Ten geleide:

de applicatie bestaat uit een tweetal grote composities.

de ene compositie staat voor **het model** van het type SolitaireGame. Bij instantiatie maakt hij al, de benodigde interne objecten aan. Zie de beschrijving verderop.

De andere compositie is **de view** van het type SolitaireView, ook dit object maakt zijn eigen set interne objecten aan. Zie de beschrijving verderop.

Beide composities zijn EventTargets en EventListeners.

EventTarget ben je als je overerft van de klasse, als je dat doet beschik je over de volgende extra faciliteiten van de klasse:

**EventTarget**

-listeners : Array

+ addEventListener(type,listener/callback)

+ removeEventListener(listener/callback)

+ dispatchEvent(event)

Alle aangemelde listeners krijgen het event dat via dispatchEvent verstuurd wordt. Het event bevat als het goed is al der update info benodigd voor de update. Deze implementatie is een toepassing van het zogeheten “observer observable model”.

Ze zijn ook beiden EventListeners. MDN stelt in deze: *je bent een EventListener als je de interface van EventListener implementeert.*

Zie ook: <https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>

<https://developer.mozilla.org/en-US/docs/Web/API/EventListener>

Dat wil zoveel zeggen als je moet verplicht een methode inhoud geven met de signatuur, je mag niet afwijken wat naam betreft.

+ handleEvent(event): void

Als je dan als object middels addEventListener toegevoegd wordt zal deze methode automatisch worden aangeroepen als er een event gestuurd wordt naar de listeners.

dispatchEvent maakt jou als programmeur degene die beslist wanneer en welk type event er verzonden wordt. Je kan dus je eigen eventtypen maken en bent dus veel vrijer dan enkel mousup, mouseenter, click etc te gebruiken.

Het event kan je gebruiken als datacarrier. Hij draagt zonodig gegevens over van de target naar de listener.

Dit ontslaat je van het gebruik van een controller. De update informatie die view aan model geeft en viceversa kan middels data aan het event worden meegegeven en er uitgehaald worden bij ontvangst van het event door de listener.

Elk eventtype draagt relevante informatie/data met zich mee. Deze data is voor de view verplicht aansturend om zich een update te geven. Voor het model beat het eventtype data over de wens die de gebruiker uitoefent op de view. Deze data kan maar hoeft niet door het model gebruikt worden om zijn toestand te updaten, bij een update krijgt de view weer een event als listener toegestuurd.

Events zien er als volgt uit

pileId is een getter voor de pilewaarop in de view gedropt is

cardId is een getter voor de eerste gesleepte kaart in de view.

Op basis van deze gegevens weet het model welke zet er in de view gedaan is en kan zich na accordering aanpassen.

piles is een getter voor de 13 piles die het model heeft. Slechts de 2 gewijzigde piles wijken af van [].

Round is een getter voor het rondenummer van een kaartspel

De view kan zich aldus aanpassen

**Event**

**ViewCardsDroppedEvent**

-#TYPE

-#pile\_id

-#card\_id

-<<get>>pileId

-<<get>>cardId

**Event**

**ModelChangeEvent**

-#TYPE

-#piles

-#round

-<<get>>piles

-<<get>>round

**Deck**

-#cards Card[]

+ <<get>>cards: Card[]

+ shuffle() void

**Card**

-#id int

+visible bool

+ <<get>>id : int

+ <<get>>value : int

+<<get>> color : int

+ <<get>>symbol:int

+ isNextInLine(c:Card):bool

+ isBestFriend(c : Card):bool

+ copy():Card

1

52

**SolitaireGame**

-#deck: Deck

-#cardPiles: Card[][]

-#gameRoundHistory:GameRoundHistory

-#round: int

**EventTarget**

- #start():void

- #openNewCard():bool

- #placeCardsOnPile(pID:int,cID:int):void

-#getChanges(pID1:int,pID2:int): Card[][]

-#fireModelChanged(cPs:Card[][]):void

-#getCardsToDrop(sPile:Card[],cId:int)Card[]

- #getSourcePileId(cardId:int):int

-#hasRoundEnded():bool

- #nextRound():void

-#updateRoundHistory(sId, s, dId, d):void

-#undoMove():void

+ handleEvent(event):void

1

1

1

52

**GameRoundHistory**

- #moves:GameMove[]

+empty():void

+add(frID,frPile,toId,toPile)

+previous():GameMove

+clear():void

1

1

1

**GameMove**

-#sourceId:int

-#source:Card[]

-#destinationId:int

-#destination:Card[]

+ <<get>>destination: Object

+ <<get>>source: Object

n

­­­

1

n

**CardPileView**

- #id: int

- #top : int

- #cardViews : CardView[]

+ CARD\_CONTAINERS : domel[]

+ LEADING\_DROPPED\_CARD\_ID

+<<get>>container : domelement

+<<get>>cardViews : CardView[]

+empty() :bool

+addEventListener(type,callback)

+show(cards : Card[]) : void

+ handleStartDrag(e)

+ handleDragDrop(e)

-#getIdDroppedContainer(e)

+ getDraggables(e):int[]

- #fireDragDropped(id : int)

- #fireDragStarted(cards,pageX, pageY)

+ cardToView(card: Card) : void

**EventTarget**

**DiscardPileView**

+ getDraggables(e):int[]

**GamePileView**

+ getDraggables(e):int[]

+ show(cards : Card[]) : void

**SolitaireView**

- #ghostDiv : domElement

- #piles CardPileView[]

- #createUndoMoveEventDispatcher()

- #createNewCardEventDispatcher()

- #createNewGameEventDispatcher()

- #startDrag(event)

- #onDrag(event)

- #showHiddenCards()

- #show(piles)

- #showRound(value)

+ handleEvent(event)

**CardView**

- #id: int

- #front : string

- #html: domelements

-#BACK : string

- #SYMBOLS : enum

- #VALUES : enum

+<<get>>html : domelements

+<<set>>front : string

+<<get>>symbol: string

+<<get>>id : int

+<<get>>value: string

+contains(domel):bool

+visible():bool

+hide(value:bool):void

1

13

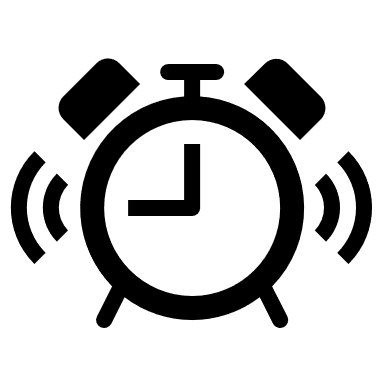
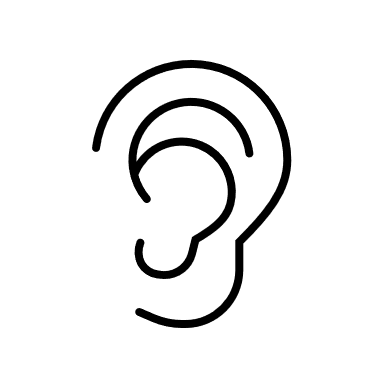
**EventTarget**

Het model zendt **ModelChangeEvents** uit als hij zijn toestand wijzigt.

De view is als listener aangemeld. De view past zich aan middels data in het event.

Zie daartoe de handleEvent methode

**SolitaireModel**



De view zendt 4 typen events uit:

**ViewNewGameEvents**

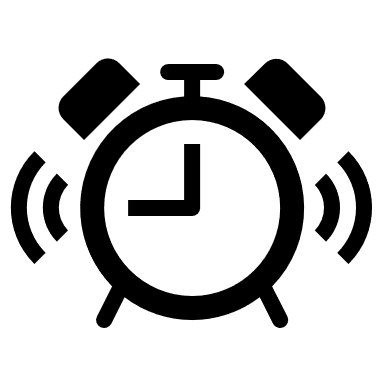
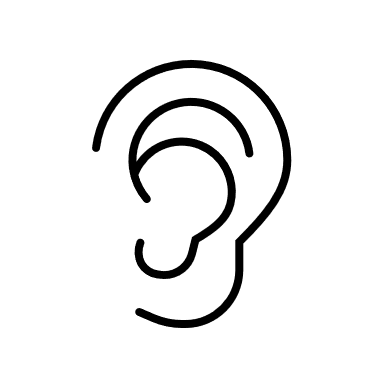
**ViewNewCardEvents**

**ViewCardsDroppedEvents**

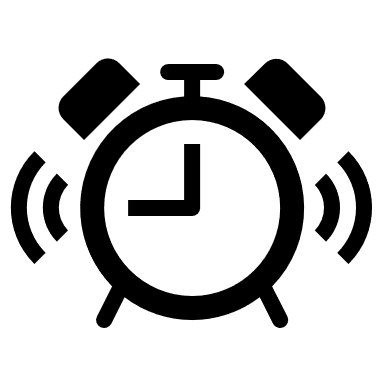
**ViewUndoMoveEvents**

Het model is als listener aangemeld. Het model kan aldus zijn toestand aanpassen. Zie daartoe de handleEvent methode

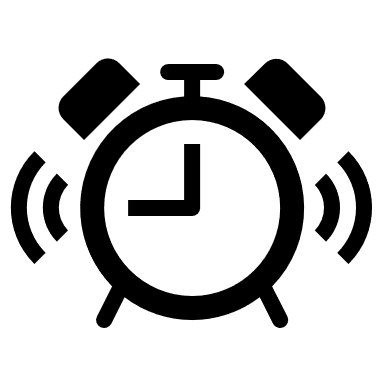
**SolitaireView**



**DisCardPileView**

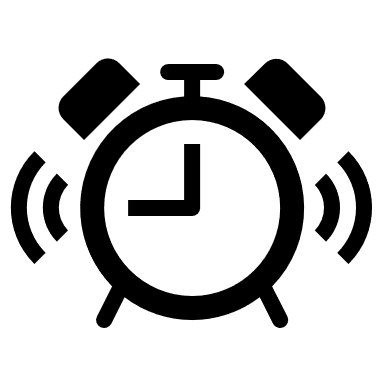


**CardPileView**





**GamePileView**



C

C

G

G

G

G

G

G

G

De view heeft:

1 DiscardPIleView (Zie D)

7 GamePileViews (zie G)

5 CardPileViews (zie C)

Deze in totaal 13 piles zijn allemaal EventTargets.

Ze zenden 2 typen events uit:

**PileDragStartEvents**

**PileDragDropEvents**

De view is als listener aangemeld -dit gebeurt in de constructor-en verzorgt het slepen van de kaarten van de ene naar de andere pile. Het **PileDragDropEvent** wordt als **ViewCardsDroppedEvent** overgedragen aan het model, die immers ook listener is van de view.

C

C

D

C

Afbeelding met tekst

Automatisch gegenereerde beschrijving