MVC

Overeenkomsten en verschillen MVCDynamicModelThread en -ThreadGeneralized

De projecten MVCDynamicModelThread en -ThreadGeneralized zijn voor het grootste gedeelte aan elkaar gelijk. Beide gebruiken de klassen Controller, CountView, Model, MVCDynamicModelThread (en Generalized), MVCRunner en PieView. De code van deze klassen zijn ook nagenoeg identiek aan elkaar. Het verschil is dat ThreadGeneralized abstracte klassen heeft waar zoveel mogelijk MVC-aspecten in zitten. Dit zijn dus de klassen AbstractView, -Controller en -Model. Het voordeel hiervan is dat het een stuk gemakkelijker wordt om meer klassen toe te voegen. Er kan bijvoorbeeld een View-klasse toegevoegd worden die AbstractView overerft. De code is ook duidelijker te volgen en is beter georganiseerd. Het project ModelThread heeft geen aparte abstracte klassen voor MVC. De MVC-aspecten van bijvoorbeeld AbstractController zitten daar in de klasse Controller. Het kost in zo'n geval meer tijd om wijzigingen aan te brengen in de code.

Een ander verschil is dat de klassen in verschillende packages zijn geordend. De klassen AbstractModel en Model (overerving) zitten in de package nl.hanze.t12.mvc.controller.

De folderstructuur is dus nl/hanze/t12/mvc/controller.

Overeenkomsten en verschillen MVCDynamicModelThreadGeneralized en Life

Het Project Life volgt dezelfde structuur voor MVC als ThreadGeneralized. Life heeft dus aparte abstracte klassen voor Model, View, en Controller die ook zijn geordend in packages.

Ze hebben allebei een aparte package voor main, waar de Viewers samenkomen en waar een JFrame aangemaakt wordt. Ook is er een package voor de runner, waar de main methode zich bevindt.

Een verschil tussen de twee projecten is dat Life in totaal 3 controllers heeft, inclusief een abstracte klasse. ThreadGeneralized heeft er 2, AbstractController en Controller. Life gebruikt het principe van het uitbreiden van de abstracte MVC-klassen dus al meer dan ThreadGeneralized.

MVC-structuur in Vossen & Konijnen

Om V&K dezelfde MVC-structuur te geven als Life, moeten de klassen dus samenkomen in packages en moeten er abstracte klassen voor de MVC-aspecten komen.

nl.vossen-en-konijnen.controller

AbstractSimulator

Simulator

nl.vossen-en-konijnen.logic

AbstractModel

Actor

Hunter

Animal

Rabbit

Fox

Lynx

Counter

Field

FieldStats

Randomizer

Location

nl.vossen-en-konijnen.main

SimulatorView

nl.vossen-en-konijnen.exeptions

Exeptions

nl.vossen-en-konijnen.runner

SimulatorRunner

nl.vossen-en-konijnen.view

AbstractView

Viewer

Opgaven 10.55 – 1.57

10.55

public interface Monitor

{

private static final int THRESHOLD = 50;

public Monitor (int initial);

public int getThreshold()

{

return THRESHOLD;

}

…

}

Omdat Monitor een interface is, kan een veld niet private zijn. Alleen public velden zijn toegestaan in een interface.

Ook kan public Monitor (int initial); niet, omdat interfaces geen constructors hebben. En als dit wel toegestaan zou zijn, zou een constructor geen semikolom hebben, maar een body.

De methode getThreshold() is hier ook niet mogelijk, omdat methodes altijd abstract zijn in interfaces. Dit betekent dat ze geen body mogen hebben, dit wordt overschreven in lager gelegen klassen.

10.61??

De view-klasse in Simulator is SimulatorView, geinitialiseerd en gedeclareerd als view = new SimulatorView(depth, width), waar de parameters een default-waarde hebben als private static veld.

View krijgt dan twee ActionListeners toegewezen, stepOneListener en stepHundredListener.

Vervolgens krijgen fox, rabbit, lynx en hunter een eigen kleur. Een konijn wordt bijvoorbeeld oranje met view.setColor(Rabbit.**class**, Color.***orange***);

10.62