Nested Classes

C++ da, bir classı başka bir class içinde tanımlayabiriz, bu classlara, dıştaki classı namespace gibi kullanarak erişebiliriz.

```
class Cat
{
    public:

    class Leg
    {
        // [...]
    };
};

class Dog
{
    public:

    class Leg
    {
        // [...]
    };
};
```

```
int main()
{
    Cat       somecat;
    Cat::Leg  somecatsleg;

}
```

Exceptions

C'de error kontrolünü; fonksiyondan -1, -2, -3 gibi değerler döndürerek; her dönüş değeri bir errora denk gelecek şekilde yapıyorduk. Hepiniz, hata çıktığında -1 döndüren o fonksiyonu yazmışsınızdır.

Bu şekilde hata kontrolü yapmanın sıkıntıları var. Bir kere, hata kodları için kullandığımız (genellikle negatif) sayıları normal dönüş değeri olarak kullanamıyorduk. Bu yöntem, yapabileceğimiz şeyleri kısıtlıyor ve gerçekten sinir bozucu.

İşte burda exceptionlar devreye giriyor. C++ da exceptionlar, error handlelamak için kullanılır. Bir dizi çağrı kullanarak errorları mesajla rapor edebilmemizi sağlar.

```
void test1()
{
    try
    {
        // Bir şeyler yap

        if( /* Error varsa */)  → error koşulunu kontrol et
        {
            throw std::exception(); → Error koşulu sağlandığında
                                      exceptionu bu şekilde atıyorum.
                                      throw, bir 'catch' bloğu
                                      bulana kadar tüm bloklara
                                      bakar.
        }
        else
        {
            // Başka bir şeyler daha yap
        }
    }
    catch(std::exception e)  → catch bloğu, std::exceptionu
    {                          yakalar (exceptiona e ismini
                               verdim.)

        // Burda error handling

    }
```

↓

Freeleme, error mesajları vs...

```
void test2()
{
    // Bir şeyler yap

    if(/* Error varsa */)
    {
        throw std::exception();
    }
    else
    {
        // Başka bir şeyler
    }
}
```

Gördüğünüz gibi, test2 fonksiyonunda try/catch bloğu yok.
Sadece bir şeyler yapıyor, ve error koşulu sağlandıysa exception atıyor.
"Eee, catch yoksa exceptionu nasıl yakalayacağız?" diye sorabilirsiniz.
Aşağıdaki fonksiyona bakın.

```
void test3()
{
    try
    {
        test2();
    }
    catch(std::exception& e) → exceptionu referansla yakalayabilirim.
    {
        // Error handling
    }
}
```

Burada test2 fonksiyonunu, test3 içindeki try/catch bloğu içinde çağırdım. Böylece, test2 fonksiyonu bir exception attığında, test3 deki catch bloğu bu exceptionu yakalayabilir.

```
void test4()
{

    class PEBKACException : public std::exception
    {

      public:

        virtual const char* what() const throw()
        {
            return("Problem exists between keyboard and chair")
        }
    };
```

// Yukarıda, PEBKACException'u tanımladık. std::exceptiondan public
// olarak miras alıyor. Bu da demektir ki, PEBKACException bir
// exceptiondır ve onu bu şekilde manipüle edebilirim.

// PEBKACException içinde tanımladığımız what() metoduna bakalım.

```
virtual const char *what const throw()
```

Fonksiyon tanımının sonuna koyulan throw() ifadesi, o fonksiyonun hangi exceptionları throwlayabileğini belirtmek için kullanılır. Bu şekilde iç: boşken, fonksiyon hiçbir exception atamaz. throw() içine herhangi bir exception yazabilirdim. Örneğin,

```
virtual const char *what const throw(PonyIsNotHereException)
```

↓

what() metodu, belirtilen exceptionu throwlayabilir.

Fonksiyon tanımlarken throw() kullanımı şiddetle tavsiye edilir. Exceptionlar ile başa çıkmanın son derece temiz bir yoludur.

```
try
{
    test5();
}
catch (PEBKACException& e)          → test5 PEBKACException atarsa çalışacak
{
    // Kullanıcının aptallığını handlela
}
catch (std::exception& e)           → test5 PEBKACException dışında bir std::exception atarsa çalışacak
{
    // farklı exceptionları handlela
}
```

---

std::exception olmayan exceptionları yakalamanın da bir yolu var. String, integer gibi şeyleri de throwlayabiliyoruz.
Peki bunu neden anlatıyorum?
Berbat ve yapmamanız gereken bir şey olduğu için tabii ki.
Gerçekten kullanmak isterseniz araştırın, ancak kısa süre içinde neden kötü bir fikir olduğunu fark edeceksiniz.

Exceptionlar hakkında bir şey daha: adı üstünde "exception" (türkçesi istisna) yani exceptionları istisnai durumlarda kullanmalısınız. Kıyamet kopmadığı sürece kullanmayın demiyorum. Demek istediğim şu:

Fonksiyonunuzun 2 çağırmadan 1'inde hata döndürmesini bekliyorsanız bir exception döndürmesini istemezsiniz. Exception içeren bu fonksiyonu sürekli çağırdığınız taktirde, her yerde resource harcayıp durur. Exceptionlar, basit bir değer döndürmekten daha fazla kaynak harcar.

Yani:

Sürekli gerçekleşip duracak bir error için: Error belirten return değeri kullanmak (klasik yöntem)

Nadiren, programın normal davranışında olmayan istisnai error için: exception kullanmak

en iyisidir.

Sistem fonksiyonları genellikle exception fırlatır (Örneğin new(), hafızada yer yoksa std::bad_alloc) bunun birçok örneği var, dilerseniz araştırın.