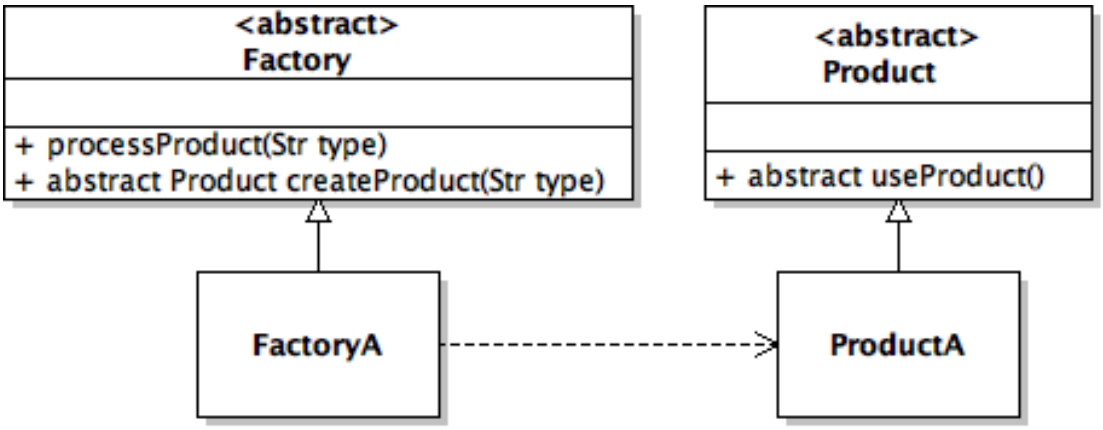
CECS 277 – Lecture 20 – Design Patterns

The Factory Method

The Factory Method minimizes dependencies by removing the process of creating objects from the client and replaces it with a factory interface for creating those objects instead. Encapsulating the factory makes it so that each part is easily modified when needed. Since all parts of the product creation process are located in the factory, whenever new Product classes are written, they can be easily added to the factory. If the client needs a different type of factory, it can be quickly switched out for another.

Factory UML – The Factory pattern is made up of four parts: An abstract base class Factory, from which, one or more types of factories are made. Concrete Factory classes each override a creation method that decides which products this factory makes and it returns a newly constructed product based on the input type. A Product interface (or abstract class) that defines what products can do. And a concrete Product class for each type of product that the factory makes.



Factory Template Code:

```
public interface Product {
   public void useProduct( );
}

public class ProductA implements Product {
   @Override
   public void useProduct( ) {
      System.out.println( "Using Product A" );
   }
}

public class ProductB implements Product {
   @Override
   public void useProduct( ) {
      System.out.println( "Using Product B" );
   }
}
```

```
public abstract class Factory {
   public void processProduct( String type ) {
      Product p = createProduct( type );
      p.useProduct();
   }
   public abstract Product createProduct( String type );
}

public class FactoryA extends Factory {
   @Override
   public Product createProduct( String type ) {
      if( type.equals( "A" ) ) {
         return new ProductA( );
      } else if( type.equals( "B" ) ) {
         return new ProductB( );
      }
      return null;
   }
}

public class Main {
   public static void main( String [] args ) {
      Factory a = new FactoryA( );
      a.processProduct( "A" );
      a.processProduct( "B" );
   }
}
```

Factory Example Code:

```
public interface Shape {
   public void draw( );
}

public class Circle implements Shape {
   @Override
   public void draw( ) {
      System.out.println( "Drawing Circle" );
   }
}

public class Square implements Shape {
   @Override
   public void draw( ) {
      System.out.println( "Drawing Square" );
   }
}
```

```
public abstract class Factory {
   public void drawShape( String type ) {
      Shape s = createShape( type );
      s.draw();
   }
   public abstract Shape createShape( String type );
}
public class ShapeFactory extends Factory {
   @Override
   public Shape createShape( String type ) {
      if( type.equals( "Circle" ) ) {
         return new Circle( );
      } else if( type.equals( "Square" ) ) {
         return new Square( );
      }
      return null;
   }
}
public class Main {
   public static void main( String [] args ) {
      Factory f = new ShapeFactory( );
      f.drawShape( "Circle" );
      f.drawShape( "Square" );
   }
}
```