – um auf dem Laufenden zu bleiben. Sie können *nicht* mehr als die oben angegebenen Gesamtpunkte erreichen.

Befassen Sie sich mit der Definition des Observer-Patterns und erstellen ein Minimalbeispiel, das seine Verwendung demonstriert. **Punkte: 6**

Anforderungen:

- Code vorhanden und ausführbar mit sinnvollen Textausgaben,
- UML vorhanden und korrekt,
- Sie können die Funktionsweise und den Zweck des Observer-Pattern erklären,
- Sie können an Ihrer Implementierung / Ihrem Klassendiagramm die Komponenten des Pattern zeigen und erklären,
- Ihr Beispiel enthält mindestens zwei unterschiedliche Beobachter.