Fehlerbehandlung

„Defensive Programmierung“ considered harmful

Stefan Lieser, CCD Akademie GmbH, 11.03.2022

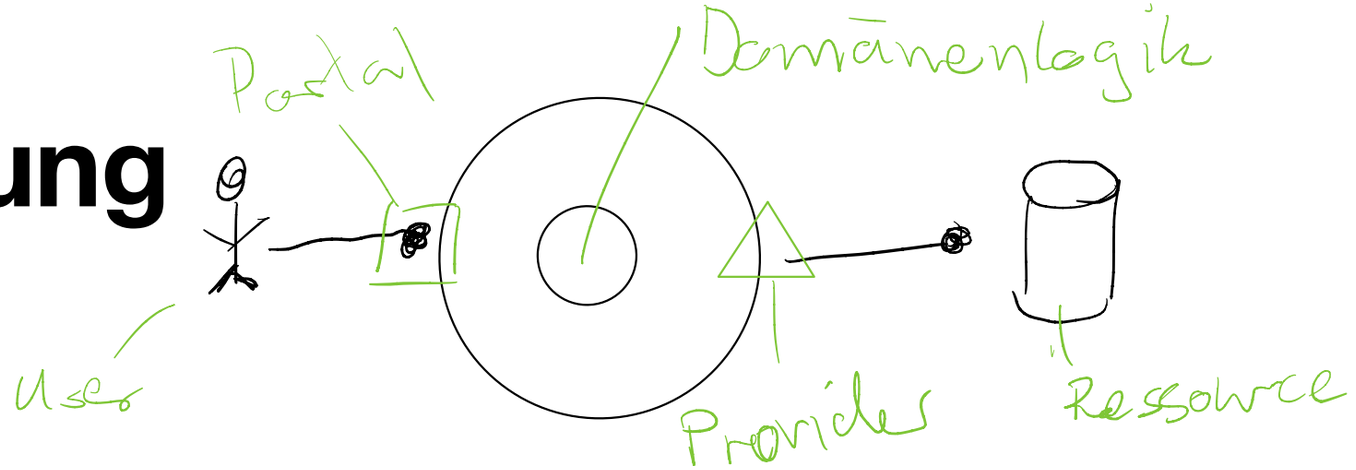
Kategorien von Fehlern

- **Bedienerfehler**
  - Bspw. fehlende oder falsche Eingaben
- **Fehler in der Umgebung**
  - Bspw. Datei nicht gefunden oder DB antwortet nicht
- **Entwicklerfehler**
  - Bspw. Zugriff auf Null oder Index out of Bounds

Umgang mit den Fehlern

- **Bedienerfehler**
  - Erwartbar, ergibt sich aus den Anforderungen -> Entwurf einer Lösung
- **Fehler in der Umgebung**
  - Erwartbar, ergibt sich aus der Systemumgebung -> Entwurf einer Lösung
- **Entwicklerfehler**
  - Nicht erwartet -> Globaler Exception Handler

Parameterprüfung



- **Vertrauensgrenze**
  - Parameter sollen möglichst nur an der Vertrauensgrenze auf Fehler geprüft werden.
- Öffentliche Schnittstelle
  - Bspw. *public* Methode als Bestandteil einer API vs. von dort aufgerufene *private* Methoden
- Sicherheitsgrenze
  - Bspw. öffentlich erreichbare API des Backends

Bedienfehler

- Bspw. ein nicht angegebener Parameter auf der Kommandozeile
- Konsequenz: die verantwortliche Funktionseinheit muss zwei mögliche Resultate liefern:
  - Wenn alles richtig bedient wurde -> die Benutzereingabe
  - Andernfalls -> Hinweis auf die Fehlbedienung

Fehler in der Umgebung

- Bspw. eine nicht lesbare Datei
- Konsequenz: die verantwortliche Funktionseinheit muss zwei mögliche Resultate liefern:
  - Wenn die Datei gelesen werden kann -> den Dateiinhalt
  - Andernfalls -> Hinweis auf die Fehlbedienung

Entwicklerfehler

- Bspw. Zugriff auf Null
- Konsequenz: die verantwortliche Funktionseinheit kann nicht weiter ausgeführt werden
  - Managed Runtime: Implizit wird eine Exception ausgelöst
    - Null Zugriff, Index out of Bounds, Division by Zero, o.ä.
  - Unmanaged Runtime: Explizite Prüfung einer Vorbedingung und Auslösen einer Exception
    - Bspw. Gültigkeitsprüfung eines Index vor einem Arrayzugriff
  - Standardlösung: Globaler Exception Handler fängt alle Entwicklerfehler

Implementation - Besser NICHT so!

```
public string GetFilename(string[] args) {
    if (args.Length > 0) {
        return args[0];
    }
    return "";      // !!! FALSCH !!!
}
```

```
var filename = GetFilename(args);
// Woran erkenne ich, dass zwei Fälle
// zu berücksichtigen sind?
```

Implementation - Callback bzw. Continuation

```
public void GetFilename(string[] args, Action<string> onFilename, Action onNoFilename) {
    if (args.Length > 0) {
        onFilename(args[0]);
    }
    else {
        onNoFilename();
    }
}
```

```
GetFilename(args,
    onFilename: filename => {
        Console.WriteLine(filename);
    },
    onNoFilename: () => {
        // ...
    });
```

Implementation - Option<T>

```
public Option<string> GetFilename(string[] args) {
    if (args.Length > 0) {
        return Option.Some(args[0]);
    }
    return Option.None<string>();
}
```

```
var filename = GetFilename(args);
if (filename.HasValue) {
    Console.WriteLine(filename.ValueOrFailure());
}
else {
    // ...
}
```

Implementation - Try…

```
public bool TryGetFilename(string[] args, out string filename) {
    if (args.Length > 0) {
        filename = args[0];
        return true;
    }
    filename = "";
    return false;
}
```

```
if (TryGetFilename(args, out var filename)) {
    Console.WriteLine(filename);
}
else {
    // ...
}
```

Implementation - Tuple Rückgabewert

```
public (bool hasResult, string filename) GetFilename(string[] args) {
    if (args.Length > 0) {
        var filename = args[0];
        return (true, filename);
    }
    return (false, "");
}
```

```
var (hasResult, filename) = GetFilename(args);
if (hasResult) {
    Console.WriteLine(filename);
}
else {
    // ...
}
```

Fehler in der Umgebung - Erkennung

```
public bool TryReadFile(string filename, out string[] fileContent) {
    try {
        fileContent = File.ReadAllLines(filename);
        return true;
    }
    catch {
        fileContent = new string[0];
        return false;
    }
}
```