Kural Dışı Durum Yönetimi (Exception Handling)

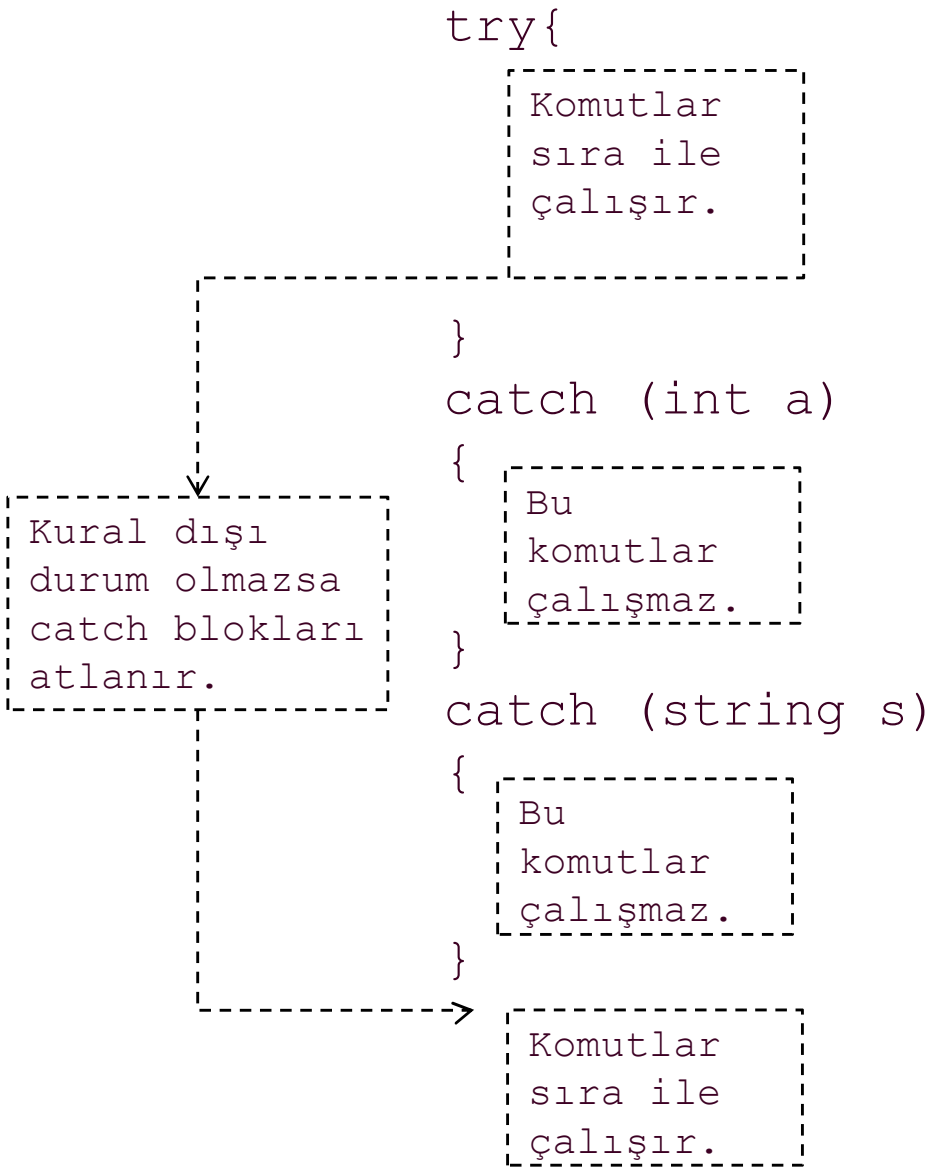
Dr. Metin Özkan

Giriş

- Kaliteli bir yazılım, her koşulda çalışmalıdır.
- Bu amaçla, karşılaşılabilecek her beklenmedik durumda, program hata vermeden çalışacak şekilde tasarlanmalıdır.
- Programın çalışması sırasında oluşan hataların veya kuraldışı durumların yakalanması, önlem alınması, raporlanması ve çalışmaya devam etmesi hedeflenir.
- Böylece daha güvenli bir yazılım geliştirilmiş olur.
- Kural dışı durumları (exception) yönetmek için üç farklı komut kullanılmaktadır:
  - `try` *(deneme)*
  - `throw` *(fırlatma)*
  - `catch` *(yakalama)*

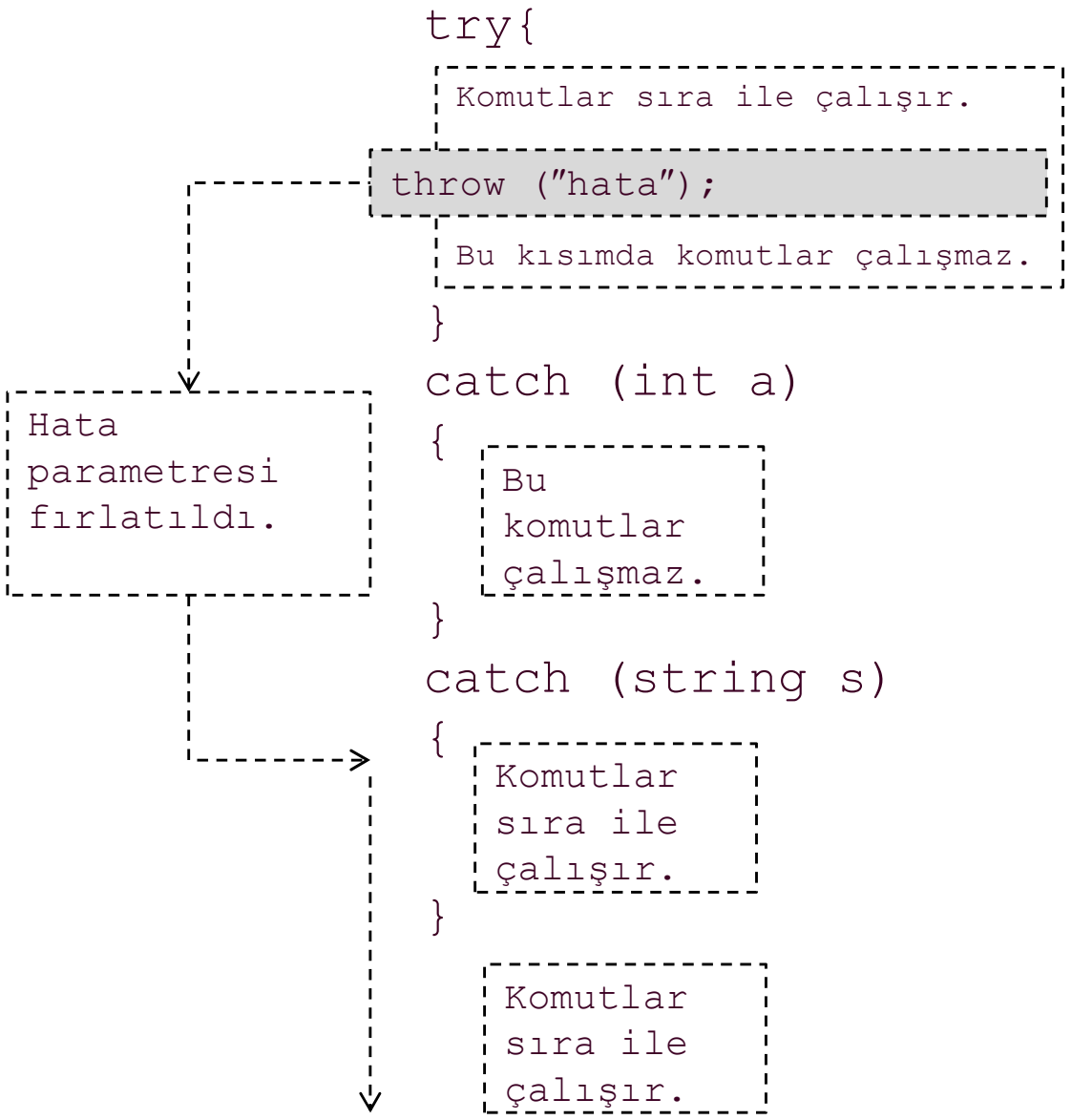
Kuraldışı Durum Yönetimi (Exception Handling)

- try (deneme) komutu: Bu komutun bloğu içerisinde yer alan komutların çalıştırılması denenir.
- throw (fırlatma) komutu: Kural dışı durum olduğunda kullanılır. Bu komut ile belirtilen parametre ile hata fırlatılır.
- catch (yakalama) komutu: throw komutu ile gönderilen parametre yakalanır ve bu blok içerisinde parametre ile ilişkilendirilmiş kodlar işletilir.



Kuraldışı Durum Yönetimi (Exception Handling)

- Eğer, kural dışı durum olmaz ise, program normal şekilde akışına devam eder. Catch blokları içindeki kodlar işletilmez.
- Eğer kural dışı durum olursa, fırlatılan parametre, ilgili catch bloğu tarafından yakalanır ve blok içerisindeki kodlar işletilir. Bu bloktan sonraki kodlar işletilmeye devam eder.



Kuraldışı Durum Yönetimi (Exception Handling)

- Bir örnek

```
#include <iostream>
#include <string>
using namespace std;

double divide(int a, int b){
          if(b==0) throw "divide by zero";
          return ((double)a/b);
}

int main(){
          int x, y;
          double result;
          try{
                    result=divide(x, y);
          }
          catch (string s){
                    cout<<s<<endl;
          }
          return 0;
}
```

Kuraldışı Durum Yönetimi (Exception Handling)

- Kuraldışı durum için bir sınıf tanımlama

```
#include <iostream>
#include <string>
using namespace std;

class Error{
public:
     string getMsg(){return "divide by zero";}
};
double divide(int a, int b){
     if(b<0)
          throw (string)"Denominator is negative";
     if(b==0)
          {Error err; throw err;}
     return ((double)(a/b));
}
int main(){
     int x, y;
     double result;
     try{
          result=divide(x, y);
     }
     catch (string &s){
          cout<<s<<endl;
     }
     catch (Error &err){
          cout<<err.getMsg()<<endl;
     }
     return 0;
}
```

Kuraldışı Durum Yönetimi (Exception Handling)

- Kural dışı durum yönetimi, özellikle yapıcı fonksiyondaki (constructor) hataları bulmak için kullanışlıdır. Çünkü yapıcı fonksiyon hata kontrolü için değer döndürmez.

```
#include <iostream>
#include <string>
using namespace std;

enum {MAXSIZE=10};
class Array{
    int size;
    int *arr;
public:
    Array(int);
    void print();
    ~Array(){delete arr;}
};
Array::Array(int size)
{
     if(size>MAXSIZE)
          throw (string)"size too large";
     arr=new int[size];
     for(int i=0;i<size;i++)
          arr[i]=i;
}
int main(){
     int sz;
     cout<<"Enter the array size:";
     cin >>sz;
     try{
          Array arr1(sz);
     }
     catch (string &s){cout<<"Exception:"<<s<<endl;}
}
```

Kaynaklar

- T.C. Lethbridge and R. Laganiere, Object-Oriented Software Engineering - Practical software development using UML and Java, McGraw Hill, Second Edition, 2005.
- H.M.Deitel and P.J.Deitel, C++ How To Program, 9E, Pearson Press, 2014.
- B. Stroustrup, The C++ Programming Language, 3rd Edition, Special Edition, Addison Wesley, 2000.
- Dr. Feza Buzluca, Ders Notları.
- Ç. Turhan ve F.C. Serçe, C++ Dersi: Nesne Tabanlı Programlama, 2nci Baskı, 2014.