Model – View – Controller

Objekt-orienterad programmering och design

Alex Gerdes, 2018

Model – View – Controller

- Model – View – Controller (MVC) är ett design pattern (architectural pattern) som är väldigt vanligt förekommande för alla typer av program som innehåller någon form av grafisk representation.

- Grunden i MVC är att vi separerar:
  - Koden för data-modellen (M) från användargränssnitt (V och C). Detta är den viktigaste uppdelningen.
  - Inom användargränssnitt skiljer vi koden som visar upp (delar av) modellen för användaren (V) från koden som hanterar input från (bl.a.) användaren (C).

Model

- Modellen (Model) är en representation av den domän som programmet arbetar över – helt oberoende av användargränssnitt.
  - Data, tillstånd, domänlogik.
  - Modellen i singular – vi har inte flera modeller för samma domän. Modellen kan dock internt bestå av flera olika delar som representerar *olika* aspekter av domänen (dvs ingen duplicering).

- Tumregel för avgränsning från V och C: Tänk ”The whole model and nothing but the model.”
  - Ska kunna presenteras för och kontrolleras av användaren på olika sett – e.g. med 2D-grafik eller text – utan att valet i sig påverkar e.g. modellens tillstånd.
    - ”Nothing but the model”
  - Ska innehålla allt det som vore detsamma oavsett hur modellen presenteras eller kontrolleras. Ingenting ska behöva dupliceras mellan olika vy- eller kontroll-komponenter.
    - ”The whole model”

View

- En vy (View) beskriver (delar av) modellen för användaren.
  - Olika tänkbara format: Text, grafik (2D, 3D, …), ljud, haptik, …
  - Vi pratar om ”vy”, oavsett vilket format presentationen görs på.

- Vi kan ha många olika vyer (ofta samtidigt) för en och samma modell.
  - Olika vyer kan visa upp olika aspekter av modellen.
    - E.g. karta, inventory, synfält för ett dataspel.
  - Olika vyer kan visa upp samma aspekt på olika format
    - E.g. musik spelas upp, och visas samtidigt grafiskt som frekvens-amplitud-diagram.

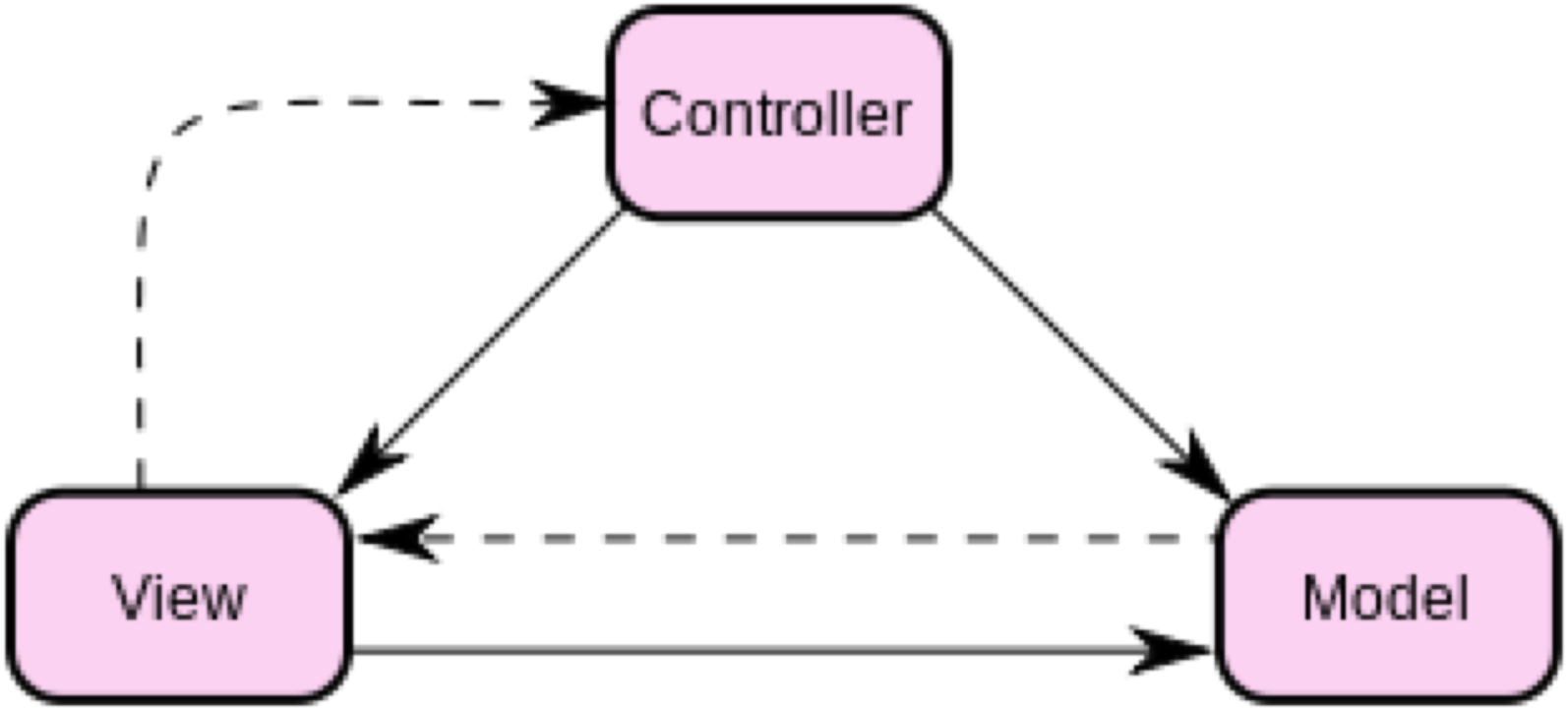
- Vi vill (oftast) att vyn uppdateras när modellen den presenterar uppdateras.
  - Finns olika sätt att se till detta (mer på nästa föreläsningen).

Controller

- En Controller styr modell och vy(er) utifrån input från (bl.a.) användaren.
  - Vi kan ha många controllers som styr olika aspekter av både modell och vy.
  - Olika varianter av MVC använder controllers lite olika: alltifrån en enskild controller som styr alla aspekter av modell och vyer, till många separata controllers för varje del-vy.

- De flesta moderna grafik-gränssnitt (som Java Swing) innehåller stöd för att förenkla för controllers att hantera användar-inputs genom *events* och *event listeners*.

MVC överblick



Source picture: Wikipedia

Övning- preamble

- Börja från koden som finns att ladda ner från hemsidan.
  - Vår gamla kod från förra veckan är uppdaterad enligt vad vi gjorde på förre föreläsningen (färdigställt): Vårt sub-paket `tda551.polygons.polygon` har nu en fasad (Facade Pattern) som består av `PolygonFactory` (Factory Method Pattern) och interfacet `IPolygon`. `DrawPolygons` beror endast på denna fasad, och inga paket-interna detaljer (det kan den inte eftersom dessa inte längre exponeras utanför paketet).
  - Paketet `tda551.shapes` är oförändrat sen förra veckan. Detta paket följer inte våra principer, och exponerar fortfarande paket-interna detaljer. Eftersom vi ”låtsas” att detta är ett paket vi laddat ner från nätet och inte vill ändra i, eftersom vi vill kunna ta del av framtida uppdateringar och bug-fixar, så låter vi det vara så för nu. För extra övning kan ni dock omvandla även detta paket så att det exponerar ett väldefinierat och genomtänkt gränssnitt (men inte just nu, det är inte fokus för dagens övning).
  - Vi har introducerat klasser och gränssnitt för att få det något annorlunda gränssnitt som `tda551.shapes` exponerar att fungera ihop med `DrawPolygons` (Adapter Pattern). Dessa ligger i ett separat paket `tda551.adapter`. Som uppföljning från förra gången, läs och förstå hur vår adapter fungerar, och varför vi behöver använda oss av denna. Byt sen import i `DrawPolygons`, från det gamla till det nya enligt instruktion i filen, och se att det ”bara funkar”.

Övning

- Betrakta nu vår kod utifrån ett MVC-perspektiv. Vad är vår Model? Vad är vår View? Vad är vår Controller? Är dessa väl åtskiljda från varandra? (Ledande fråga – svaret är såklart nej.)
- Den stora boven är klassen `DrawPolygons`, som har aspekter av alla tre benen av MVC i sig. Låt `DrawPolygons` vara den klass som definierar vårt toppnivå-program, som innehåller metoden `main`, och initierar de ingående komponenterna.
  - Skapa tre olika klasser:
    - `PolygonModel` (vår Model, hanterar alla polygoner och deras tillstånd);
    - `PolygonViewer` (vår View, visar polygoner på skärmen);
    - `PolygonController` (vår Controller, styr animation och user input).
  - Flytta relevanta delar av den gamla `DrawPolygons` till var och en av dessa.
  - Fundera över hur de behöver kommunicera med varandra. Vem behöver bero på vem?
  - Med de ändringar ni nu gjort – har vi nu en väl avgränsad uppdelning i enlighet med MVC? Svaret är inte helt självklart, på mer en ett sätt, oavsett hur ni löst det. Fundera över argument för och emot.
- Istället för att starta med en fix lista av polygoner, låt användaren klicka ut nya rektanglar. Titta på gränssnittet `java.awt.event.MouseListener`.
- För mer utmaning (kräver eget grävande i APIerna för Java Swing), lägg till fler komponenter till vår View, t ex:
  - Utöka fönstret med en panel som skriver ut en lista av de polygoner som finns i visningspanelen. Det räcker med att skriva ut center point för dem. Informationen ska uppdateras i samband med animationen.
  - Lägg till en knapp som startar och stoppar animationen.
  - Lägg till en väljare där användaren kan ställa in vilken sorts polygon som ritas ut när nya polygoner klickas fram.