NACHPRÜFUNG FS 2017

OOP2

Bedingungen:

- Erlaubte Hilfsmittel: Unterrichtsunterlagen, Java Buch und Übungen.
- **Die Prüfung ist schrittweise, gemäss Aufgabenstellung lokal auf Ihrem Computer zu lösen. Kopieren sie zu diesem Zwecke den gesamten Ordner *PN_Pruefung_I_FS2017_Vorlage* auf Ihre lokale Harddisk und importieren sie das Projekt in Eclipse. Am Ende der Prüfung ist der Ordner src umbenannt in *NameVorname* abzugeben.**
- **Setzen sie als erstes Ihren Namen und Vornamen in die Datei.**
- **Gegenseitiges Abschreiben in irgendeiner Form führt zur Note 1!**
- **Folgend sie bei der Wahl von Variablen den Angaben in der Aufgabenstellung.**
- **Die Beilage muss abgegeben werden!**

Beschreibung:

Ziel dieser Prüfung ist es, ein einfaches E-Bike-O-Meter zur Visualisierung von KML[1] – Dateien zu schreiben:

1 Keyhole Markup Language (KML) ist eine Auszeichnungssprache zur Beschreibung von Geodaten für die Client-Komponenten der Programme Google Earth und Google Maps. KML befolgt die XML-Syntax, liegt in der Version 2.2 vor und ist ein Standard des Open Geospatial Consortium.

KML – Dateien lassen sich mittels Handy – Applikationen wie *GPSLogger* erzeugen, die in regelmässigen Abständen die mit dem GPS ermittelte Position inklusive Höhenangabe in eine KML-Datei ablegen. Damit lässt sich beim Radfahren die gefahrene Route aufzeichnen und analysieren.

Das E-Bike-O-Meter erlaubt die Geschwindigkeit, die Höhe und die Route als 2D-Trajektorie plus zusätzliche Informationen darzustellen. Die Applikation ist im uns bekannten Model-View-Controller Entwurfsmuster geschrieben.

Die Position ist im XML-Format:

<Point><coordinates>8.130859304219484,47.44138348381966,445.0999755859375</coordinates></Point>

angegeben.

Top – Down – Beschreibung[2]

Benutzerschnittstelle:

Die Benutzerschnittstelle ist in fünf *JPanel* gegliedert, die alle auf der *TopView* in einem *GridBagLayout* beheimatet sind: Der *VPlot* visualisiert den Geschwindigkeitsverlauf in Funktion der Zeit, der darunter liegende *HPlot* zeichnet den Höhenverlauf in Funktion der Zeit und der *XYPlot* zeichnet auf einem Hintergrundsbild den Verlauf der Trajektorie der Route. Der *VPlot* hat zwei mit der Maus verschiebbare Marker, deren Position in den übrigen Plots nachgeführt werden. Das *InfoPanel* beinhaltet Informationen, die aufgrund der Markerpositionen berechnet werden. Das *InputPanel* erlaubt mittels „Durchsuchen“ eine neue kml – Datei mit zugehörigem Zeitintervall zu laden. Die JPanel *HPlot* und *VPlot* wachsen bei einer Grössenänderung des Fensters in beide Richtungen, der *XYPlot* wächst in vertikaler Richtung und das *InfoPanel* ändert nur in horizontaler Richtung.

MenuBar:

Die *JMenuBar* verfügt über ein *JMenu* „Datei“ mit den *JMenuItem* „Lade Bild“ und „Exit“. Zur Auswahl der Dateien hat die *JMenuBar* einen *JFileChooser*. Die *JMenuBar* implementiert den benötigten *ActionListener* und verfügt über den *Controller*.

Controller:

Der *Controller* übernimmt die Steueraufgaben der Applikation und delegiert allenfalls den Aufruf an das *Model*. Die Methoden *loadKMLDatei*() und *loadXYBackgroundImage*() werden durch die durch die entsprechende Ladefunktion aufgerufen. Die Methode *setMarkerPosition*() dient der Steuerung der Marker und wird von *VPlot* aufgerufen.

Model:

Das Model beheimatet die Daten und Algorithmen und hat zu diesem Zwecke den zweidimensionalen Array *trace*, der in Zeilen abgelegt, die geografischen Längen-, Breiten- und Höhenangeben beinhaltet. Aufgrund dieser Daten werden mit *calculateProfiles*() und mit *calculateProperties*() die entsprechenden Profile und die fürs *InfoPanel* benötigten Eigenschaften berechnet. Das *Model* ist *Observable* und die *TopView* wird bereits in *BikeOMeter* als *Observer* registriert.

MVC Pattern:

Wir betrachten als erstes den Use-Case „Lade Bild“: Das Ereignis des *JMenuItems* löst die Methode *actionPerformed*() der *MenuBar* aus. *actionPerformed*() ruft *loadXYBackgroundImage* () des *Controllers* auf, die das entsprechende Bild des *XYPlots* via *TopView* setzt. Der Use-Case „Lade KML“ funktioniert sinngemäss.

Werden die Marker im *VPlot* bewegt, werden via *Controller* die Markerpositionen im *Model* verändert, das *Model* berechnet die neuen Informationswerte und ruft die entsprechende Methode der *TopView* via *notifyObservers*() auf. Die *TopView* delegiert das Update an all ihre *JPanel*.

---

[2] Als Top-down (engl., etwa ‚von oben nach unten‘) und Bottom-up (engl., etwa ‚von unten nach oben‘) werden zwei entgegengesetzte Arbeitsrichtungen eines Modellierungs-Prozesses bezeichnet, die in verschiedenen Sinnzusammenhängen für Analyse- oder Syntheserichtungen verwendet werden.
Die Bezeichnungen der Klassen werden im Text sinngemäss für deren Objekte verwendet. In diesem Dokument werden beim Genitiv werden die Klassenbezeichnungen in ihrer ursprünglichen Form beibehalten. Klassen, Methoden und Attribute werden kursiv geschrieben.

Aufgabe 1: Erstellen des Grundgerüstes (~ 5 Pte.)

Als erstes wollen wir das Grundgerüst des Programms anhand des Klassendiagramms ergänzen. Viele Elemente der Klassen sind dabei bereits gegeben. Gehen Sie jede Klasse durch und schauen Sie was bereits vorhanden ist und welche Elemente noch fehlen.

a) Ergänzen sie sämtliche *Attribut-Deklarationen* gemäss Klassendiagramm.

b) Füllen Sie in jede Klasse Ihren Namen ein.

Aufgabe 2: User – Interface (~ 75 Pte.)

Als erstes schreiben wir das User – Interface bestehend aus den Klassen *TopView*, *InfoPanel*, *InputPanel* und *MenuBar*.

a) Implementieren Sie das User – Interface gemäss Dokumentation.

b) Implementieren Sie alle *ActionListener* inklusive Code in den Methoden *actionPerformed*().

c) Implementieren Sie sämtliche *update*() – Methoden.

Aufgabe 3: Klasse *Controller* (~ 7 Pte.)

Als nächstes wollen wir den Controller schreiben.

a) Implementieren Sie die Klasse Controller gemäss Dokumentation im Code.

Aufgabe 4: Klasse *Model* (~ 51Pte.)

Nun gilt es, das Model zu implementieren.

a) Implementieren Sie die Klasse *Model* gemäss Dokumentation im Code und gemäss den Angaben auf dem nachfolgenden Blatt.

b) Fügen Sie allenfalls noch fehlende Elemente hinzu.

c) Testen Sie ihre Applikation und freuen Sie sich, wenn alles funktioniert.

Berechnungsalgorithmen:

- **Berechnung der Werte für *traceX* und *traceY***
  trace ist ein zweidimensionaler Array, dessen i-te Zeile *trace[i]* die Information bezüglich geografischer Länge, Breite und Höhe des i-ten Punktes trägt. *traceX[i]* und *traceY[i]* beinhalten die XY – Koordinaten des i-ten Punktes relativ zum Punkt gegeben durch *trace*[0]. Für alle i grösser gleich Null und kleiner der Länge des Arrays *traceX* gilt:

$$traceX[i] = \frac{ERDDURCHMESSER}{360}\cos(trace[i][1]) \cdot (trace[i][0] - trace[0][0])$$

$$traceY[i] = \frac{ERDDURCHMESSER}{360}(trace[i][1] - trace[0][1])$$

  Achtung: trace[i][1] beinhaltet den Winkel in Grad und muss zuerst in Radiant umgerechnet werden.

- **Berechnung der Werte für *sProfile***
  sProfile[i] beinhaltet die Distanz vom Ursprung bis zum i-ten Punkt. Für alle i grösser gleich Null und kleiner der Länge von *sProfile* gilt:

$$sProfile[i] = \sum_{k=0}^{i} \sqrt{(traceX[k+1] - traceX[k])^2 + (traceY[k+1] - traceY[k])^2}$$

- **Berechnung der Werte für *vProfile***
  vProfile[i] beinhaltet das Geschwindigkeitsprofil in km. Für alle i grösser gleich Null und kleiner der Länge von *vProfile* **minus 1** gilt:

$$vProfile[i] = 3.6 \cdot \frac{(sProfile[i+1] - sProfile[i])}{zeitIntervall}$$

  Der letzte Wert in *vProfile* wird gleich dem zweitletzten gesetzt.

- **Berechnung der Werte für *hProfile***
  hProfile[i] beinhaltet die Höhe über Meer. Für alle i grösser gleich Null und kleiner der Länge von *hProfile* gilt:

$$hProfile[i] = trace[i][2]$$

- **Berechnung der Werte für *tAxis***
  tAxis beinhaltet den Zeitvektor. Für alle i grösser gleich Null und kleiner der Länge von *tAxis* gilt:

$$tAxis[i] = zeitIntervall \cdot i$$