cards

Design Patterns Team

November 11, 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);
    ```

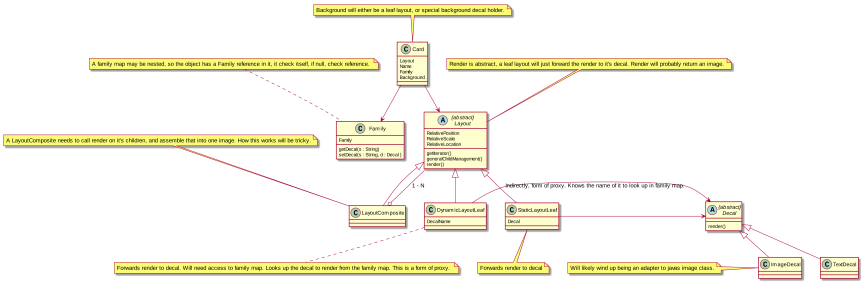
4 Cards

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.



- 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);
```