Universität Leipzig **WS 14/15**
Institut für Informatik **Dr. Monika Meiler**

Modellierung und Programmierung 1
Übungsserie 6

Abgabetermin: 01.02.2015, 23:55 Uhr

Grundsätzlich sind Nebenrechnungen anzugeben und Antworten zu begründen.
Einzureichen sind, bei mehreren Dateien als .zip-Archiv:
Programme als Quellcode, Ergebnisdateien, Online-Dokumentationen.

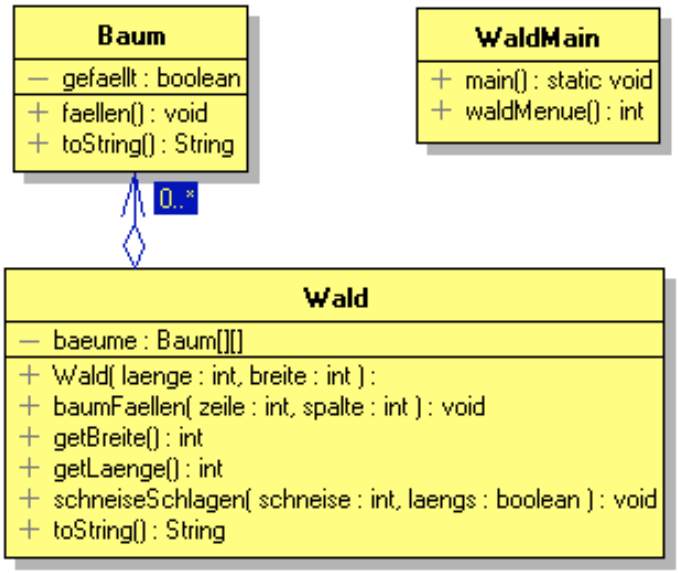
---

1. **OOP - Implementieren**

   Man sieht den Wald vor lauter Bäumen nicht, wenn die Bäume im Wald systematisch in Reihe und Glied stehen, denn von jedem Standort aus ist der Blick auf viele Bäume versperrt.
   In einem rechtwinkligen Gitter einer Schonung stehen auf allen Gitterpunkten Bäume, punktförmig, ohne Ausdehnung. Zur Weihnachtszeit wurde durch das Fällen einiger Bäume die Schonung ausgelichtet.
   Der für die Schonung zuständige Förster hält im Büro mittels eines Verwaltungsprogramms den Zustand seiner Schonung fest. Folgende Aktivitäten sollen im Programm berücksichtigt werden:

   - Ein Baum wird gefällt.
   - Eine Schneise (Zeile oder Spalte von Bäumen) wird geschlagen.

   Gegeben sei das UML-Klassendiagramm einer ***Klasse Baum*** und einer ***Klasse Wald***:



   Implemetieren Sie die beiden Klassen:

   a) ***Klasse Baum***: Legen Sie die Instanzvariable `gefaellt` an, simulieren Sie das Fällen eines Baumes in der Instanzmethode `faellen()` und markieren Sie in der `toString`-Methode einen gefällten und nicht gefällten Baum unterschiedlich, zum Beispiel mit “o” und “x”.

   b) ***Klasse Wald***: Legen Sie den Wald als Instanzvariable `baeume` an. Führen Sie dabei einen notwendigen Konstruktor zum Walderzeugen ein. Simulieren Sie das Fällen eines Baumes

oder einer Baumschneise in entsprechenden Instanzmethoden und ermöglichen Sie durch Überschreiben der `toString`-Methode eine Darstellung des Waldes als Raster der gefällten bzw. ungefällten Bäume.
In dem Beispiel einer 7×9 - Schonung wurden alle Bäume der 4. Zeile und der 5. Spalte, sowie ein Baum in der 7. Zeile an der 8. Stelle gerodet:

```
x x x x o x x x x
x o x x o x x x x
x x x x o x x x x
o o o o o o o o o
x x x x o x x x x
x x x x o x x x x
x x x x o x x o x
```

c) Dem Förster wurde eine Fichtenschonung zugeordnet. In einem bereits entwickelten Verwaltungsprogramm **WaldMain.java** kann er menügesteuert Veränderungen innerhalb seiner Schonung eintragen. Dieses Programm kann das Fällen von Bäumen oder ganzer Baumschneisen registrieren und den aktuellen Zustand eines Waldes anzeigen.
Wenden Sie die von Ihnen entwickelten Klassen zusammen mit dem Verwaltungsprogramm in einem Test auf eine 23×35 - Schonung an:

- Ein Baum musste wegen eines Blitzeinschlages gefällt werden. Er stand in der 13. Zeile an 19. Stelle von links gezählt. Jede 10. Baumspalte und 7. Baumzeile wurde aus Brandschutzgründen geschlagen. Zum Ausdünnen wurden jede 3. Spalte und 3. Zeile als Weihnachtsbäume abgeholzt.
- Erzeugen Sie mit Ihrem Programm die Rasterdarstellung und übertagen Sie den aktuellen Zustand der Schonung in eine Datei **WaldMain.out**.
- Versuchen Sie einen Baum außerhalb der Schonung zu fällen, etwa in der 24. Zeile an 18. Stelle. Wie reagiert das Programm?

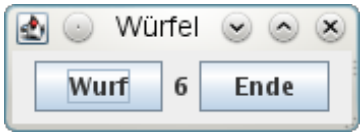
Vervollständigen Sie die Klassen durch geeignete `javadoc`-Kommentare und erstellen Sie eine Online-Dokumentation für das Gesamtprojekt:

```
javadoc -private -d WaldDoc *.java
```

2. **GUI - Benutzeroberflächen**

Entwickeln Sie ein Programm **Wuerfel.java** zum Würfeln der Zahlen von 1 bis 6 mit grafischer Benutzerschnittstelle. Analysieren Sie dazu die Programme **HalloWeltGUI1.java** und **HalloWeltGUI2.java** aus der Vorlesung.

a) Aufbau des Würfels:
Die Menüleiste des Fensters erhält „Würfel“ als Titel. Ein Button `btWurf` wird mit der Aufschrift „Wurf“ und ein Button `btEnde` mit der Aufschrift „Ende“ installiert. Ein Label `lbAugen` zeigt die gewürfelte Zahl und startet mit der Augenzahl 0.
Erzeugen sie eine Oberfläche mit den entsprechenden Komponenten:



b) Ereignisverarbeitung:
Mit dem Wurf-Button wird das Würfeln ausgelöst, indem eine zufällige Zahl zwischen 1 und 6 erzeugt und durch das Label angezeigt wird. Der Ende-Button beendet die Anwendung und schließt das Fenster. Eine main-Methode startet den Würfel.
Verarbeiten Sie in Ihrem Programm die entsprechenden Ereignisse.

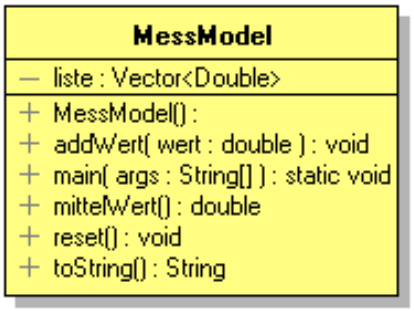
Achten Sie auf eine strukturierte und gut kommentierte Programmierung. Erzeugen Sie eine Online-Dokumentation mit **javadoc**:

```
javadoc -private -d WuerfelDoc *.java
```

Hinweise: Das Label `lbAugen` sollte als Instanzvariable für die sich während des Programms ändernde Augenzahl vereinbart werden. `lbAugen.setText( String str)` legt die Aufschrift eines Labels fest.

3. **MVC - Architektur**

Gegeben sei eine ***Klasse MessModel*** für die Mittelwertberechnung von Messwerten. Entwickeln Sie unter Verwendung der **MVC-Architektur** eine grafische Benutzerschnittstelle dazu.



Ergänzen Sie die Grundstrukturen der MVC-Klassen wie folgt:

a) Analysieren Sie den Quellcode der ***Klasse MessModel*** und richten Sie diese als das zu überwachende **Model** ein.

b) Installieren Sie im **View** ein Feld `JTextField` für die Werteeingabe und übergeben Sie im **Controller** exakt eingegebene Werte dem **Model**.

c) Binden Sie ein Label ein, welches durch die `update`-Methode des **Views** nach jeder Eingabe den vom **Model** neu berechneten Mittelwert anzeigt.

d) Nehmen Sie im **View** die Button „Neu“ für eine neue Berechnung und „Beenden“ für das Beenden des Programms auf und sorgen Sie im **Controller** für die exakte Funktionalität beider Button.

e) Schreiben Sie ein Startprogramm **MessMain.java** für die Mittelwertsberechnung. Starten Sie das Programm, geben Sie die Messwerte aus der Datei **MessWerte.txt** hintereinander ein und übertragen Sie den errechneten Mittelwert als letzte Zeile in diese Datei. Geben Sie die Datei **MessWerte.txt** mit ab.

Achten Sie auf eine strukturierte und gut kommentierte Programmierung. Erzeugen Sie eine Online-Dokumentation mit **javadoc**:

```
javadoc -private -d MessDoc *.java.
```

Hinweise:
Zur Tastaturabfrage benötigt man den `KeyListener`. Dieses Interface besitzt drei abstrakte Methoden zur Ereignisauswertung:

- „Taste wurde gedrueckt“: `public void keyPressed( KeyEvent ke)`
- „Taste wurde gedrueckt und losgelassen“: `public void keyTyped( KeyEvent ke)`
- „Taste wurde losgelassen“: `public void keyReleased( KeyEvent ke)`

Für unser Problem genügt es, die letzte Methode zu implementieren. Die ersten beiden erhalten einen leeren Methodenrumpf.

- `ke.getKeyChar()` liefert das zuletzt eingelesene Zeichen.
- `((JTextField)ke.getSource()).getText()` liefet den zuletzt eingelesenen String.
- `Double.valueOf( String)` wandelt den angegebenen String in ein `Double`-Objekt um, vorausgesetzt der String stellt eine Zahl dar, sonst wirft diese Methode eine Ausnahme.
- `setFehlerDialog( String)` erzeugt ein Standardfenster mit dem angegebenen String als Kommentar.