Factories

*Creational Patterns*
Simple Factories
Abstract Factory Pattern (GoF 87)
Factory Method Pattern (GoF 107)

Making Things
A disclaimer

I Love Canada!
I’ve been there many times and spent 6 months in Toronto once. Any perceived insult to the fine country of Canada should be considered for humor purposes only.

Blame Canada
Blame Canada
With all their beady little eyes
And flapping heads so full of lies

_Q: How do you stop bacon from curling in the frying pan?_
_A: Take away their brooms!_

_Blame Canada, Academy Award nominee for Best Song_
How would you describe this guy (in software)

Harold
His Parts

- Shirt
- Right Leg
- Left Leg
- Right Arm
- Right Hand
- Left Arm
- Left Hand
- Vest
- Head Bottom
- Head Top

Any patterns come to mind?
His Parts

- Shirt
- Right Leg
- Left Leg
- Right Arm
- Right Hand
- Left Arm
- Left Hand
- Vest
- Head Bottom
- Head Top

You should be thinking of the *composite pattern* here
His Parts

- Shirt
- Right Leg
- Left Leg
- Right Arm
- Right Hand
- Left Arm
- Left Hand
- Vest
- Head Bottom
- Head Top

Our system has drawable items, each of which has a location, rotation, and a center of rotation. Items can be arranged hierarchically, such that moving a parent also moves the children. Some drawables, like the arms, are simple polygons. Some, like the vest, are images. The top of the head has an image and also draws different types of eyebrows and eyes. We’ll also need to be able to draw text bubbles on the screen so our characters can talk. We will need to be able to draw our items in an order unrelated to the tree structure.
UML (my idea)

Difference from previous Composite Pattern: Every Drawable can have children (why?)

Just the drawables
Design Decisions

What classes?

All items need a center, position, and angle – hence base class Drawable.

Hand and arms are simple polygons.

Vest and shirt are pictures.

Head is picture plus eyes and eyebrows (special).
What I’m Really Interested In Today

What is it going to take to create Harold?

Where should we put this code?

What if we have several characters, props, and even quadrupeds?
First Idea

CDrawable *CFrame::CreateHarold()
{
    //...
}

What is bad about this approach?
Separation of Concern

The process of separating code into distinct features that overlap in functionality as little as possible.

```cpp
CDrawable *CFrame::CreateHarold()
{
    //...
}
```

One class here is managing the window on the screen AND creating our character. We have a lot of overlap.
Next Idea

“Global” functions?

CDrawable *CreateHarold()
{
  //...
}

What if CreateHarold needs some temporary values? (it will)

In general, global functions are to be avoided in object-oriented programming.
The Simple Factory Pattern

```cpp
class CHaroldFactory
{
public:
    CHaroldFactory(void);
    ~CHaroldFactory(void);
    CDrawable *Create();
};
```

A Factory Class encapsulates the code to create an object or collection of objects.

How it is used

```cpp
CHaroldFactory factory;
CDrawable *harold = factory.Create();
canvas.Add(harold);
```
Factories are a very powerful tool

It is easy to tell your factory something it needs to know to create something for you.

```cpp
class CHaroldFactory
{
    public:
        CHaroldFactory(CImageCollection *imageCollection) :
            mImageCollection(imageCollection) {}
        ~CHaroldFactory(void);

        CDrawable *Create();

    private:
        CImageCollection *mImageCollection;
};
```

```cpp
CHaroldFactory factory(mImageCollection);
CDrawable *harold = factory.Create();
canvas.Add(harold);
```
Parameterized Factories

```cpp
class CHaroldFactory
{
public:
    CCharoldFactory(CImageCollection *imageCollection) :
        mImageCollection(imageCollection) {}
~CCharoldFactory(void);

    CDrawable *Create();
    void SetFat(bool f) {mFat = f;}

private:
    CImageCollection *mImageCollection;
    bool mFat;
};
```

Harold’s fat American cousin
Options

class CHaroldFactory
{
public:
    CHaroldFactory(CImageCollection *imageCollection) :
        mImageCollection(imageCollection) {}
    ~CHaroldFactory(void);

    CDrawable *CreateHarold();
    CDrawable *CreateHaroldsSign();
    void SetFat(bool f) {mFat = f;}

private:
    CImageCollection *mImageCollection;
    bool mFat;
};
But...

What if we want to create many different characters and things? We can create a simple factory for each.

What are the disadvantages of this approach?
Factory Classes Defined

A Factory Class defines objects that encapsulate the functionality that creates objects or sets of objects of a different type.
Remember our discussion about redundant code?

Are there elements of the factory process that are redundant between factories?

If so, create a common base class and move functionality up.
Disadvantages of Simple Factory

We really have **two** concepts here:

- Make something we can draw
- Make Harold

Remember *Separation of Concern*?

The first applies to anything we might want to draw. Any ideas on what we should be considering?
But, more important

class CDrawableFactory
{
    public:
        CDrawableFactory(CImageCollection *imageCollection) :
            mImageCollection(imageCollection) {}
        virtual ~CDrawableFactory(void);

        virtual CDrawable *Create() = 0;

    private:
        CImageCollection *mImageCollection;
};

class CHaroldFactory : public CDrawableFactory
{
    public:
        CHaroldFactory(CImageCollection *imageCollection) :
            CDrawableFactory(imageCollection) {}
        ~CHaroldFactory(void);

        virtual CDrawable *Create();
        void SetFat(bool f) {mFat = f;}

    private:
        bool mFat;
};
Abstract Factory Pattern

class CDrawableFactory
{
public:
    CDrawableFactory(CImageCollection *imageCollection) :
        mImageCollection(imageCollection) {}
    virtual ~CDrawableFactory(void);

    virtual CDrawable *Create() = 0;

private:
    CImageCollection *mImageCollection;
};
Most *Basic* Abstract Factory

class CAbstractFactory
{
public:
    CAbstractFactory() {}  
    virtual ~CAbstractFactory() {}  

    virtual CProduct *Create() = 0;
};

An Abstract Factory is a generalization of how to make some “product”. A product is an object or set of objects.

CProduct is likely a base class as well.
Other Observations

Abstract Factories are great for team projects. You have a common interface to call and everyone writes their own factory to make their objects.

Abstract Factories make it easy to add a new type to your system. The code to create the new type goes in a new derived class.
An Example

CSE 471 Builds a Music Synthesizer.

There are instruments (like a piano or drum) and notes (what is making the sound right now). A piano or drum instrument generates notes when they need to be played. We tell the instrument what note to play using XML.

```xml
<?xml version="1.0" encoding="utf-8"?>
<score bpm="60" beatspermeasure="4">  
  <instrument instrument="Piano">
    <note measure="1" beat="1" duration="1.9" note="C4"/>
    <note measure="1" beat="3" duration="1.9" note="G4"/>
    <note measure="2" beat="1" duration="3.7" note="C5"/>
    <note measure="2" beat="4.75" duration="0.25" note="C4"/>
  </instrument>
</score>
```
The Factory Method

We commonly use Factory Methods in classes other than actual factory classes. The idea is to let a subclass handle the actual creation.
Document/View Architectures

Very common approach. What pattern was this?

The Factory Method (CreateView) means a document knows how to create a view compatible with the document.

wxWidgets, Microsoft MFC, Word, Excel, lots of other systems use the Document/View architecture.
Definition of Factory Method pattern

Defines an interface (pure virtual function) for creating an object, but lets subclasses decide what object to instantiate.
All Together Now

*Simple Factory* – A class that encapsulates the functionality of creating objects or sets of objects.

*Abstract Factory* – An interface for creating families of related objects.

*Factory Method* – An interface *in a class* that lets subclasses create objects of an appropriate type.