مراحل ایجاد یک پروژه کامپیوتری:

مثل برنامه حسابداری

مثل برنامه انبارداری

حداقل ۶۵ تا ۷۵ درصد

تیم شناخت

تیم تحلیل

تیم طراحی

UML (Rational Rose) (Power Designer)

حداکثر ۲۵ تا ۳۵ درصد

تیم پیاده‌سازی

تیم تست

وراثت Inheritance

UML

Class Diagram

Person

Employee

نماد فوق یعنی: کلاس Employee از کلاس Person، Inherit شده است.

تعبیر:

Employee is a (an) Person

نکته: **تمام** Member های کلاس پدر (Person) **بلااستثناء** به کلاس فرزند (Employee) منتقل می‌شود.

نکته:‌ زمانی که از کلاس پدر Inherit می‌کنیم و کلاس فرزندی (Employee) ایجاد می‌کنیم، می‌توانیم Member‌ هایی را به کلاس فرزند اضافه کنیم و یا بعضی از رفتارهای کلاس پدر را تغییر داده و یا اصطلاحا تحریف نماییم.

نکته: دقت داشته باشید که به هیچ عنوان، نمی‌توان Access Modifier کلاس پدر را در کلاس فرزند تغییر داد!

نکته: زمانی که از کلاس پدر به کلاس فرزند می‌رسیم، اصطلاحا داریم کلاس **خاص‌تری** را ایجاد می‌کنیم و به این عمل اصطلاحا Specialization می‌گویند.

نکته: زمانی که از کلاس فرزند به کلاس پدر می‌رسیم، اصطلاحا داریم کلاس **عام‌تری** را ایجاد می‌کنیم و به این عمل اصطلاحا Generalization می‌گویند.

نمونه حالت اول:

**Person**

FirstName

LastName

Age

**Manager**

Bonus

**Employee**

Salary

نمونه حالت دوم:

**Person**

FirstName

LastName

Age

**Manager**

~~FirstName~~

~~LastName~~

~~Age~~

Bonus

**Employee**

~~FirstName~~

~~LastName~~

~~Age~~

Salary

در ریاضیات:

XY + XZ = X(Y + Z)

نکته: دقت داشته باشید که در حالت دوم (فاکتورگیری) باید مفهوم is a یا is an معنی داشته باشد!

یعنی اگر دو کلاس به نام‌های قورباغه و تخته‌نرد پیدا کردیم، و احتمالا در داخل این دو کلاس تعدادی Member‌ مشترک وجود داشت! شاید ایجاد یک کلاس General برای آنها بی‌معنا باشد!

آیا

Employee is a Person?

آیا

Manager is a Person?

از نظر C# چگونه وراثت را نمایش می‌دهیم:

class Person  
{  
}

class Employee **: Person**  
{  
}

در وراثت قانون تعدی / ترایایی / تراگذاری وجود دارد:

A < B and B < C => A < C

class A  
{  
}

class B : A  
{  
}

class C : B  
{  
}

🡺

که تلویحا یا به طور ضمنی: کلاس C از کلاس A نیز با واسطه Inherit‌ شده است!

If B is an A

And

C is a B

🡺

C is an A

نکته: بر خلاف زبان C++، در زبان‌های C# و JAVA هر کلاسی (به غیر از کلاس object یا System.**O**bject) از **یک و فقط یک** کلاس Inherit می‌شود!

در زبان C++، ما Multiple Inheritance‌ داریم!

در واقع کلاس object یا System.**O**bject در C#‌ مانند حضرت آدم در دنیای واقعی است!

اگر کلاسی تعریف کرده و نگوییم که از چه کلاسی Inherit شده است، به طور تلویحی یا ضمنی از کلاس object یا System.Object ارث‌بری می‌کند:

class Person  
{  
}

class Person : System.Object  
{  
}

class Person : **object**  
{  
}

قبول دارید که تمامی کلاس‌ها در C#، باواسطه یا بی‌واسطه از کلاس object، Inherit می‌شوند؟

در بحث مفهومی Inheritance هم قبول دارید که زمانی که کلاسی، باواسطه یا بی‌واسطه از کلاسی Inherit‌ می‌شود، مفهوم is a یا is an معنی پیدا می‌کند؟

و اما شعار:

**Everything in C# is an object!**

اصل Up Casting (Implicit Casting):

هر متغیری از جنس کلاس پدر، می‌تواند به اشیایی از جنس خودش و یا فرزندانش اشاره کند.

Person p1;

Person p2 = new Person();

Employee e1 = new Employee();

p1 = p2;

p1 = e1;

اصل Down Casting (Explicit Casting):

هر متغیری از جنس کلاس فرزند، در صورتی می‌تواند برابر متغیری از جنس کلاس پدر شود، که قبلا، متغیر از جنس کلاس پدر، به شی‌ای از همان جنس کلاس فرزند اشاره کرده باشد.

Person p1 = new Person();

Employee e1 = new Employee();

Person p2 = e1; // صحیح می‌باشد Up Casting این دستور طبق اصل

پس به همین دلیل، می‌توان اینگونه برداشت کرد که هر متغیری از جنس object یا System.Object می‌تواند به هر شی‌ای در سی‌شارپ اشاره نماید.

System.Object obj = هرچی;

Employee e = p1; // Compile Error

Employee e2 = (Employee) p2; // OK

Employee e3 = (Employee) p1; // Runtime Error