Unterschied klassisches C Programm & Java Swing

- C: EVA-Prinzip (Eingabe-Verarbeitung-Ausgabe)
- Java: Event-Listener-Prinzip (eventgesteuert)
  - Events werden von verschiedenen Ereignissen ausgelöst (Mausklick, Mausbewegung, Tastatureingaben, Klicken von GUI-Elementen, ...)
  - Listener-Interfaces warten, bis sie von einem Event ausgelöst werden. So kann gezielt auf ein gewünschtes Event reagiert werden

Java Swing

Rahmenfenster (JFrame):

- Grundlage für die meisten GUIs
- Möglichkeit zum Maximieren und Minimieren

Dialogfenster (JDialog):

- nur einfache/wenige Bedienelemente
- abhängig von einem anderen Fenster
- modal-Flag
  - true → Nutzer kann nur im Dialog arbeiten bis dieser geschlossen wird (Hauptfenster wird angehalten, bis der Dialog geschlossen wird)
  - false → Nutzer kann gleichzeitig im Dialog und im Hauptfenster arbeiten (Hauptfenster läuft weiter)
- Änderungen dürfen erst übernommen werden, wenn auf „OK" gedrückt wurde
- „Abbrechen" muss alle im Dialog vorgenommenen Änderungen verwerfen

GUI-Elemente (Controls):

- JList
  - anzeigen einer Liste (nur eine Spalte)
  - Model von der Klasse *AbstractListModel* ableiten
  - MVC-Pattern (Model-View-Controller)
- JTable
  - anzeigen einer Tabelle (mehrere Spalten)
  - Model von der Klasse *AbstractTableModel* ableiten
  - MVC-Pattern (Model-View-Controller)
- JButton
  - Führt eine durch den Text des Buttons beschriebene Aktion durch
- JLabel
  - zeigt einen beschreibenden Text an
- JTextField
  - Eingabefeld für Text
- JRadioButton
  - Möglichkeit zur Auswahl aus einer Liste von Möglichkeiten
- JSlider
  - Schieberegler mit maximalem und minimalem Wert
- JSpinner
  - Auswahl von Objekten in geordneter Reihenfolge (meistens Zahlen)

Container:

- z.B. JPanel
- enthalten weitere GUI-Elemente
- Layoutmanager können zugewiesen und somit geschachtelt werden

Menü:

- JMenubar: Leiste, die alle Menüs enthält
  - Kann nur einmal in der obersten Ebene des JFrames platziert werden
- JMenu: Menü
  - Kann Menü-Punkte und weitere Menüs enthalten
- JMenuItem: Menü-Punkt
  - Selbe Funktion wie JButton
  - Icons und Tastenkürzel können festgelegt werden

Layoutmanager:

Werden zum Positionieren von Controls in einem Layout verwendet, damit nicht jedes Element absolut positioniert werden muss. Abhängig vom Layoutmanager können Größe, Position und einige weitere Anpassungen vorgenommen werden.

- FlowLayout:
  - Alle Elemente werden in ihrer natürlichen Größe dargestellt (z.B.: ein JLabel ist so lang wie der enthaltene Text)
  - Elemente werden in Zeilen zentriert angeordnet. Wenn eine Zeile voll ist, wird eine neue angefangen.
- BorderLayout:
  - Untergliederung in fünf Bereiche:
    - North (Anfang des Fensters)
    - South (Ende des Fensters)
    - West (Anfang der Zeile)
    - East (Ende der Zeile)
    - Center (übriger Platz in der Mitte des Fensters)
- GridLayout:
  - Die Elemente werden in einem Gitter angeordnet (Anzahl der Zeilen und Spalten und deren Abstand kann festgelegt werden)
  - Alle Zellen sind gleich groß
- GridBagLayout:
  - Erweiterung des GridLayouts
    - Zeilen und Spalten können unterschiedlich breit bzw. hoch sein
    - Elemente können sich über mehrere Zellen erstrecken
    - ...

Multitasking mit SwingWorker

Standartmäßig läuft ein Java Swing Programm nur in einem Thread. Dieser heißt EDT (Event Dispatch Thread). Darin laufen alle Methoden sequenziell ab, was den Nachteil hat, dass sich große Applikationen langsamer aktualisieren. Das kann sich durch Ruckeln bemerkbar und das Programm für

den Benutzer unbrauchbar machen. Aus diesem Grund müssen alle rechenintensiven Vorgänge in einen extra Thread ausgelagert werden. Dabei handelt es sich zum Beispiel um:

- aufwendige Berechnungen
- Schreiben und Lesen von Dateien
- Netzwerkkommunikation (Timeout)

In Java Swing wird für Multitasking die abstrakte Klasse *SwingWorker* verwendet. Diese erlaubt, unter anderem, das Zurückgeben von Daten aus dem Nebenthread in den EDT, was bei der Verwendung der Klasse *Thread* nicht möglich ist. Folgende „Komfort-Methoden" werden geboten:

- *doInBackground()*
  Diese abstrakte Methode führt ihren Inhalt in einem Nebenthread aus.
- *done()*
  Nachdem *doInBackground()* fertig ist, wird diese Methode aufgerufen. Diese läuft bereits wieder im EDT und wird meist dazu verwendet, um die GUI mit den berechneten Werten zu aktualisieren oder um den Benutzer über das Ergebnis eines Vorgangs (z.B. Schreiben einer Datei) zu informieren.
- *get()*
  Um in *done()* auf den Rückgabewert von *doInBackground()* zugreifen zu können, muss dort diese Methode aufgerufen werden. Sie hat denselben Rückgabewert wie *doInBackground()*.
- *publish()*
  Eine andere Möglichkeit dem EDT Werte zu übergeben. Dabei handelt es sich um Zwischenergebnisse, die während der Laufzeit von *doInBackground()* entstehen.
- *process()*
  Diese Methode verarbeitet die Zwischenergebnisse, welche sie als eine Liste von Objekten übergeben bekommt. Die Liste ist notwendig, da, wenn sehr viele Zwischenergebnisse vorliegen, mehrere auf einmal übergeben werden. (chunks) Um alle verarbeiten zu können, wird normalerweise ein Zählschleife verwendet.
- *isCancelled()*
  Diese Methode liefert TRUE, wenn der Worker vor dem Beenden von *doInBackground()* abgebrochen wurde.

Da es sich beim *SwingWorker* um eine abstrakte Klasse handelt, kann dieser nur verwendet werden, wenn eine Klasse erstellt wird, die von ihm abgeleitet ist. Eine übersichtliche Lösung ist das Erstellen einer Klasse, in der nur *doInBackground()* überschrieben wird. *done()* und der Rest werden dann in einer inneren Klasse überschrieben. So wird ein Teil des Codes ausgelagert, was die GUI-Klasse etwas übersichtlicher macht.

Um eine Worker-Klasse zu starten, ruft man die Methode *execute()* aus. Um den Prozess abzubrechen wird *cancel(true)* verwendet. Diese Methode bekommt einen Boolean-Wert übergeben, der festlegt, ob ein laufender Thread unterbrochen werden darf. Um sicherzustellen, dass der Vorgang richtig abgebrochen wird, muss regelmäßig überprüft werden, ob dieser abgebrochen wurde (mit *isCancelled()*). Dies sollte am Besten bei jedem Schleifendurchlauf geschehen.

Designpatterns

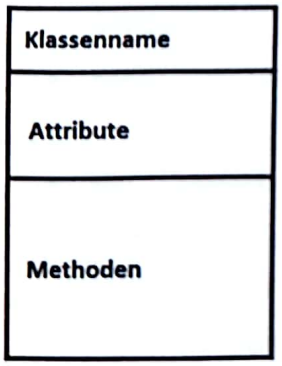
Entwurfsmuster die sich über die sich für immer wiederkehrende Probleme bewährt haben.

- Immuteable → unveränderlich

- auf eine Klasse kann nur lesend zugegriffen werden
- Daten werden im Konstruktor übergeben
- Attribute sind *private* und *final*. Zugriffe nur über getter-Methoden möglich

- Singleton
  - von einer Klasse kann nur eine Instanz existieren
  - *private* Konstruktor
  - getInstance-Methode → wenn Instanz gleich NULL dann erzeuge neue Instanz → gebe Instanz zurück
- Delegation
  - Schnittstelle zu einer Klasse
  - Nutzer oder Programm kann nur darüber auf jene Klasse zugreifen
  - Verwendung z.B. bei einer Datenhaltungsklasse einer Liste von Objekten
    - Klasse beinhaltet eine Liste (ArrayList, LinkedList, ...)
    - auf die Liste kann nun nur über die Methoden der Delegation-Klasse zugegriffen werden
    - Diese Vorgehensweise bietet z.B. den Vorteil, dass man die Klasse die zum Speichern der Daten verwendet wird, geändert werden kann (z.B.: von *ArrayList* auf *LinkedList*), ohne diese im gesamten Code ändern zu müssen
- Rechnerklasse
  - bekommt die Werte der Attribute im Konstruktor übergeben
  - Konstruktor ruft die Methode *calc()* auf
  - *calc()* berechnet nun die gewünschten Werte
  - Die berechneten Werte werden über getter-Methoden ausgelesen

UML-Klassendiagramm



Sichtbarkeit:
+ public
- private
protected
~ package private

Unterschied Binär- und Textdateien

Eine Binärdatei kann jede Kombination aus 1en und 0en enthalten. Sie muss keinem bestimmten Muster folgen. Der Inhalt einer Textdatei ist als Text codiert. UTF-8 ist dabei am weitesten verbreitet.

Streams

InputStream/OutputStream

- Abstrakte Klasse → read/write müssen überschrieben werden
- read/write → Byteweises Lesen/Schreiben
- Verwendung bei Netzwerkkommunikation und Dateien

Reader/Writer

- Abstrakte Klasse → Schreiben/Lesen von Text (Character)-Streams
- BufferedReader/BufferedWriter
  - Die zu lesenden bzw. schreibenden Daten werden in einem Buffer zwischengespeichert. Dadurch werden das Lesen bzw. Schreiben von einzelnen Zeichen, Arrays und Strings effizienter. (z.B.: readLine())
  - newLine() → neue Zeile wird begonnen. Die Verwendung dieser Methode ist besser als das direkte Schreiben des Charakters ‚\n', da dieser nicht auf allen Plattformen zum Beenden einer Zeile verwendet wird.
- FileReader/FileWriter
  - Komfort-Klasse zum Lesen bzw. Schreiben von Dateien
- Brückenklassen
  - InputStreamReader: ByteStreams → TextStreams
  - OutputStreamWriter: TextStreams → ByteStreams
  - Für eine höhere Effizienz sollten diese Klassen mit einem BufferedReader bzw. BufferedWriter verwendet werden:

```
Writer out =
new BufferedWriter(new OutputStreamWriter(System.out));
```

Dateiformate

CSV

- Comma (oder Character) Separated Values
- Datensätze (Zeilen) werden durch einen Zeilenumbruch getrennt
- Datenfelder (Spalten) werden durch einen Character (meist Komma oder Strichpunkt) getrennt
- Kann z.B. mit Excel in eine Tabelle umgewandelt werden.

JSON

- Hierarchische Datenablegung → Daten können „geschachtelt" werden
- Objekte
  - beginnt mit { und endet mit }. Es enthält eine durch Kommas geteilte, ungeordnete Liste von Eigenschaften. Objekte ohne Eigenschaften („leere Objekte") sind zulässig.
- Array
  - beginnt mit [und endet mit]. Es enthält eine durch Kommas geteilte, geordnete Liste von Werten gleichen oder verschiedenen Typs. Leere Arrays sind zulässig.
- Eigenschaft
  - besteht aus einem Schlüssel und einem Wert, getrennt durch einen Doppelpunkt (Schlüssel:Wert)

XML

- Hierarchische Datenablegung → Daten können „geschachtelt" werden
- Elemente

- Start-Tag `<Tagname>` und End-Tag `</Tagname>`
- Leer-Tag `<Tagname/>`
- Kann weitere Elemente enthalten

- Attribute
  - zusätzliche Eigenschaften eines Elements
  - Aufbau: `Attributname="Attributwert"`
  - Anwendung: `<Tagname Attributname="Attributwert" />`