**Dynamic types**

C# 4.0 (.NET 4.5) introduced a new type called dynamic that avoids compile-time type checking. A dynamic type escapes type checking at compile-time; instead, it resolves type at run time.

dynamic a = 3;

a = "hello";

The last one is gonna be taken as a type which is a string in our case.

Int a =3; is a static type

The difference between generic types and dynamic types is that generic

types are resolved at compile time however dynamic types are decided at runtime.

**Constructor**

List<int> x = new List<int>();

Here we initialize parameterless constructor.

List<int> y = new List<int> {1,3};

Here we don’t need a parameterless constructor.

**Method Signature**

It includes The return type, the name, and parameters.

Public int GetName(bool isGraduated)

**S.O.L.I.D. Principles**

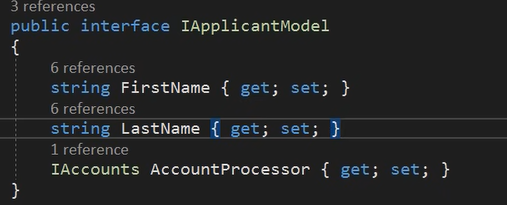
SOLID is one of the most popular sets of design principles in object-oriented software development.

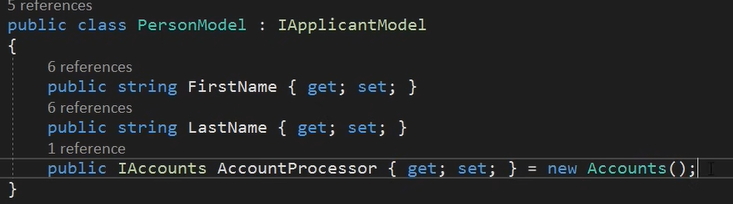
Single Responsibility Principle - Robert C. Martin describes it as: A class should have one, and only one, reason to change. Meaning that one class should have just one responsibility and not more.



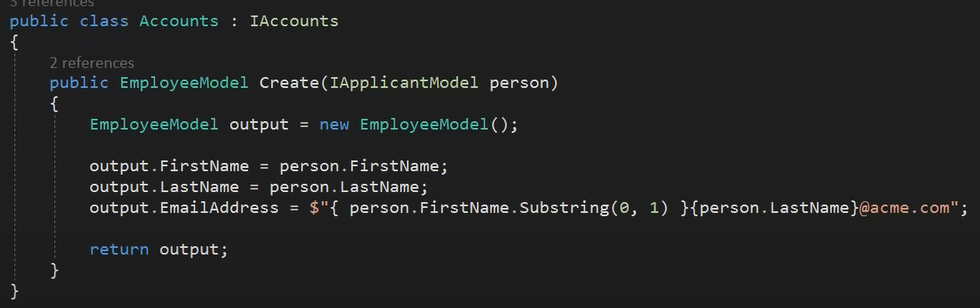
For instance, here this Program class has just one responsibility which is to control the flow of the application. And all the other responsibilities such as Capturing Person’s first name and last name or Messages(StandardMessage) are handled by other classes so it is their responsibility. So we should separate everything. It is not a problem to have many classes. Each of them has their own responsibility. Of any of the classes that we create if one of them has a scroll then we have done something wrong because they gotta be short. Only Main( ) method ,of course, can be a little bit bigger.

Open-Closed Principle - The Open-Closed Principle (OCP) states that software entities (classes, modules, methods, etc.) should be open for extension, but closed for modification. For example, let’s say that we have different models which are just classes: Person, ManagerModel, ExecutiveModel, and also we have different accounts for this models: Accounts, ManagerAccounts, ExecutiveAccounts. So to implement OCP we need to create a IApplicantModel which will be inherited by Person, ManagerModel, ExecutiveModel. And for accounts we will also create an interface (Accounts, ManagerAccounts, ExecutiveAccounts)🡪

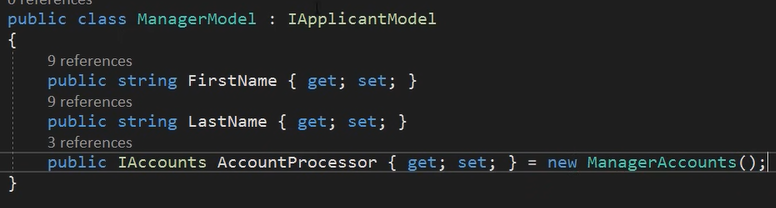




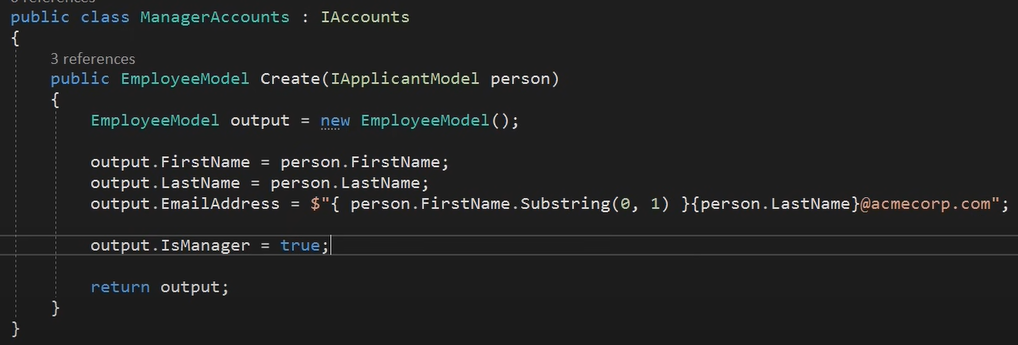
Accounts is a class for normal employees (that are just employees not managers or etc.).

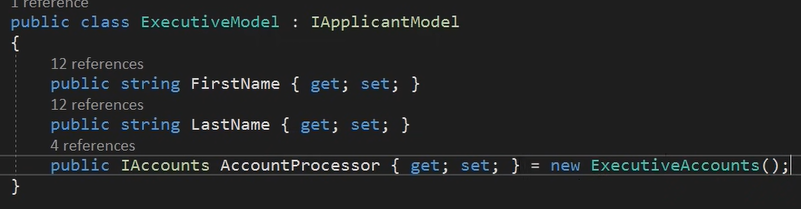


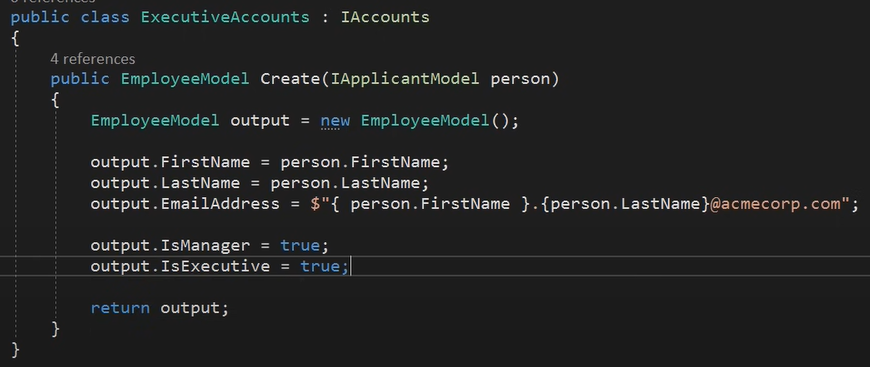
This is for normal employees.



This model is for Managers.



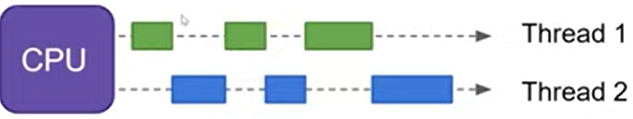


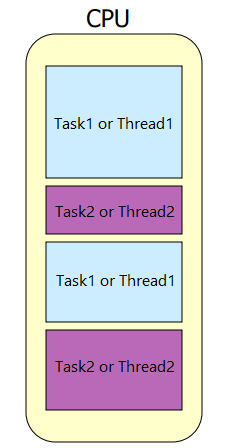


So in the end, if we we wanted to add a new model, for instance, TechnicianModel, then we would create a TechnicianModel class that would inherit from IApplicantModel and we would create Accounts for this Model (TechnicianAccounts) which would be somewhat different. And IAccounts for TechnicianAccounts would point to TechnicianAccounts.

**Concurrency and Parallelism**

Concurrency- Making progress on more that one task –seemingly at the same time. But this is actually happining one at a time, meaning that firt we do a little bit of task 1 or thread 1 then we go to task 2 or thread 2 then again we go back to task 1 or thread1 to do a little bit of that





Async,await tasks are concurrent.

Paralellism is exactly what we think of this, doing multiple things at the same time independently.

