Inheritance

C++ da, birbirine çok benzeyen classları miras yoluyla ortak bir class altında toplayarak daha efektif bir şekilde yazabiliriz.
Aşağıdaki classları inceleyin.

```
class Cat
{
  private:

      int _numberOfLegs;

  public:

      Cat();
      Cat( Cat const &);
      Cat& operator=( Cat const &);
      ~Cat();

      void run(int distance);

      void scornSomeone(std::string const & target);
};
```

```
class Pony
{
  private:

      int _numberOfLegs;

  public:

      Pony();
      Pony( Pony const & );
      Pony& operator=( Cat const &);
      ~Pony

      void run(int distance);

      void doMagic(std::string const & target);
};
```

Elimde bir Cat classım var. Kedimin belli bir sayıda bacağı var (_numberOfLegs), ve koşabiliyor (run()). Kendine özgü küçümseme fonksiyonu da var (scornSomeone) çünkü Kediler küçümseyici hayvanlardır.

Bir de Pony classım var. Cat classımla bazı ortak memberları var. Midillilerin de tıpkı kediler gibi bacak sayısı vardır ve koşabilirler. Ancak, kediler gibi insanları küçümsemezler, bunun yerine sihir yaparlar. Bu yüzden kendilerine özgü doMagic metodu var.

Hem Cat classında hem de Pony classında _numberOfLegs ve run memberlarına sahip olduğumuzu biliyoruz. Eğer Inheritance kullanmazsak, _numberOfLegs attributeunu ikisi için ayrı ayrı eklememiz, ayrıca Cat::run() fonksiyonunu ve Pony::run() fonksiyonunu aynı şeyi yapıyor olmalarına rağmen ayrı ayrı tanımlamamız gerekir.

Bu durum, sadece iki classım varken çok sorun değil. Ama ya hem bacak sayısı olan, hem de koşabilen bir sürü classım olsun istiyorsam? Hepsi için tek tek _numberOfLegs yazmak ve run fonksiyonunu tekrar tekrar implement etmek çok zahmetli olacaktır.

Heritage sayesinde "Cat ve Ponyler Hayvandır Bütün hayvanların bacak sayısı vardır ve koşabilirler" diyebileceğim. Ortak özellikleri tek bir class altında toplayacağım, kendilerine özel özellikleri ise kendi class tanımlarına ekleyeceğim. Bu şekilde, aynı şeyleri tekrar tekrar kodlamaktan kurtulacağım.

Aşağıdaki Animal classını inceleyin.

```
class Animal
{
  private:

    int _numberOfLegs;

  public:

    Animal();
    Animal(Animal const &);
    Animal& operator=(Animal const &)
    ~Animal();

    void run(int distance);
};
```

Pony ve Cat'in ortak özelliklerini ayrı ayrı yazmak yerine, tek class altında topladım.

```
class Cat : public Animal
{
  public:
    Cat();
    Cat(Cat const &);
    Cat & operator=(Cat const &);
    ~Cat();

    void scornSomeone(std::string const &target);
};
```

→ Animal classından miras almak istediğimi belirtiyorum.
Neden "public" yazdığımız şu anlık önemli değil.
Bu, Cat classının her örneğinin aynı zamanda bir "Animal" class örneği olduğunu gösterir. Cat class, Animal classı içindeki attribute ve metodları miras alacak.

→ Cat classının kendine özgü metodu

```
class Pony : public Animal
{
  public:
    Pony();
    Pony(Pony const &);
    Pony & operator=(Pony const &);
    ~Pony();

    void doMagic(std::string const &target);

    void run(int distance);
};
```

→ Diyelim ki, Pony'nin diğer "Animal"lardan farklı koşmasını istiyorum.
O zaman run fonksiyonunu Pony için bu şekilde tekrar tanımlayabilirim.
Böyle yaparsam Animal::run yerine Pony::run kullanır.

Bir classın başka bir classtan miras alması demek, miras aldığı classın tüm attribute ve metodlarına sahip olması demektir.

! Başka bir classtan miras alan bir classın örneğini oluşturmak istediğimde, öncelikle miras aldığı classın bir örneğini oluşturmam gerekir.

Örneğin, bir "Cat" oluşturmak istiyorsam, önce bir "Animal" oluşturup, "Cat"i onun üstüne oluşturmam gerekir. Geriken zamanda da hem Cat, hem Animal için Destructor çağrılması gerekir.

Bunu nasıl yapacağını araştırmak sana kalmış, karışık bir konu değil.

```
class Quadruped // erişebilir: name, run() ve legs
{
  private:
    std::string name; // Sadece bir Quadruped objesi erişebilir.
  protected:
    int legs[4]; // Sadece bir Quadruped ve Quadrupedden miras
                 alan obje erişebilir.
  public:
    void run(); // Her yerden erişilebilir.
};
```

```
class Dog : public Quadruped // erişebilir: run() ve legs
{
  ;
};
```

Public ve private kavramlarını görmüştük. Heritage için kullanılan yeni bir kavram görüyoruz: protected.

Private memberlar, yalnızca ve yalnızca classa ait instancelarda erişilebilirdir.
Public memberlar ise her yerden erişilebilirdir.
Protected memberlar, hem classın kendisinde, hem de ondan miras alan (child) classlarda erişilebilirdir ancak bunların dışında bir yerden erişilemezler.

Bu tam olarak ne demek? Diyelim ki, dog classı içinde bir method tanımladık: void bark(). Bu method içinde, Quadruped classındaki protected olan memberları kullanabilirsiniz ancak private olanları kullanamazsınız.

```
void Dog::bark(void)
{
  std::cout << this->legs[2]; → Doğru kullanım

  std::cout << this->name → Error verecektir.
                            Dog metodlarından
                            Quadruped'in private
                            memberlarına erişemezsin.

};
```

Bir child class tanımlarken aşağıdaki syntaxi kullanmıştık:

class Dog: public Quadruped

↓

Burada kullandığımız keyword'ü public olmak zorunda değil.
Public, private ya da protected olabilir.
Bunların arasındaki farkı araştırmak sizin işiniz.

Araştırmanız gereken bir başka konu ise "Multiple Inheritance" (Çoklu miras). Bu zamana kadar gördüğümüz örnekler basitti. Bir class, yalnızca bir classtan miras alıyordu. Ancak bu her zaman böyle değildir. Bir class birden fazla classtan miras alabilir.

Örneğin; bir Dog classı, birbirinden miras almayan Animal (hayvan) ve Quadruped (dört bacaklı) classlarının ikisinden de miras alabilirdi. Bu durum, karşımıza birkaç sorun çıkarabilir.

Örneğin, bu classlardan birinde zaten olan bir metodu, öbür classta da tanımlarsam ne olacak? Hangi classtaki davranışı göstermesi gerektiğini nasıl belirteceğim?

Daha da tuhaf bir durum düşünelim:

Diyelim ki "Animal" classından miras alan bir "TwoLegged" classımız olsun.
Ayrıca, yine "Animal" classından miras alan bir de "TwoArmed" classımız olsun.

Eğer, hem "TwoLegged" hem de "TwoArmed" classlarından miras alan bir "Human" yaparsam, bu iki classın da "Animal" kısmı olduğu için, içeride iki tane "Animal" parçası görürüz.

Ne kadar garip değil mi? Bu duruma "Diamond Inheritance" denir. Kullanışlı olduğu pek durum yok, ama bir ara ihtiyacınız olacaktır.

Bu sayfadaki kavramları kendiniz araştırmalısınız.