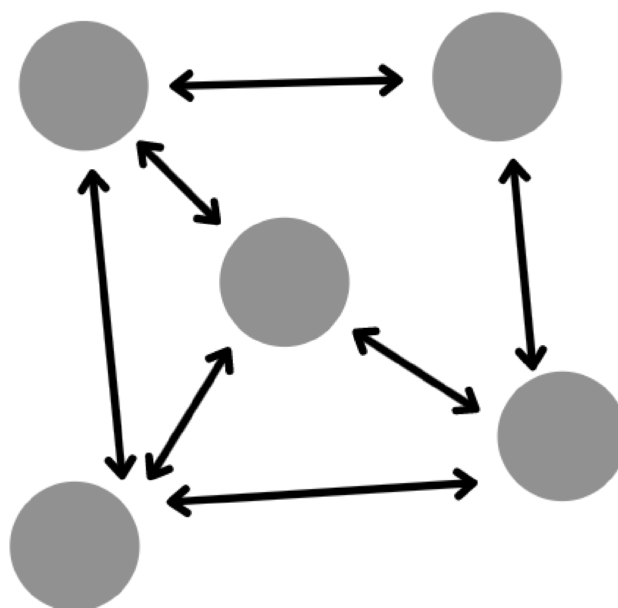
36. What is cohesion?

Brief summary: Cohesion is the degree to which elements of a module belong together. In simpler words, it measures how strong the relationship is between members of a class. High cohesion is a desirable trait of the classes and modules.

Cohesion is the degree to which elements of a module belong together. In simpler words, it measures how strong the relationship is between members of this class or module. The closer related the members of a class are, the better.

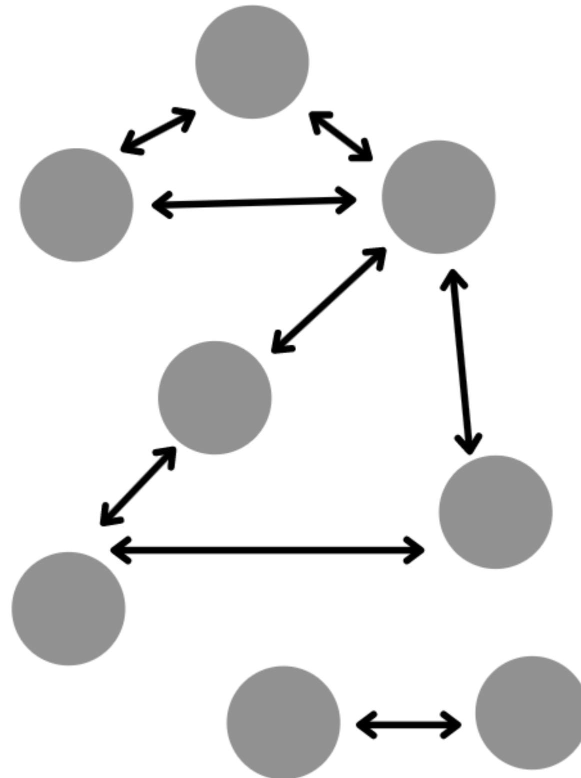
High cohesion is a desirable trait of classes and modules.

This illustrates a highly cohesive class or module:

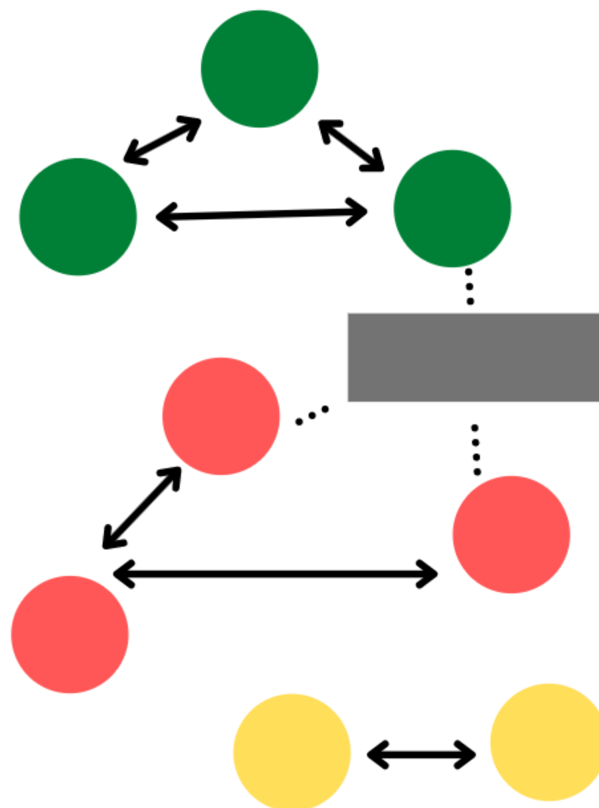


As you can see no piece seems to be "lonely". There is a lot of connections between them, and it would be hard to draw any line in which this module could be split.

Now, let's see a module that's not cohesive:



In this case, there are some pieces that seem to have very little or nothing to do with others. We can easily see how this module could be divided into highly-cohesive modules:



As you can see I’ve kept the connection between the green and red parts, but I made it go through some abstraction - in C# this would most likely be an interface.

Let’s see some code now:

```
class PetsCollection
{
    private List<Pet> _pets = new ();

    public void Add(Pet pet) => _pets.Add(pet);
    public int Count => _pets.Count;
    public IEnumerable<PetType> GetCurrentlyStoredTypes() =>
        _pets.Select(pet => pet.PetType).Distinct();
    public bool Contains(PetType petType) =>
        GetCurrentlyStoredTypes().Any(type => type == petType);
}
```

This class is characterized by **high cohesion**. All those methods use the underlying collection called _pets. The Contains method uses the GetCurrentlyStoredTypes method. None of those methods could be easily moved away from this class. There is no easy way to split this class into separate classes, nor it would make much sense, as those methods naturally belong together.

Now, let's consider a different class:

```
public class HousePricer
{
    private IOwnersDatabase _ownersDatabase;
    private decimal _dollarsPerSquareMeter;
    public HousePricer(
        decimal dollarsPerSquareMeter,
        IOwnersDatabase ownersDatabase)
    {
        _dollarsPerSquareMeter = dollarsPerSquareMeter;
        _ownersDatabase = ownersDatabase;
    }

    public decimal GetPrice(House house) =>
        _dollarsPerSquareMeter * (decimal)house.Area *
        GetPriceMultiplierBasedOnFloors(house.Floors);

    private decimal GetPriceMultiplierBasedOnFloors(int floors) =>
        floors switch { 1 => 1m, 2 => 1.5m, _ => 1.6m };

    public void SendPriceToOwner(House house) =>
        Console.WriteLine($"Sending price {GetPrice(house)}" +
            $" to {FindOwnerEmail(house.Address)}");

    private string FindOwnerEmail(string address) =>
        _ownersDatabase.GetEmailByAddress(address);
}
```

This class is **not** cohesive. It has two quite separate responsibilities. First, it evaluates a price of a house, and second, it notifies the owner about the calculated price. The only point where those two responsibilities meet is that the SendPriceToOwner method needs the information about the price, but this is something that can be easily refactored.

Let’s create two highly-cohesive classes:

```
public class HousePricer
{
    private decimal _dollarsPerSquareMeter;
    public HousePricer(decimal dollarsPerSquareMeter)
    {
        _dollarsPerSquareMeter = dollarsPerSquareMeter;
    }

    public decimal GetPrice(House house) =>
        _dollarsPerSquareMeter * (decimal)(house.Area) *
        GetPriceMultiplierBasedOnFloors(house.Floors);

    private decimal GetPriceMultiplierBasedOnFloors(int floors) =>
        floors switch { 1 => 1m, 2 => 1.5m, _ => 1.6m };
}
```

```
public class OwnerNotifier
{
    private IOwnersDatabase _ownersDatabase;
    public OwnerNotifier(IOwnersDatabase ownersDatabase)
    {
        _ownersDatabase = ownersDatabase;
    }

    public void SendToOwner(string information, string address) =>
        Console.WriteLine($"Sending {information}" +
            $" to {FindOwnerEmail(address)}");

    private string FindOwnerEmail(string address) =>
        _ownersDatabase.GetEmailByAddress(address);
}
```

Now we can simply use them one after another:

```
var housePricer = new HousePricer(2000);
var price = housePricer.GetPrice(house);
var ownerNotifier = new OwnerNotifier(ownersDatabase);
ownerNotifier.SendToOwner(price.ToString(), house.Address);
```

By now you may probably be thinking "Oh, so high cohesion and Single Responsibility Principle are the same things?". Well, no, but it's common that a highly cohesive class meets the SRP and vice versa.

High cohesion means that the data and methods that belong together, are kept together. If following only the SRP, we **could** (but it doesn't mean we should!) keep splitting classes into smaller pieces until every class would have only one public method. Each of those tiny classes would definitely meet the SRP, as they would only have a single responsibility and single reason to change. But they wouldn't be cohesive, as they should belong together.

But, does it mean we should do it?

Well, no! Imagine what would happen if the List class was split into tiny classes, like ListAdder, ListRemover, ListClearer, ListCountGetter, etc. That would be unmaintainable and hard to understand. Now all those methods - Add, Remove, Clear and the Count property belong to a highly-cohesive List class. This class is focused on providing a generic, dynamic collection, and this is its responsibility. It still meets the SRP, because it has one reason to change - it will change if the idea of how such collection structure should be represented in C# changes.

If you want to read more about the relation between the SRP and high cohesion, I recommend this thread on Stack Overflow:
https://stackoverflow.com/questions/11215141/is-high-cohesion-a-synonym-for-the-single-responsibility-principle

High cohesion is not something we should create. It's something we observe and our job is not to break it. So how to recognize high cohesion?

High cohesion	Low cohesion
most or all members use the same private data and they reuse member methods	some private members are used by a group of members only; other members are used by a different group
the functionalities of a class have much in common	the functionalities of a class are unrelated
the class would be hard to split - if we did it, a lot of private data would need to be passed from one part to another	the class is easy to split and the line of splitting is natural and obvious
class is easy to name and its name is accurate	class is hard to name precisely or its name lies about what it does

If you see high cohesion - don't break it. High cohesion gives us a lot of benefits:

- Highly cohesive classes are easier to understand and use. They provide a highly-focused set of operations instead of more functionality than we need. Think of our OwnerNotifier class - it could easily be reused to send some other information to the person living at some address.
- When a change is needed, it's easier to introduce, as it affects fewer modules.
- Cohesive classes are easy to test.
- They are reusable.

On the other hand, when you see a class that is not highly cohesive, consider refactoring it and splitting it into highly-cohesive pieces.

Let's summarize. Cohesion is the degree to which elements of a module belong together. In simpler words, it measures how strong the relationship is between members of a class. High cohesion is a desirable trait of the classes and modules.

Bonus questions:

- "**Is following the Single Responsibility Principle and keeping high cohesion the same thing?**"
  No, but it's common that a highly cohesive class meets the SRP and vice versa. High cohesion means that the data and methods that belong together, are kept together. If following only the SRP, we ***could*** *(but it doesn't mean we should!) keep splitting classes into smaller pieces until every class would have only one public method. Each of those tiny classes would definitely meet the SRP, as they would only have a single responsibility and single reason to change. But they wouldn't be cohesive, as they should belong together.*