

- \* Tekrar:
- Hatalı kodu önlemek için → dynamic / static assertion
  - Run time'da hatalarla baş etmek için → exception handling

- Throw statement ile istenilen hata nesnesini oluşturmak için kod, daha yukarıdaki kodları haberle etmek için bir hata nesnesi oluşturur ve gelenek gider.

```
throw expr;
```

→ expr L-value bile olsa, compiler'in kendisinden önce gider.

- Gündelik nesneler, genetik nesne genelde. Sınıf hiyerarşisi var ve virtual function'la sınırlı

- Try bloğu, catch'in throw ettiği hataları, bu blokta handle ederler.

- Ceten parametreleri → references olmalı:
  - references olmazsa copy ctor çağırılır, ve copy ctor'la exception giderilir
  - Object slicing'e neden olur. Run time polymorfizminin yaratsızdır.

```
throw expr;

try {
    yok olan exception
    nesnesini degritirmeyebilen const
}
catch (const std::exception &){
```

↓  
Sadece const type çağrılabilir,

→ exception nesnesini hala kullanılamazken, döndürür references yes. form ve ne.

- Exception yukarıdakilerin yanı sıra → terminale:
  - exception translate: istenilen exception'ın yerine exception'ın代替
  - exception rethrow: kendisinden exception'ın代替

## \* Exception Lethrow:

```
void func()
{
    try {
        // throw std::out_of_range{ "range hatası" };
    }
    catch (const std::exception& ex) {
        std::cout << "hata func içinde yakalandı : " << ex.what() << "\n";
        // iki farklı throw statement arasındaki farkı görünüz.
        throw ex; → 2. caten'sideki çağrındı. → Original exception nesnesi yerine exception tekrarı
        //throw; → 1. catch'teki çağrındı. → out of range exception'ı tekrar kaçırdı
    }
}

int main()
{
    try {
        func();
    }
    1. catch (const std::out_of_range&){
        std::cout << "hata yakalandı (std::out_of_range)\n";
    }

    2. catch (const std::exception&){
        std::cout << "hata yakalandı (std::exception)\n";
    }
}
```

→ Dokka 35' ornegine bak / Dikkatim dogildi.

→ Stack unwinding: otomatik türkic nesneler, exception yipwendiğinde, programın akisi her bir start frame'dan gitgitinde, destruktorlarının çağrılması garantiye sunur.

```
void foo();  
  
void func()  
{  
    FILE* f = std::fopen("a.txt", "w");  
    //...  
  
    foo(); //if throws;  
    std::fclose(f);  
}
```

programın okisi berasın çıkar,  
ve before yapılır

→ unique pointer, (smart pointer) → stack unwinding'e yararlı, dinamik türkic nesnelerin, exception state'ı bir selfde destroy etmeli olduğu.

### \* Exception Safety Guarantees:

#### - No Guarantee:

```
void func()  
{  
    FILE *f = fopen....  
  
    foo(); → exception gönderirse, file  
    | → Celleşmez. Zaten.  
    fclose(f);  
}
```

#### - Basic Guarantee / No-leak:

- resource leak olmayacağı
- hayatı devam eden nesneler → valid state'ı kalacak.

##### basic guarantee (no-leak)

hiç bir kaynak sizdirmeyecek  
hayatı devam eden nesneler "valid state" (geçerli durum)  
nesne geçerli bir değerde kalacak  
nesnenin state'i değişmiş olabilir → program state'st'e devam etmeye devam etmez  
nesnenin destructor çağrılabılır.

#### - Strong guarantee:

- commit or rollback. → Filtre mevcuttur.  
(Düzenleme, Inceleme gibi topyolar)

- invariant: bir sınıf nesnesinin, her zaman aldığı durum genken stateleri  
(link list son elemen null olmalı, ...)

#### - No Throw Guarantee:

- Destructörler, swap fonksiyonları, move memberler, koput roda eden fonksiyonlar (delete) no throw garantisi verir.

```
void func() throw(std::bad_alloc); → Dynamic exception specification
```

```
void func() throw(); → Exception throw etmeye garanti
```

- modern C++'da "noexcept" geldi → hem specifier  
→ hem operator olmas.

void func() noexcept;  
void foo();

bu exception gonderir.

noexcept specifier

void func() noexcept(false);  
void func();

bu ilk ayin onlemo gelir.

true'dan olustur.

```
void func(int) noexcept;
```

```
int main()
{
    try {
        func(12);
    }
    catch (const std::exception& ex) {
        std::cout << "exception caught: " << ex.what() << '\n';
    }
}

void func(int x) noexcept
{
    if (x % 2 == 0)
        throw std::runtime_error{ "hata" };
}
```

Local bugs  
Solved:

Footer noexcept  
foreign exception  
gonderisse  
handle edilmesi  
terminate edilmesi.

• Destruktor nedensel noexceptdir: - Destruktor 2 nedenle çağırılır.

Scope'lü ömrü biten nesneler

return

Exception yakalaması ve bu yorden

stack unwinding sonucu yaratabilece ve otomatik  
ömrü biten nesnelerin destruktörleri çağırılır. Còngte öncelikle  
otomatik ömrü biten nesneler destruktur edilir, sonra catch  
blokuna bilgi. → bunlardan biri exception throw  
edirse, terminate edilir program, catch'e gitmesi gerekece!!

```
class MyClass {
public:
};

int main()
{
    using namespace std;
    const auto H = is_nothrow_default_constructible_v<MyClass>;
```

compile time da  
default edilen constructor'ın  
noexcept mi değil mi olduğunu söyley.

(Local variable) const bool H = true

Search Online

true

→ C++ move semantics kitabı oku. Ondan özet verdii. Vay aq dedim!

Move konstuktor noexcept olmasa, copy assignment with borate

• Operatör olan noexcept:

```
class Nec {  
public:  
};  
  
int main()  
{  
    constexpr auto b = noexcept(Nec{});  
}
```

Default ctor, compiler tarafından noexcept  
operatör olarak brr ifade olur.  
→ const bool return değerli, compile time  
da böyle "eger verilen ifade exception göndermesse"  
"true", "eger verilen ifade exception gönderme  
istemci versa "false" return eder.

```
class Myclass {};  
  
void func(const Myclass& r) noexcept(noexcept(Myclass{}));
```

b operator  
b specifier : true / false  
eger myclass default  
ctor'un noexcept garantisiyle,  
func'da noexcept  
func'in noexceptlik  
durumu, myclass'a bağlı

```
int main()  
{  
    int (*fp)(int)noexcept; ] → noexcept garanti  
    r̄sen function  
    fp(12)|  
}
```

```
class Base {  
public:  
    virtual void func()noexcept;  
};  
  
class Der : public Base {  
public:  
    void func()override; ] → t̄c̄nem̄ simdi  
    Overriden'da noexcept  
    olmali, yoksa system  
    hatasi.
```

→ Fakat tam tersi değil: Base class func noexcept  
olmadan, derived class func noexcept tömürde  
jaceltilir.

→ Exception'a enrypten fonksiyon → constructor  
Eğer konstruktörde exception  
genetilirse, destekler GABIZULMAZ  
= RESOURCE LEAK