Design Patterns

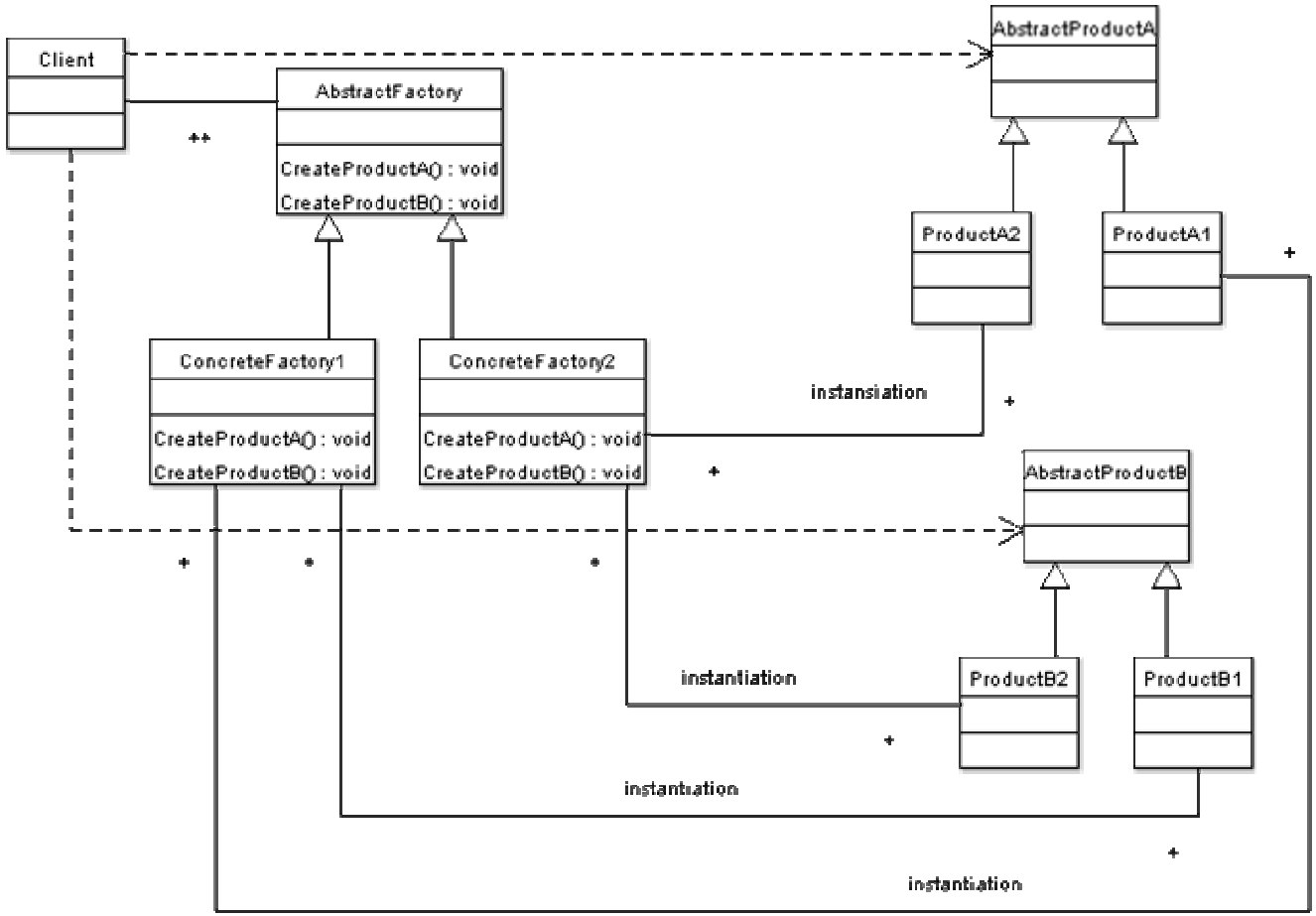
ECE355 Tutorial

Ali Razavi

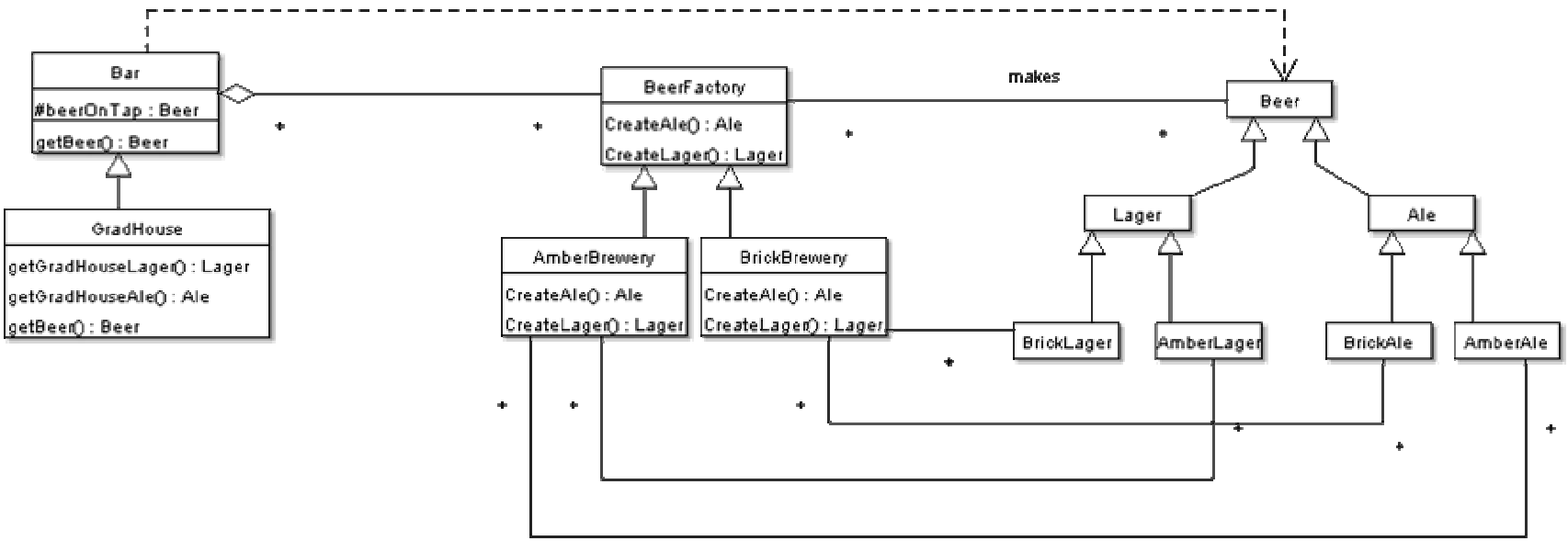
AbstractFactory – Quick Review

- Intent: Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- Applicability:
  - A system should be independent of how its products are created, composed and represented.
  - A system should be configured with one of multiple families of products
  - A family of related product objects is designed to be together and this constraint needs to be enforced
  - You want to provide a class library of products and you want to reveal just their interfaces and not their implementations

Abstract Factory – Quick Review



AbstractFactory – Example 1 Brewery



Beer Implementation in Java

```
public abstract class Beer {
}

public abstract class Lager extends Beer {

}

public abstract class Ale extends Beer {
}

public class BrickLager extends Lager implements BrickBeer {
}

public class BrickAle extends Ale implements BrickBeer {
}

public class AmberLager extends Lager implements AmberBeer {
}

public class AmberAle extends Ale implements AmberBeer {
}
```

Brewery Implementation in Java

```
public abstract class BeerFactory {

        Lager CreateLager() {
                // default Lager production process
        }

        Ale   CreateAle() {
                // default Ale production process
        }

}

public class BrickBrewery extends BeerFactory {

        Lager CreateLager() {

                BrickLager bricklager = super.CreateLager();
                // Brick Specialization on Brick Lager
                return bricklager;
        }

        Ale CreateAle() {
                BrickAle brickale = super.CreateAle();
                // Brick Specialization on Brick Ale
                return brickale;
        }
}

public class AmberBrewery extends BeerFactory {

        Lager CreateLager() {

                AmberLager amberlager = super.CreateLager();
                //Amber Specialization on Amber Lager
                return amberlager;
        }

        Ale CreateAle() {

                AmberAle amberale = super.CreateAle();
                //Amber Specialization on Amber Ale
                return amberale;
        }
}
```

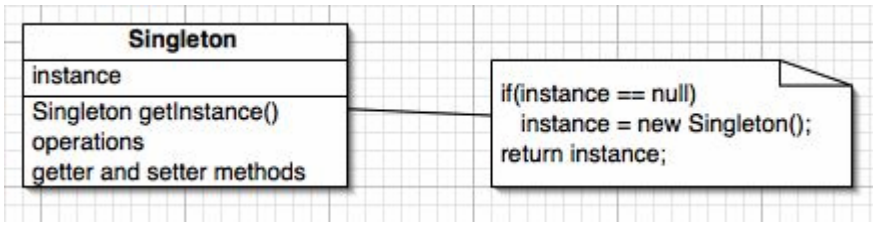
Bar Implementation in Java

```
public abstract class Bar {

        protected Beer beerOnTap;

        public Beer getBeer() {

                return beerOnTap;
        }
}

public class GradHouse extends Bar implements UWCampus, GSA {

        BeerFactory brewery;
        public GradHouse {

           brewery = new BrickBrewery();
        }

        public Lager GetGradHouseLager() {

                return brewery.CreateLager();
        }

        public Ale GetGradHouse Ale() {
                return brewery.CreateAle();
        }
}
```

Singleton – Quick Review

- Intent : Ensure a class only has one instance, and provide a global point of access to it.
- Applicability:
  - There must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
  - When the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code.

Singleton in Java

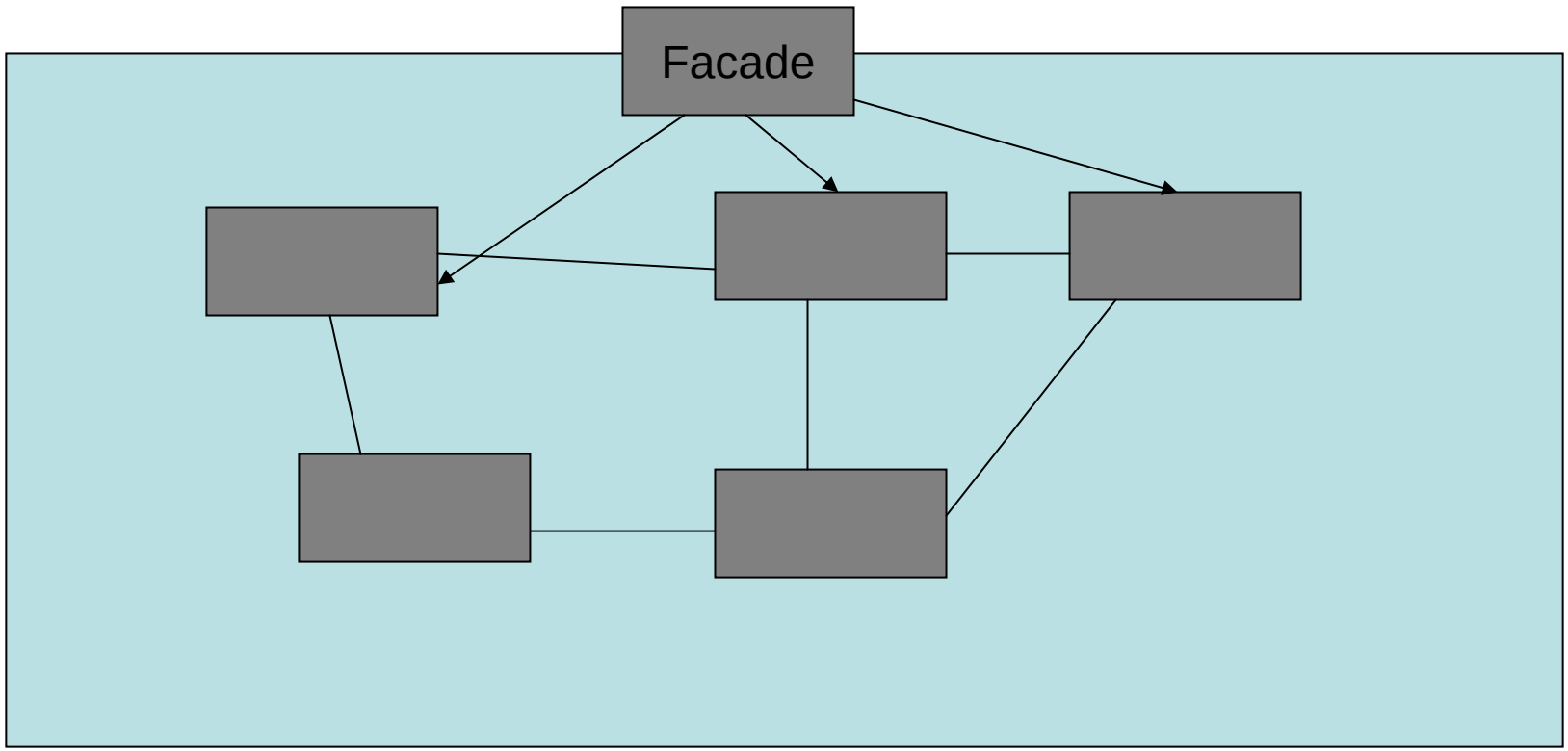



```
public class ClassicSingleton {
   private static ClassicSingleton instance = null;

   protected ClassicSingleton() {
      // Exists only to defeat instantiation.
   }
   public static ClassicSingleton getInstance() {
      if(instance == null) {
         instance = new ClassicSingleton();
      }
      return instance;
   }
}
```

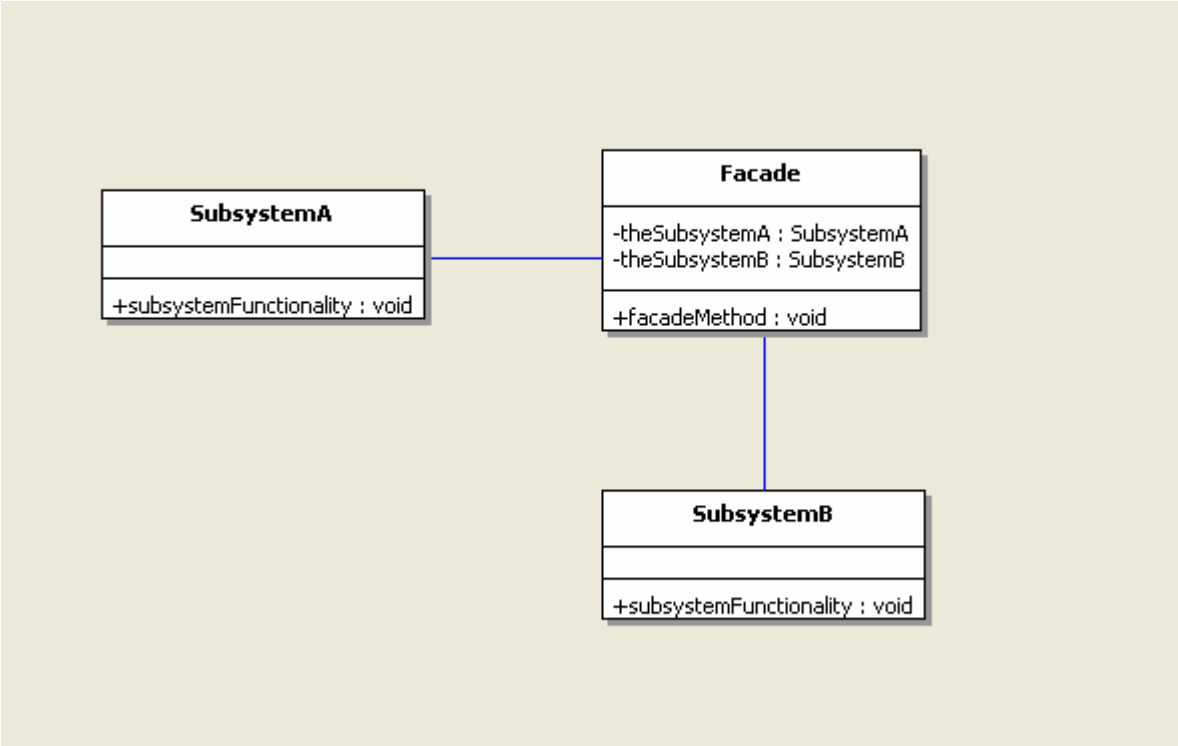
Facade

- Intent: Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
- Applicability:
  - A simple interface for complex subsystems
  - There are many dependencies between clients and the implementation classes of an abstraction
  - You want to layer your subsystems

Facade structure



Facade in Java



Facade Example