FlyweightCode

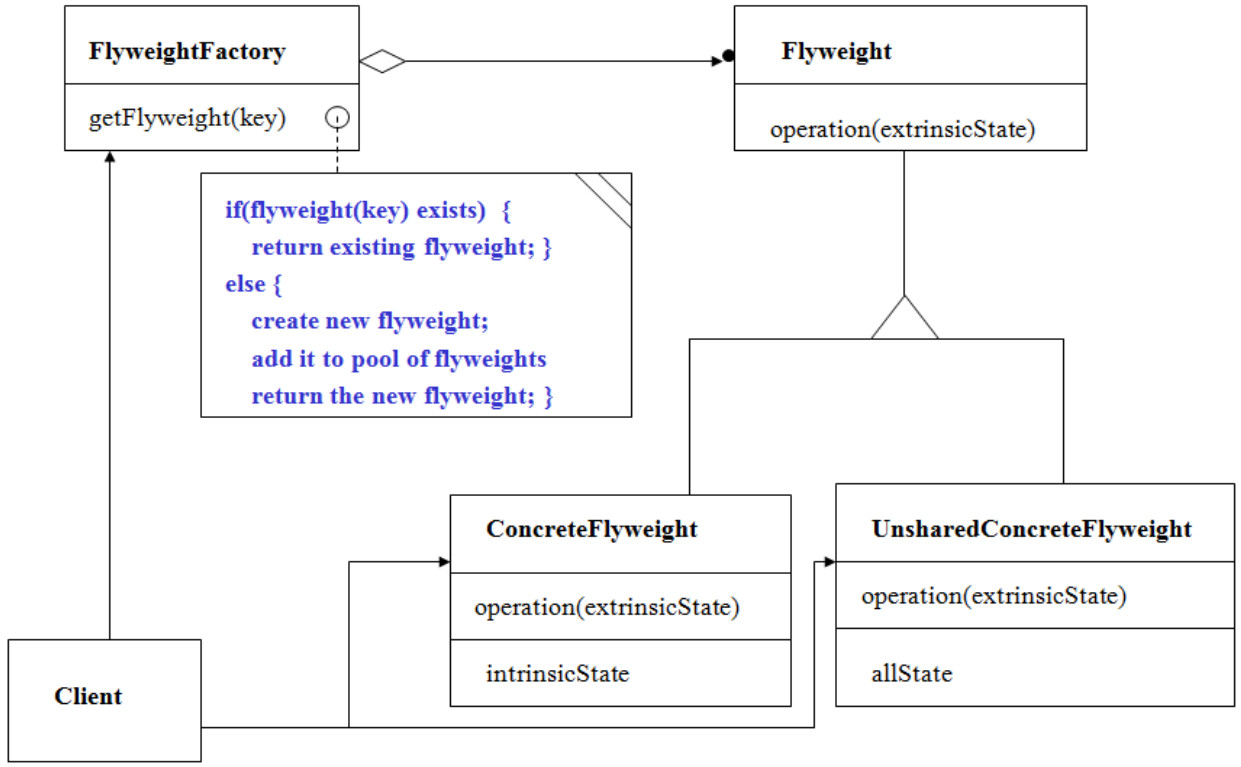
Jim Fawcett

CSE776 – Design Patterns

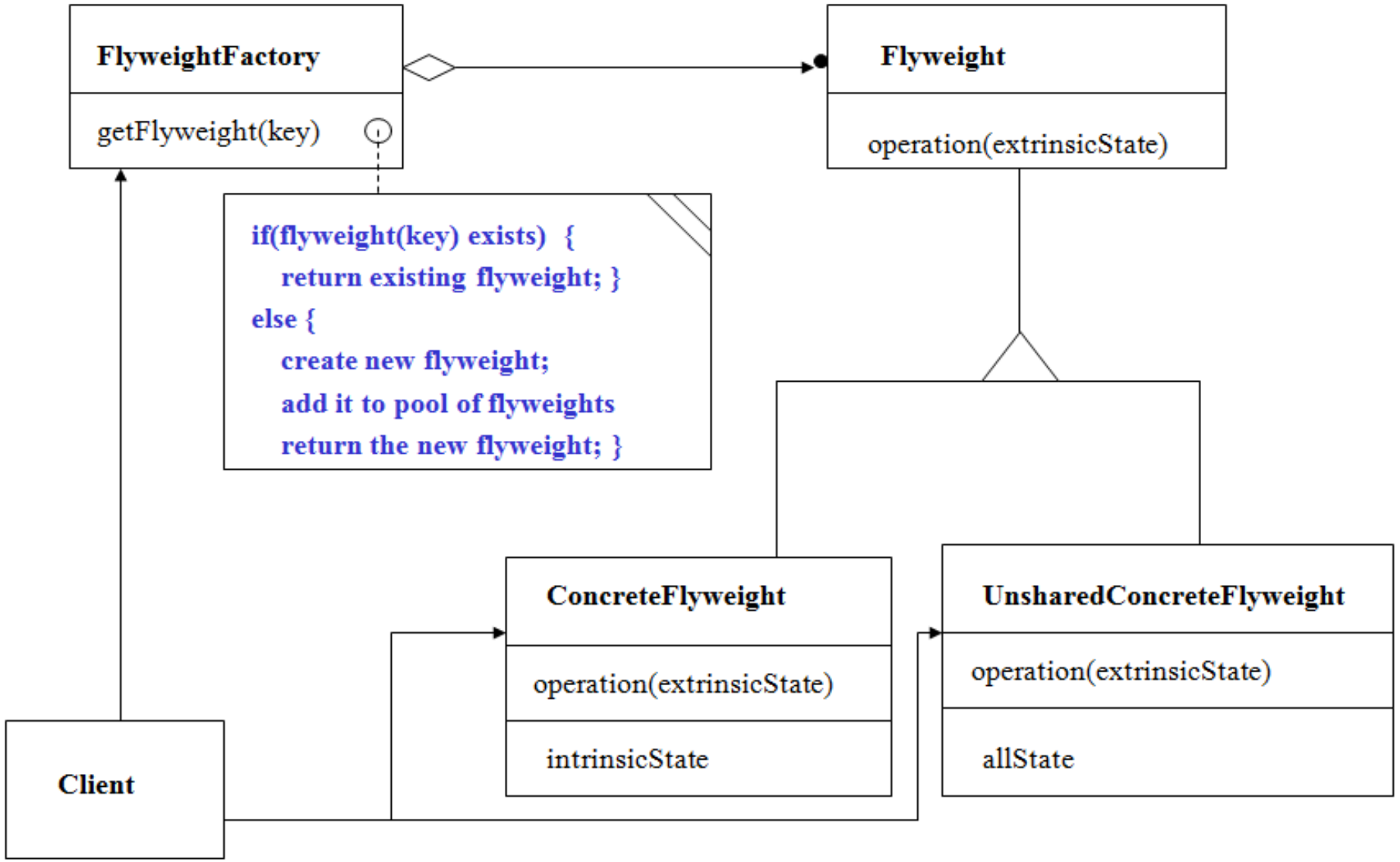
Fall 2018

Flyweight Pattern

- Flyweight instances have two kinds of state:
  - Large (intrinsic) shared state – same for all instances
  - Small (extrinsic) state unique to each object, often computed by the application and passed to its operation(extrinsicState) method.



Structure

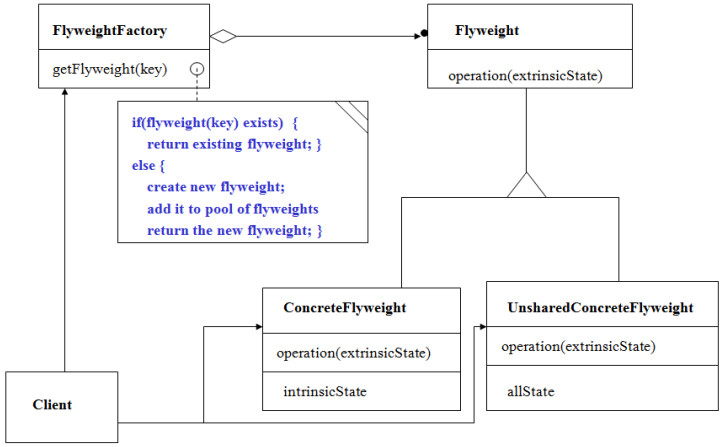

FlyweightFactory
getFlyweight(key)
Flyweight
operation(extrinsicState)
if(flyweight(key) exists) {
return existing flyweight; }
else {
create new flyweight;
add it to pool of flyweights
return the new flyweight; }
ConcreteFlyweight
operation(extrinsicState)
intrinsicState
UnsharedConcreteFlyweight
operation(extrinsicState)
allState
Client

Skeleton Code

- This skeleton code is really a framework for building application specific flyweighs.
- It defines its ExtrinsicState and IntrinsicState as template parameters
  - For some applications, these may be the only things the application has to build to represent its flyweights.
- Example Application:
  - Game with swordsmen, archers, footsoldiers, and cavalrymen.
    - A game state may have hundreds of active soldiers.
  - Intrinsic state is the graphics lines and fills necessary to render each type of soldier.
    - Assumes that all characters are rendered by a single procedure that uses lines and fills.
  - Extrinsic state is their positions, actions, and health.

Flyweight classes

```
template <typename ExState, typename InState>
class FlyweightFactory;

//////////////////////////////////////////////
// Flyweight

template <typename ExState, typename InState>
class Flyweight
{
public:
  virtual ~Flyweight() {}
  virtual void Operation(ExState)=0;
};
```

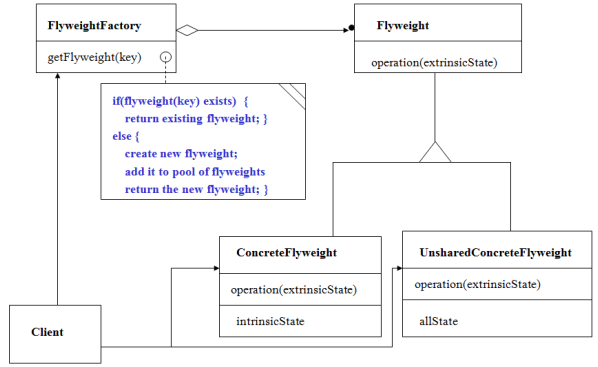



```
//////////////////////////////////////////////////////////
// ConcreteFlyweight

template <typename ExState, typename InState, size_t key>
class ConcreteFlyweight : public Flyweight<ExState, InState>
{
  friend class FlyweightFactory<ExState, InState>;
public:
  ConcreteFlyweight() {}
  ~ConcreteFlyweight() {
    std::cout << "\n  destroying concrete flyweight";
  }
  void Operation(ExState);
private:
  void SetIntrinsicState(InState ins);
  InState inState;
};
//----< Flyweight Operation >--------------------------------

template <typename ExState, typename InState, size_t key>
void ConcreteFlyweight<ExState, InState, key>::
Operation(ExState exState)
{
  exState.Operation();
  inState.Operation();
}
```

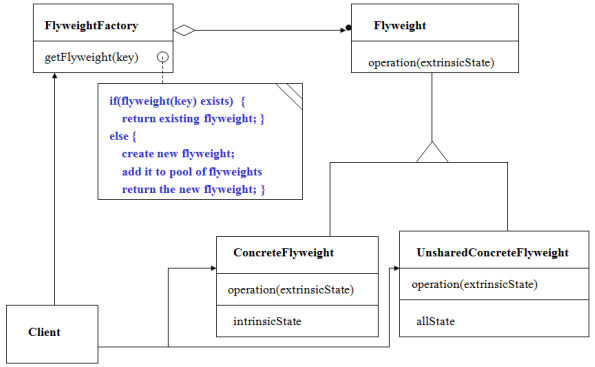
```
////////////////////////////////////////////////
// ConcreteUnSharedFlyweight

template <typename ExState, typename InState>
class ConcreteUnSharedFlyweight :
   public Flyweight<ExState, InState>
{
  friend class FlyweightFactory<ExState, InState>;
public:
  ~ConcreteUnSharedFlyweight() {
   std::cout << "\n  destroying ConcreteUnSharedFlyweight";
}
  void Operation(ExState);
  void SetIntrinsicState(InState);
private:
  InState inState;
  ExState exState;
};

//----< Flyweight Operation >-----------------------------

template <typename ExState, typename InState>
void ConcreteUnSharedFlyweight<ExState, InState>::
Operation(ExState state)
{
  state.Operation();
  inState.Operation();
}
```

```
////////////////////////////////////////////////////////////
// FlyweightFactory

template <typename ExState, typename InState>
class FlyweightFactory
{
public:
  FlyweightFactory() {};
  ~FlyweightFactory();
  Flyweight<ExState, InState>* GetFlyweigth(size_t mykey);
  ConcreteUnSharedFlyweight<ExState, InState>*
  GetUnSharedFlyweight();
private:
  std::map<size_t, Flyweight<ExState, InState>*> availableFWs;
};

//----< FlyweightFactory destructor >--------------------------

template <typename ExState, typename InState>
FlyweightFactory<ExState, InState>::~FlyweightFactory()
{
  std::cout << "\n  Destroying FlyweightFactory";
  std::map<size_t, Flyweight<ExState, InState>*>::
  iterator iter;
  for(iter=availableFWs.begin(); iter!=availableFWs.end(); ++iter)
    delete(iter->second);
  availableFWs.clear();
}
```

```
//----< get shared Flyweight from factory >----------------

template <typename ExState, typename InState>
Flyweight<ExState, InState>*
FlyweightFactory<ExState, InState>::GetFlyweigth(size_t mykey)
{
  std::map<size_t, Flyweight<ExState,InState>*>::iterator iter;
  switch(mykey)
  {
  case 0:
    iter = availableFWs.find(0);
    if(iter != availableFWs.end())
    {
      return iter->second;
    }
    else
    {
      ConcreteFlyweight<ExState, InState, 0>* pFlywt =
        new ConcreteFlyweight<ExState, InState, 0>();
      pFlywt->SetIntrinsicState(
        InState("Intrinsic state for all FWs of category #0")
      );
      availableFWs[0] = pFlywt;
      return pFlywt;
    }
```




```
case 1:
  iter = availableFWs.find(1);
  if(iter != availableFWs.end())
  {
    return iter->second;
  }
  else
  {
    ConcreteFlyweight<ExState, InState, 1>* pFlywt =
      new ConcreteFlyweight<ExState, InState, 1>();
    pFlywt->SetIntrinsicState(InState("Intrinsic state for all FWs of category #1"));
    availableFWs[1] = pFlywt;
    return pFlywt;
  }
case 2:
  iter = availableFWs.find(2);
  if(iter != availableFWs.end())
  {
    return iter->second;
  }
  else
  {
    ConcreteFlyweight<ExState, InState, 2>* pFlywt =
      new ConcreteFlyweight<ExState, InState, 2>();
    pFlywt->SetIntrinsicState(InState("Intrinsic state for all FWs of category #2"));
    availableFWs[2] = pFlywt;
    return pFlywt;
  }
default:
  return 0;
}
```

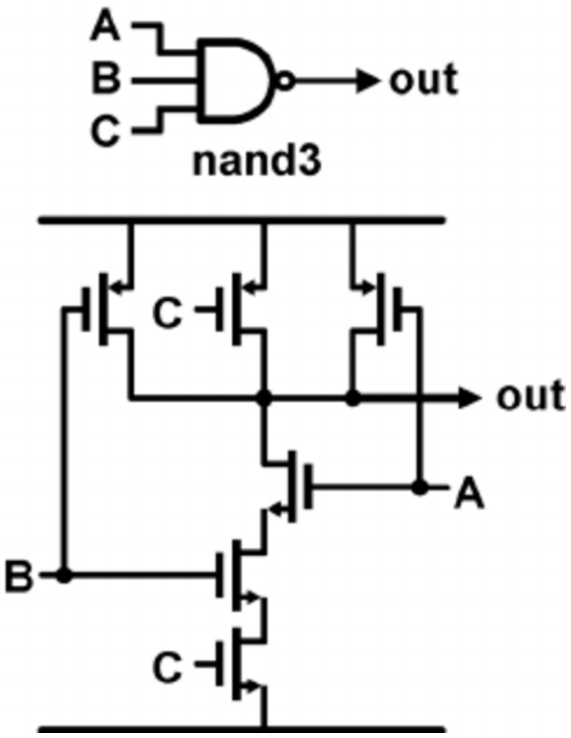



```
//----< test application's intrinsic (shared) state type >-----

class IntrinsicState
{
public:
  IntrinsicState() : myState("default") {};
  IntrinsicState(const std::string& str) : myState(str) {}
  void Operation() { std::cout << "\n  " << myState; }
private:
  std::string myState;
};
//---< test application's extrinsic (unshared) state type >----

class ExtrinsicState
{
public:
  ExtrinsicState() : myState("default") {}
  ExtrinsicState(const std::string& str) : myState(str) {}
  void Operation() { std::cout << "\n  " << myState; }
private:
  std::string myState;
};
```

```
 void main()
{
  std::cout << "\n  Demonstrating Basic Flyweight Pattern";
  std::cout << "\n ======================================\n";

  FlyweightFactory<ExtrinsicState, IntrinsicState> FWfactory;

  ////////////////////////////////////////////////////////////
  // Shared Flyweight operations
  //   Factory will delete these Flyweights

  Flyweight<ExtrinsicState, IntrinsicState>* pFW1 = FWfactory.GetFlyweigth(0);
  pFW1->Operation(ExtrinsicState("extrinsic state for FW1 of category #0"));
  std::cout << "\n";

  Flyweight<ExtrinsicState, IntrinsicState>* pFW2 = FWfactory.GetFlyweigth(0);
  pFW2->Operation(ExtrinsicState("extrinsic state for FW2 of category #0"));
  std::cout << "\n";

  Flyweight<ExtrinsicState, IntrinsicState>* pFW3 = FWfactory.GetFlyweigth(1);
  pFW3->Operation(ExtrinsicState("extrinsic state for FW3 of category #1"));
  std::cout << "\n";

  Flyweight<ExtrinsicState, IntrinsicState>* pFW4 = FWfactory.GetFlyweigth(1);
  pFW4->Operation(ExtrinsicState("extrinsic state for FW4 of category #1"));
  std::cout << "\n";

  ////////////////////////////////////////////////////////////
  // UnShared Flyweight operations
  //   Client must delete these Flyweights
  ConcreteUnSharedFlyweight<ExtrinsicState, IntrinsicState>* pFW5 =
    FWfactory.GetUnSharedFlyweight();
  pFW5->SetIntrinsicState(IntrinsicState("intrinsic state for FW5"));
  pFW5->Operation(ExtrinsicState("extrinsic state for unshared FW5"));
  delete pFW5;
  std::cout << "\n";
}
```

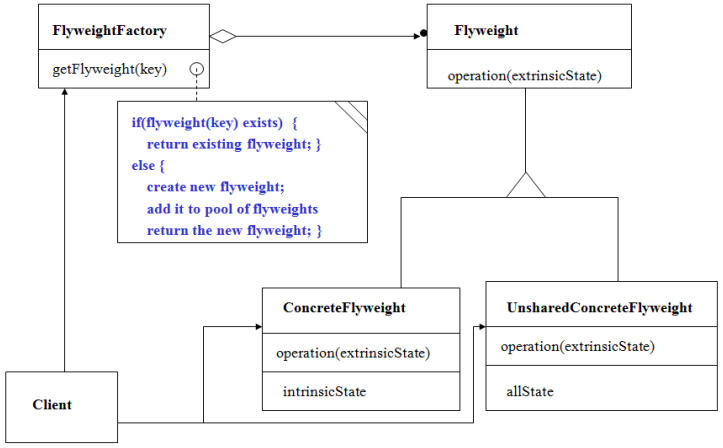
Application Code

- Digital Computer-Aided Design application
- Flyweights are gates
  - Intrinsic data are the lines and shapes necessary to render
  - Extrinsic data are the layout position and net list need to complete a logic schematic.



Flyweight classes

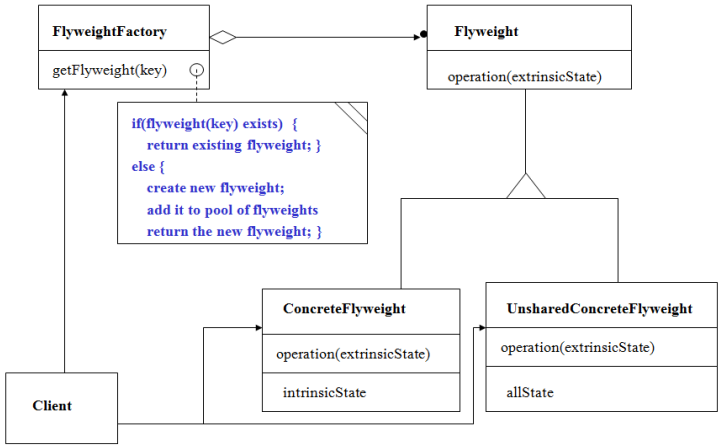
```
template <typename PlacementData, typename DrawingData>
class FW_CADFactory;

enum cellType { nandGate, norGate, latch, Register };

////////////////////////////////////////////////////////
// FW_CAD

template <typename PlacementData, typename DrawingData>
class FW_CAD
{
public:
  virtual ~FW_CAD() {}
  virtual void Operation(PlacementData)=0;
};
```




```
////////////////////////////////////////////////////////
// ConcreteFW_CAD

template <typename PlacementData, typename DrawingData, size_t cellType>
class ConcreteFW_CAD : public FW_CAD<PlacementData, DrawingData>
{
  friend class FW_CADFactory<PlacementData, DrawingData>;
public:
  ConcreteFW_CAD() {}
  ~ConcreteFW_CAD() { std::cout << "\n  destroying concrete FW_CAD"; }
  void Operation(PlacementData);
private:
  void SetIntrinsicState(DrawingData ins);
  DrawingData inState;
};
//----< FW_CAD Operation >--------------------------------

template <typename PlacementData, typename DrawingData, size_t cellType>
void ConcreteFW_CAD<PlacementData, DrawingData, cellType>::
Operation(PlacementData exState)
{
  exState.Operation();
  inState.Operation();
}
```

Flyweight classes

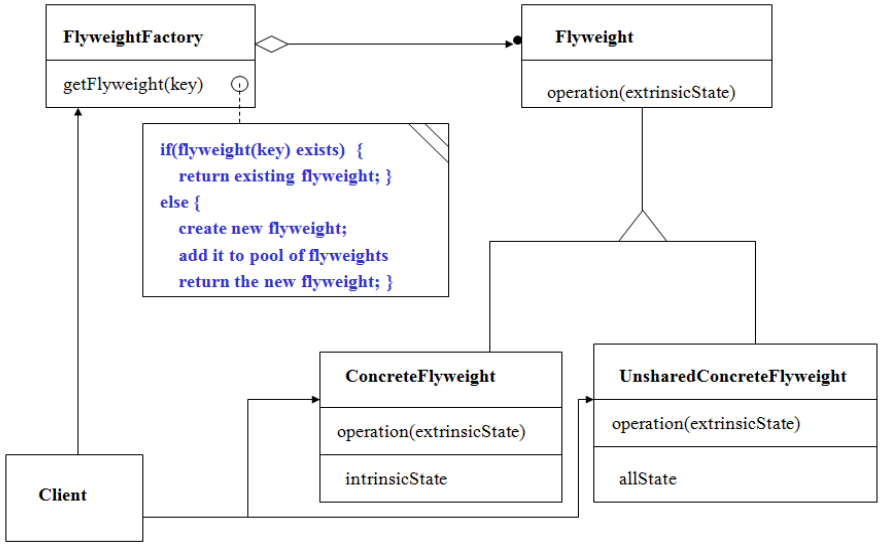
```
//----< Set Intrinsic state >-------------------------

template <
  typename PlacementData,
  typename DrawingData,
  size_t cellType
>
void ConcreteFW_CAD<
  PlacementData, DrawingData, cellType
>
::SetIntrinsicState(DrawingData state)
{
  inState = state;
}
```




```
////////////////////////////////////////////////////////////
// FW_CADFactory

template <typename PlacementData, typename DrawingData>
class FW_CADFactory
{
public:
  FW_CADFactory() {};
  ~FW_CADFactory();
  FW_CAD<PlacementData, DrawingData>* GetFlyweigth(size_t mycellType);
  ConcreteUnSharedFW_CAD<PlacementData, DrawingData>* GetUnSharedFW_CAD();
private:
  std::map<size_t, FW_CAD<PlacementData, DrawingData>*> availableFWs;
};
//----< FW_CADFactory destructor >-------------------------

template <typename PlacementData, typename DrawingData>
FW_CADFactory<PlacementData, DrawingData>::~FW_CADFactory()
{
  std::cout << "\n  Destroying FW_CADFactory";
  std::map<size_t, FW_CAD<PlacementData, DrawingData>*>::iterator iter;
  for(iter=availableFWs.begin(); iter!=availableFWs.end(); ++iter)
    delete(iter->second);
  availableFWs.clear();
}
```

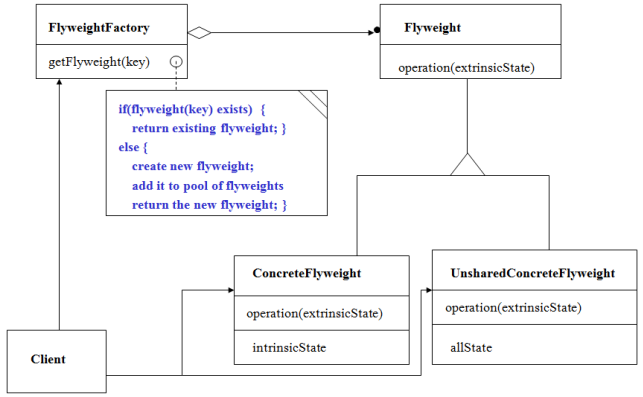
Flyweight Factory

```
template <typename PlacementData, typename DrawingData>
FW_CAD<PlacementData, DrawingData>*
FW_CADFactory<PlacementData, DrawingData>::
GetFlyweigth(size_t mycellType)
{
  std::map<size_t, FW_CAD<PlacementData,DrawingData>*>::
  iterator iter;
  switch(mycellType)
  {
```




```
case nandGate:
   iter = availableFWs.find(nandGate);
   if(iter != availableFWs.end())
   {
     return iter->second;
   }
   else
   {
     std::cout << "\n  -- Loading nandGate drawing list from XML file --";
     ConcreteFW_CAD<PlacementData, DrawingData, nandGate>* pFlywt =
       new ConcreteFW_CAD<PlacementData, DrawingData, nandGate>();
     pFlywt->SetIntrinsicState(
       DrawingData("Using nandGate Cell graphics list")
     );
     availableFWs[nandGate] = pFlywt;
     return pFlywt;
   }
```

Flyweight Factory




```
case norGate:
 iter = availableFWs.find(norGate);
 if(iter != availableFWs.end())
 {
   return iter->second;
 }
 else
 {
   std::cout <<
   "\n  -- Loading norGate drawing list from XML file --";
   ConcreteFW_CAD<PlacementData, DrawingData, norGate>*
   pFlywt =
   new ConcreteFW_CAD<PlacementData, DrawingData, norGate>();
   pFlywt->SetIntrinsicState(
     DrawingData("Using norGate Cell graphics list")
   );
   availableFWs[norGate] = pFlywt;
   return pFlywt;
 }
```

```
case latch:
  iter = availableFWs.find(latch);
  if(iter != availableFWs.end())
  {
    return iter->second;
  }
  else
  {
    std::cout << "\n  -- Loading latch drawing list from XML file --";
    ConcreteFW_CAD<PlacementData, DrawingData, latch>* pFlywt =
      new ConcreteFW_CAD<PlacementData, DrawingData, latch>();
    pFlywt->SetIntrinsicState(
      DrawingData("Using latch Cell graphics list")
    );
    availableFWs[latch] = pFlywt;
    return pFlywt;
  }
default:
  return 0;
}
```

Application Code

```
//----< test application's intrinsic (shared) state type >-----

class DrawingDataFromFile
{
public:
  DrawingDataFromFile() : myState("default") {};
  DrawingDataFromFile(const std::string& str) : myState(str) {}
  void Operation() { std::cout << "\n  " << myState; }
private:
  std::string myState;
};
//---< test application's extrinsic (unshared) state type >----

class PlacementDataFromCAD_Editor
{
public:
  PlacementDataFromCAD_Editor() : myState("default") {}
  PlacementDataFromCAD_Editor(const std::string& str) :
    myState(str) {}
  void Operation() { std::cout << "\n  " << myState; }
private:
  std::string myState;
};
```

```
void main()
{
  std::cout << "\n  Demonstrating Basic FW_CAD Pattern";
  std::cout << "\n =====================================\n";

  FW_CADFactory<PlacementDataFromCAD_Editor, DrawingDataFromFile>
  FWfactory;

  ////////////////////////////////////////////////////////////
  // Shared FW_CAD operations
  //   Factory will delete these FW_CADs

  FW_CAD<PlacementDataFromCAD_Editor, DrawingDataFromFile>*
  pFW1 = FWfactory.GetFlyweigth(nandGate);
  pFW1->Operation(
    PlacementDataFromCAD_Editor("Draw nandGate at (120,245)")
  );
  std::cout << "\n";

  FW_CAD<PlacementDataFromCAD_Editor, DrawingDataFromFile>*
  pFW2 = FWfactory.GetFlyweigth(nandGate);
  pFW2->Operation(
    PlacementDataFromCAD_Editor("Draw nandGate at (250,245)")
  );
  std::cout << "\n";

      --- rest elided ---
```

Application Output

```
 Demonstrating Basic FW_CAD Pattern
====================================

 -- Loading nandGate drawing list from XML file --
 Draw nandGate at (120,245)
 Using nandGate Cell graphics list

 Draw nandGate at (250,245)
 Using nandGate Cell graphics list

 Draw nandGate at (530,200)
 Using nandGate Cell graphics list

 Draw nandGate at (750,245)
 Using nandGate Cell graphics list

 -- Loading norGate drawing list from XML file --
 Draw norGate at (350, 220)
 Using norGate Cell graphics list

 Draw norGate at (120, 400)
 Using norGate Cell graphics list

 Draw norGate at (620, 220)
 Using norGate Cell graphics list

 -- Loading latch drawing list from XML file --
 Draw latch at (300, 400)
 Using latch Cell graphics list


 Destroying FW_CADFactory
 destroying concrete FW_CAD
 destroying concrete FW_CAD
 destroying concrete FW_CAD


C:\su\CSE776\Presentations-Fawcett\Flyweight\DigitalCADapp\Deb
Press any key to close this window . . .
```

- Flyweights:
  - nandGate, norGate, Latch
- Intrinsic data:
  - Lines and fills read from XML file or database
- Extrinsic data:
  - Position passed to draw (operation)
  - Connections drawn from netlist.
- Potentially large savings in memory and processing
  - Layout and netlist have to be computed anyway, so we just use them for editor rendering.