cards

Design Patterns Team

November 16, 2015

Contents

1 Card shape

- Rectangle, circle, etc
- Corners, rounded, sharp?

2 Decal

- Generic art, gets an art object
- Decals will need to be patterned, one decal might want to be on each corner, or might want to go in a row, would be very awkward to have to manually do all this patterning : Decal shouldn't do patterning, let the transformation class do it.

2.1 Decal types

- Image decals
- Number decals
- Text decals
- Shape decal

2.2 Nesting

Decals will want to be nested, a text box might want to be on top of a back ground.

2.3 Layout class

Decorates an image with scale, position, etc, can be composited, and is clonable?

- Could then handle patterning
- Composite

2.3.1 Background

- Decal, covering the whole card

2.3.2 General patterns

- Textbox
- Border
- Etc?

3 Theming

- Sub decks
- Some sets will have the same decal applied to the same spot
- Others will have the same decal, but used in a different spot per card
  - EX) Cards have some number of $family decals, but those decals are in different spots, they don't know what image they are until we tell it its family. Thus, would say something like $family = spades, and all the $family decals will use the spades image.

    ```
    clone =  prototype.clone();
    clone.setDecal("family",spades);
    ```

    Families will still need per card information. So perhaps...

    ```
    clone.setCharacter(charizard.jpg)
    clone.setHP(120);
    ```

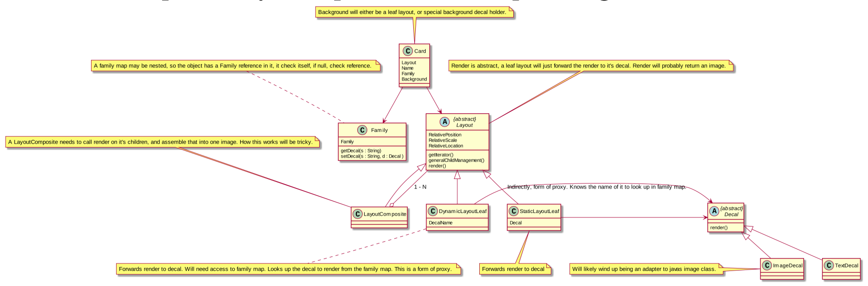
4 Cards

Card has a layout. Layouts can be cloned. Cards can be cloned.

- Card will have a name.
- Card will have a layout
- Card have a family? Family is just a map, string -> decal, Two types of layout leafs. FamilyLeaf, which just has a string, asks its family. DecalLeaf, which holds a decal.
  - A family can be nested, will query its map, then parent map.

5 Example UML

This should probably be split into multiple diagrams.


Background will either be a leaf layout, or special background decal holder.
Card
Layout
Name
Family
Background
A family map may be nested, so the object has a Family reference in it, it check itself, if null, check reference.
Render is abstract, a leaf layout will just forward the render to it's decal. Render will probably return an image.
Family
Family
getDecal(s : String)
setDecal(s : String, d : Decal)
{abstract} Layout
RelativePosition
RelativeScale
RelativeLocation
getIterator()
generalChildManagement()
render()
A LayoutComposite needs to call render on it's children, and assemble that into one image. How this works will be tricky.
1 - N
Indirectly, form of proxy. Knows the name of it to look up in family map.
LayoutComposite
DynamicLayoutLeaf
DecalName
StaticLayoutLeaf
Decal
{abstract} Decal
render()
Forwards render to decal. Will need access to family map. Looks up the decal to render from the family map. This is a form of proxy.
Forwards render to decal
Will likely wind up being an adapter to javas image class.
ImageDecal
TextDecal

- Some patterns used here
  - Proxy : the dynamicleaflayout
  - Composite : The layouts
  - Adapter : Image Decal
  - Iterator : For the composite

6 Scripting

6.1 Configuration file

```
decal = ...; //Some image file...
LeafDecalFoo = { "position" : ..., "decal" : decal }
LeafDecalBar = { "position" : ..., "decal" : "suite" }
positionA = [ 50, 100, 50, 100 ]
positionB = [ 50, 100, 50, 100 ]
LayoutB = { "position" : positionB, "leafa":LeafDecalBar, "leafb":LeafDecalFoo };

LayoutC = {
    "position" : positionA;
    "layoutA" : {"position" : positionA, "foo" :LeafDecalFoo, "bar":LeafDecalBar },
    "layoutB" : LayoutB
}

LayoutD = {
    "position" : positionB,
    "layoutA" : LayoutC, //"layoutA" is a clone of LayoutC
    "layoutB" : LayoutC["layoutA"] //"layoutB" is a clone of LayoutC["layoutA"]
}
LayoutD["layoutA"]["position"] = [0,100,0,100];


//Layouts are always cloned! But they point to the same, immutable decals,
//so they are cheap to clone.
backgroundDecal = ..;
heartDecal = ...;
clubDecal = ...;
familyBackground = {
    family : null,
    "background" : backgroundDecal
}
heartFamily = {
    family : familyBackground,
    "suite" : heartDecal
}

clubFamily = {
    family : familyBackground,
    "suite" : clubDecal
}

CardOne = {
    "name" = "cardA",
    "LayoutFront" = LayoutD,
    "LayoutBack" = LayoutC,
    family = heartFamily
}
```

6.2 Script

```
CardSetA = [CardOne,CardOne];
CardSetB = CardOne.clone(10);
CardSetA.setFamily(clubFamily);
CardSetA.add(CardOne);
```

7 Example with new lisp for 52 cards

```
spadeImage = Image("spadeImage.jpg")
```

```
heartImage = Image("heartImage.jpg")
diamondImage = Image("diamondImage.jpg")
clubImage = Image("clubImage.jpg")

jackImage = Image("jack.jpg")
queenImage = Image("queen.jpg")
kingImage = Image("king.jpg")

whiteRectangle = Rectangle("white")
wholeCard = Position(0,0,100,100)
cardSize = Size(1000,1000)
//..Moreposiitons

ace = character(A) //This needs to be replaced with string.
two = character(2)
three = character(3)
four = character(4)
five = character(5)
six = character(6)
seven = character(7)
eight = character(8)
nine = character(9)
10 = character(10)
J = character(J)
Q = character(Q)
K = character(K)

(define emptyFamily (family))
(define heartFamily (family (cons "suite" heartImage)))
(define diamondFamily (family (cons "suite" diamondImage)))
(define spadeFamily (family (cons "suite" spadeImage)))
(define clubFamily (family (cons "suite" clubImage)))

(define backgroundLayout
  (layout
  (cons (leaf-layout whiteRectangle) wholeCard)
  (cons (leaf-layout "suite") topRightCorner)
  (cons (leaf-layout "suite") bottomLeftCorner))
)
```

```
(define jackLayout
  (layout
  (cons backGroundLayout wholeCard)
  (cons (leaf-layout J) topRightCorner)
  (cons (leaf-layout J) bottomLeftCorner)
  (cons (leaf-layout jackImage) center))
)

(define queenLayout
  (layout
  (cons backGroundLayout wholeCard)
  (cons (leaf-layout Q) topRightCorner)
  (cons (leaf-layout Q) bottomLeftCorner)
  (cons (leaf-layout queenImage) center))
)

(define kingLayout
  (layout
  (cons backGroundLayout wholeCard)
  (cons (leaf-layout K) topRightCorner)
  (cons (leaf-layout K) bottomLeftCorner)
  (cons (leaf-layout kingImage) center))
)

(define KingCard
  (card cardSize "King" kingLayout kingLayout)
)

(define JackCard
  (card cardSize "Jack" JackLayout JackLayout)
)

(define QueenCard
  (card cardSize "Queen" QueenLayout QueenLayout)
)

(define oneLayout
  (layout
  (cons backGroundLayout wholeCard)
  (cons (leaf-layout one) topRightCorner)
  (cons (leaf-layout one) topLeftCorner)
  (cons (leaf-layout "suite") center)
)

(define oneCard (card cardSize "One" oneLayout oneLayout))
...

(define tenLayout
  (layout
  (cons backGroundLayout wholeCard)
  (cons (leaf-layout ten) topRightCorner)
  (cons (leaf-layout ten) topLeftCorner)
  (cons (leaf-layout "suite") center-10-1)
  (cons (leaf-layout "suite") center-10-2)
  (cons (leaf-layout "suite") center-10-3)
  (cons (leaf-layout "suite") center-10-4)
  (cons (leaf-layout "suite") center-10-5)
  (cons (leaf-layout "suite") center-10-6)
  (cons (leaf-layout "suite") center-10-7)
  (cons (leaf-layout "suite") center-10-8)
  (cons (leaf-layout "suite") center-10-9)
  (cons (leaf-layout "suite") center-10-10)
)

(define tenCard (card cardSize "Ten" tenLayout tenLayout))


(define suite (list QueenCard JackCard KingCard oneCard ... tenCard))

(render suite heartFamily)
(render suite clubFamily)
(render suite diamondSuite)
(render suite spadeSuite)
```