## Collections

<https://www.codeproject.com/Articles/832189/List-vs-IEnumerable-vs-IQueryable-vs-ICollection-v>

## stack vs heap memory

## Delegate amd multi cast delegate

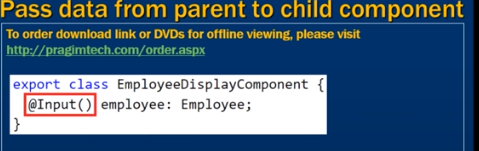
## Abtraction v/s encapsulation

## How to pass from parent to child component

<https://www.c-sharpcorner.com/blogs/update-the-child-component-and-parent-component-using-input-output-in-angular-4>

pass data from a parent component to a child directive or component using the @Input() decorator in the child component/directive.

To let Angular know that a property in a child component or directive can receive its value from its parent component we must use the @Input() decorator in the said child.



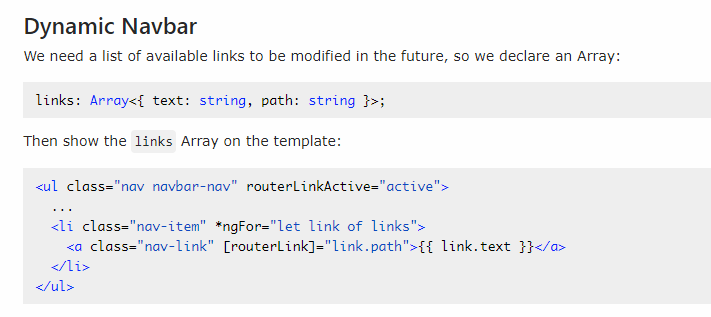
# Sharing Data from Child to Parent in Angular 8 Using @viewchild

<https://dzone.com/articles/sharing-data-from-child-to-parent-in-angular-8-usi>



## How to pass from parent to child grid

## How to dynamically implemnt naviagtion bar in angular



## What is docker and containers

## What is kubernates

## How to host web api and angular apps in azure

## Design Patterns

## Design pattern for toggle features

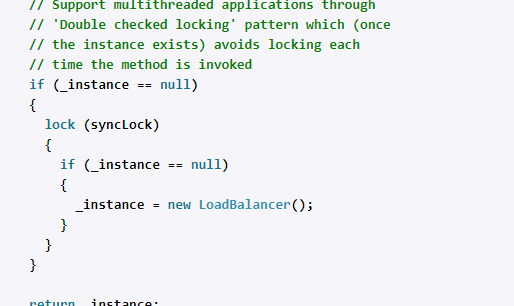
<https://www.c-sharpcorner.com/UploadFile/questpond/design-pattern-interview-questions/>

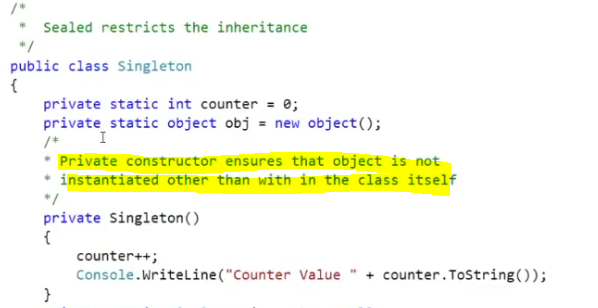
## Which are the three main categories of design patterns?

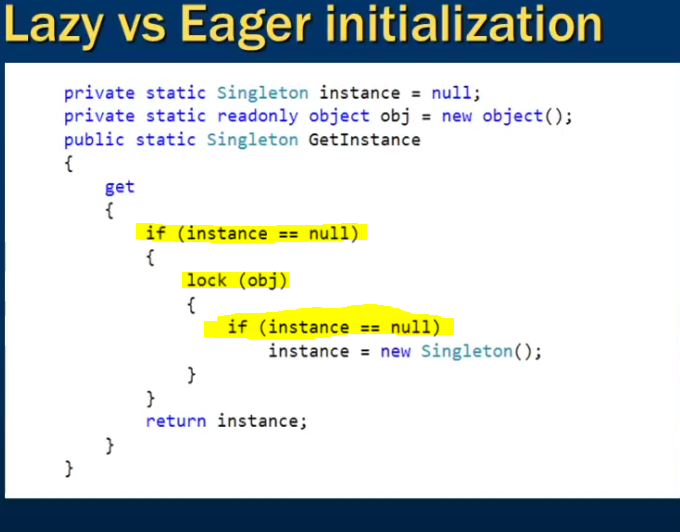
There are three basic classifications of patterns Creational, Structural, and Behavioral patterns.  
***Creational Patterns***  
  
Abstract Factory : Creates an instance of several families of classes  
Builder : Separates object construction from its representation  
Factory Method : Creates an instance of several derived classes  
Prototype : A fully initialized instance to be copied or cloned  
Singleton : A class in which only a single instance can exist  
 **Note**   
  
The best way to remember Creational pattern is by remembering ABFPS (Abraham Became First President of States).  
  
***Structural Patterns***  
Adapter : Match interfaces of different classes .  
Bridge : Separates an object's abstraction from its implementation.  
Composite : A tree structure of simple and composite objects.  
Decorator : Add responsibilities to objects dynamically.  
Flyweight : A fine-grained instance used for efficient sharing.  
Proxy : An object representing another object.  
  
**Note**  
To remember structural pattern best is (ABCDFFP)  
  
***Behavioral Patterns***  
Mediator : Defines simplified communication between classes.  
Memento : Capture and restore an object's internal state.  
Interpreter : A way to include language elements in a program.  
Iterator : Sequentially access the elements of a collection.  
Chain of Resp : A way of passing a request between a chain of objects.  
Command : Encapsulate a command request as an object.  
State : Alter an object's behavior when its state changes.  
Strategy : Encapsulates an algorithm inside a class.  
Observer : A way of notifying change to a number of classes.  
Template Method : Defer the exact steps of an algorithm to a subclass.  
Visitor : Defines a new operation to a class without change.

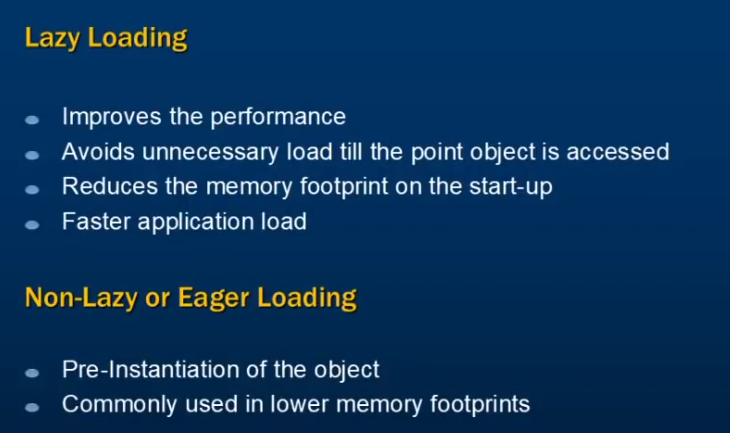
Singleton

<https://www.dofactory.com/net/singleton-design-pattern>

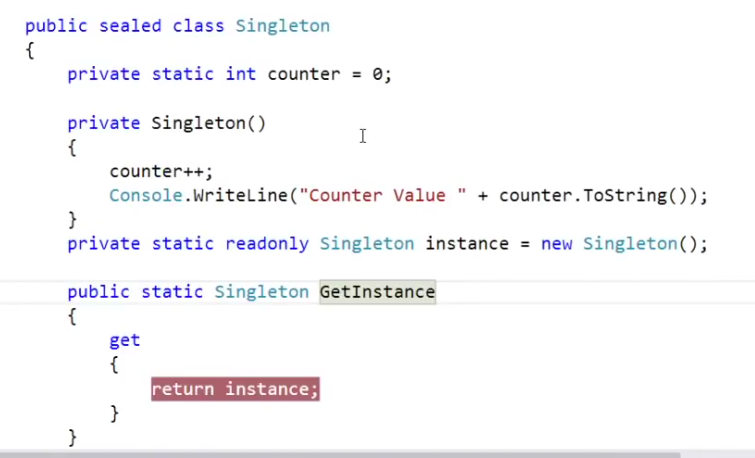


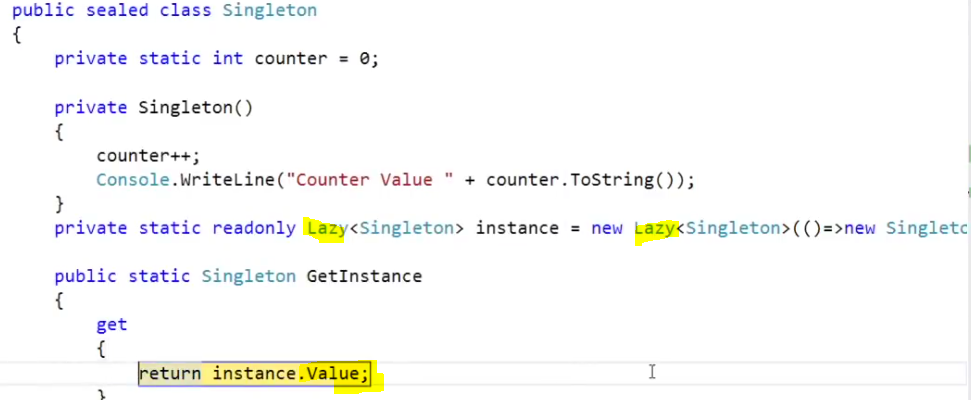




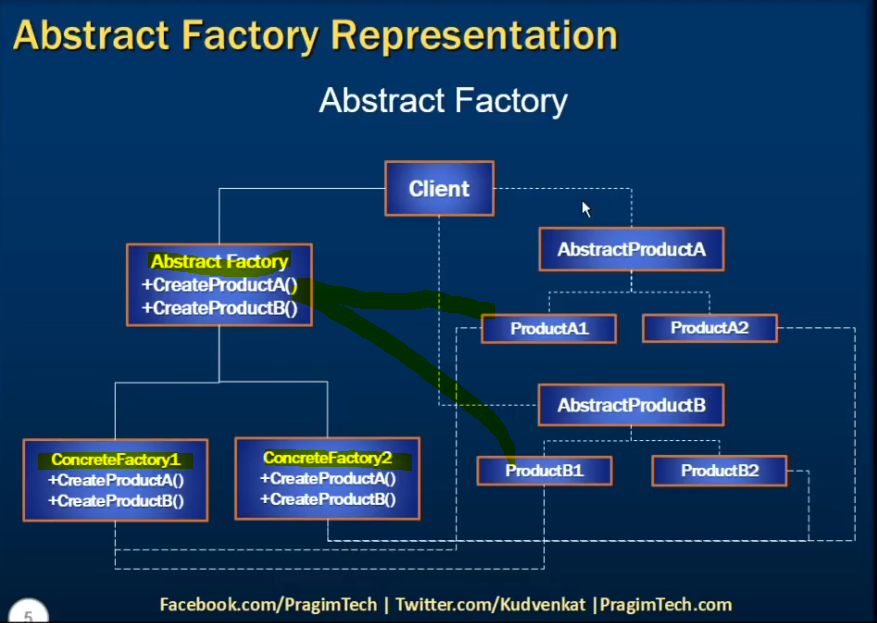


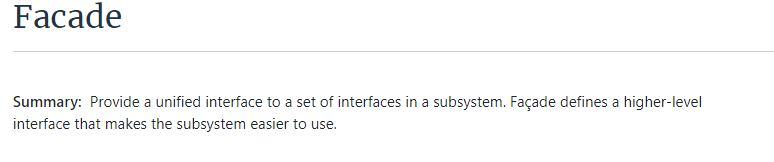
Non Lazy/Eager Loading











<https://www.dofactory.com/net/facade-design-pattern>

### Calling Store proc in EF

1. / Creating Custom class to hold result of Stored Procedure
2. **public** **class** EmployeeDetail
3. {
4. **public** **int** EmployeeID { get; set; }
5. **public** string EmployeeName { get; set; }
6. **public** string DepartmentName { get; set; }
7. }
9. // using Object Context (EF4.0)
10. using (Entities context = **new** Entities())
11. {
12. IEnumerable<EmployeeDetails> empDetails  =  context.ExecuteStoreQuery<EmployeeDetails>
13. ("exec GetEmployeeData").ToList();
14. }
16. // using DBContext (EF 4.1 and above)
17. using (Entities context = **new** Entities())
18. {
19. IEnumerable<EmployeeDetails> empDetails  =  context. Database.SqlQuery
20. < EmployeeDetails >("exec GetEmployeeData ", **null**).ToList();
21. }