Ich liebe Java && Ich liebe C#

Rolf Borst

```
public class Einfuehrung {

    private int gesamtzahl = 0;

    /* Ermittelt die Anzahl der geraden und durch drei teilbaren Zahlen */
    public String ErmittelnErgebnis(String praefix, int[] zahlen) {

        int anzahl = 0;

        // Schleife
        for (int i = 0; i < zahlen.Length; i++) {
            if (zahlen[i] % 2 == 0 && zahlen[i] % 3 == 0) {
                anzahl++;
            }
        }

        gesamtzahl += anzahl;

        return praefix + ": " + anzahl;

    }
}
```

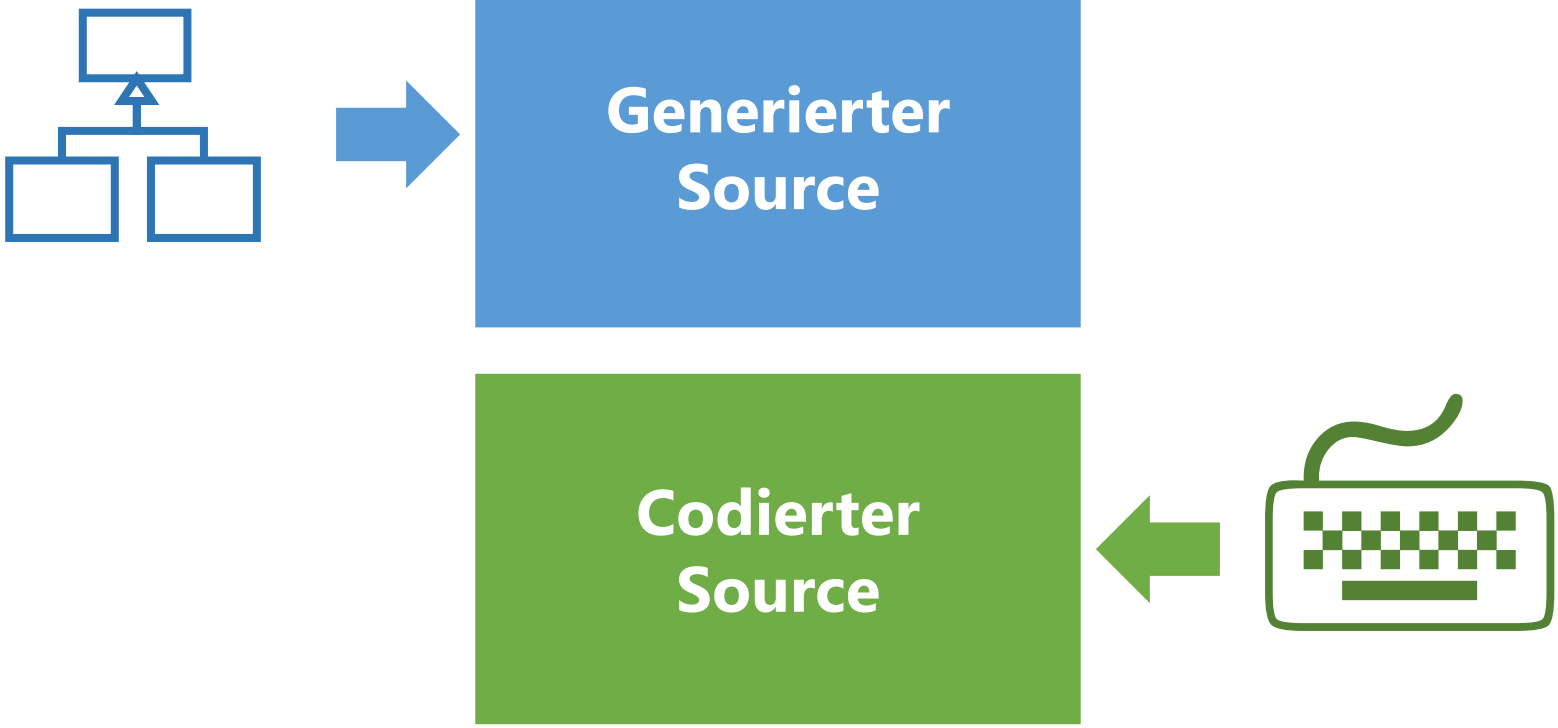
Agenda

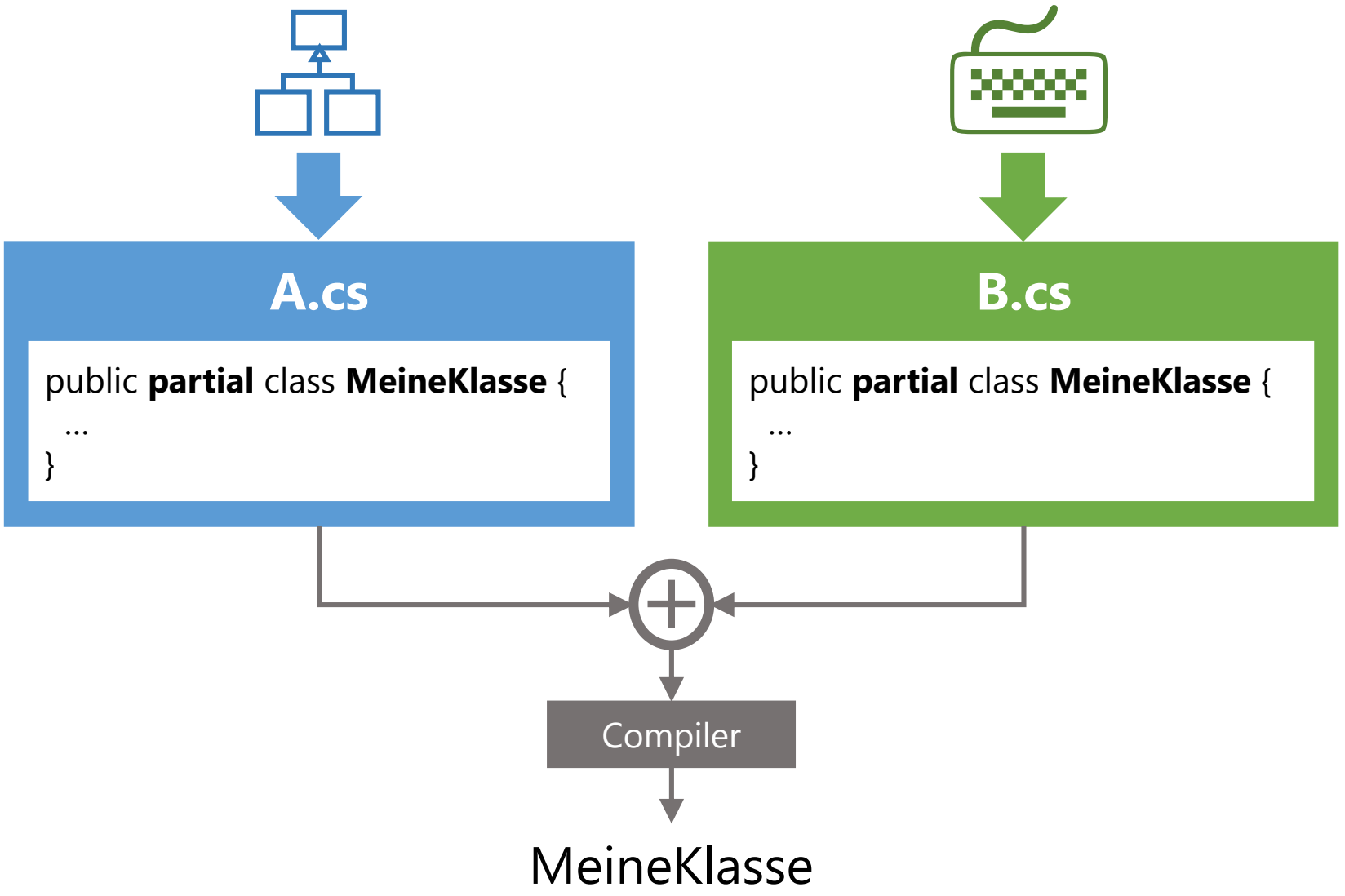
1. Partial Classes
2. Extensions Methods
3. LINQ
4. Lambda Expressions
5. Asynchronous Functions

Um was es **nicht** geht:
Was ist die bessere Sprache?

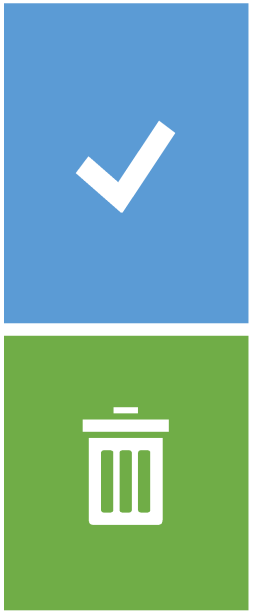
Ich liebe **beide!**

1 Partial Classes

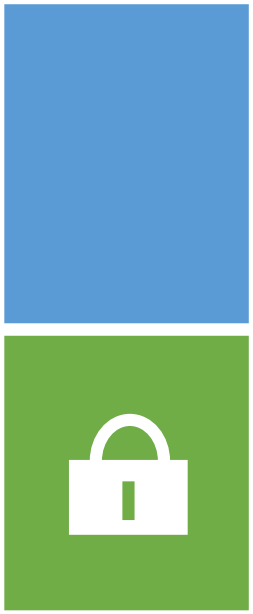

Generierter
Source
Codierter
Source


A.cs
public partial class MeineKlasse {
...
}
B.cs
public partial class MeineKlasse {
...
}
Compiler
MeineKlasse

Aufteilung großer Klassen



Auslagerung von Deprecated-Bestandteilen



Aufteilung Änderungs-rechte

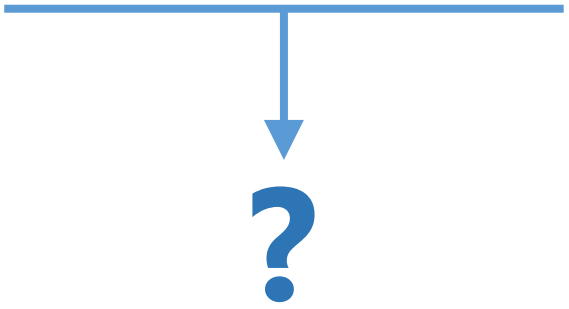
Partial Classes

★★★★

Mein Fazit: **Ganz interessant**

2 Extension Methods

```
String text = "Streng geheimer Text";
String geheim = text.verschluesseln("Passwort");
```



```
public class SecurityUtils {

    public static String verschluesseln(String text, String passwort) {
        // Verschlüsselung mit Framework
        return ...;
    }

}
```

```
String geheim = SecurityUtils.verschluesseln(text, "Passwort");
```

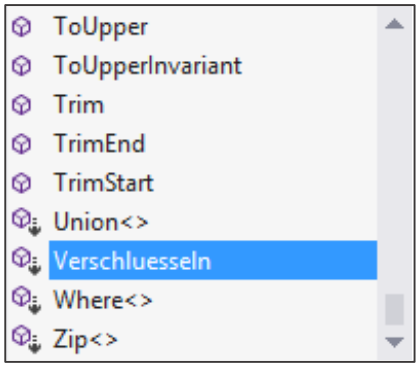
```
public static class SecurityUtils {

    public static String Verschluesseln(this String text, String passwort) {
        // Verschlüsselung mit Framework
        return ...;
    }

}
```

```
String geheim = SecurityUtils.Verschluesseln(text, "Passwort");
```

```
public static class SecurityUtils {

    public static String Verschluesseln(this String text, String passwort) {
        // Verschlüsselung mit Framework
        return "...";
    }

}
```

```
String geheim = SecurityUtils.Verschluesseln(text, "Passwort");
```

```
String text = "Streng geheimer Text";
String geheim = text.Verschluesseln("Passwort");
```



Intern: SecurityUtils.Verschluesseln(text, "Passwort");

```
public static double Arctan(this double zahl) { ... }
double c = (1.0).Arctan();
```

```
public static String ToXml(this object objekt) { ... }
```

```
public static T[] Verschieben<T>(this T[] array) { ... }
```

```
if (name != null && name.length() > 0) {
    ...
}

if (name != null && !name.isEmpty()) {
    ...
}

if (!StringUtils.isEmpty(name)) {
    ...
}
```

```
if (name.IsSet()) {
    ...
}
```

```
String name = null;

if (name.IsSet()) {
    ...
}
```

```
public static bool IsSet(this String text) {
    if (text == null) {
        return false;
    } else {
        ...
    }
}
```

```
public static bool IsSet(this String text) { … }
public static bool IsSet(this object objekt) { … }
public static bool IsSet<T>(this T[] array) { … }
public static bool IsSet(this double zahl) { … }
```

IsEmpty

IsEmpty

IsEmpty

IsEmpty

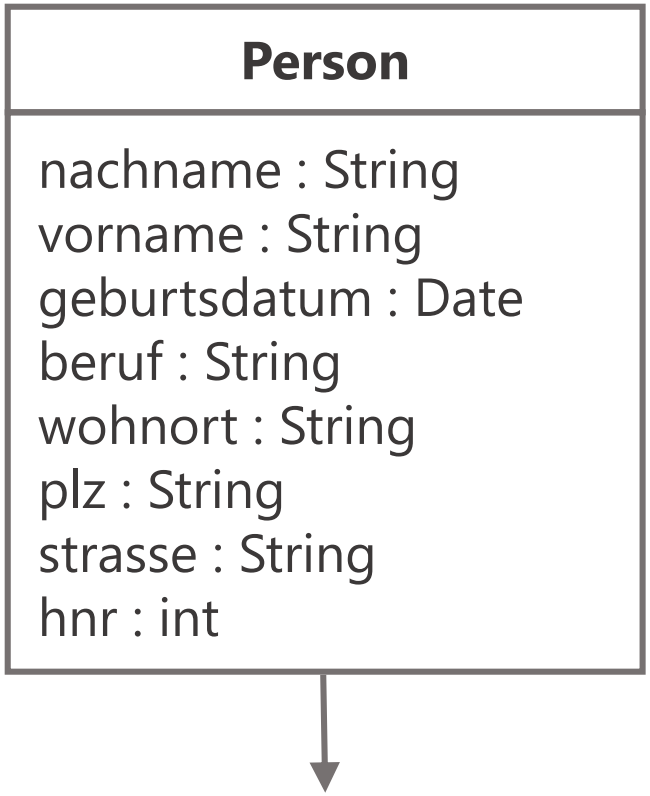
```
if (any.IsSet()) {
    …
}
```

Extension Methods

Mein Fazit: **Genial**

3 LINQ

Language Integrated Query


Person
nachname : String
vorname : String
geburtsdatum : Date
beruf : String
wohnort : String
plz : String
strasse : String
hnr : int
List<Person> personen

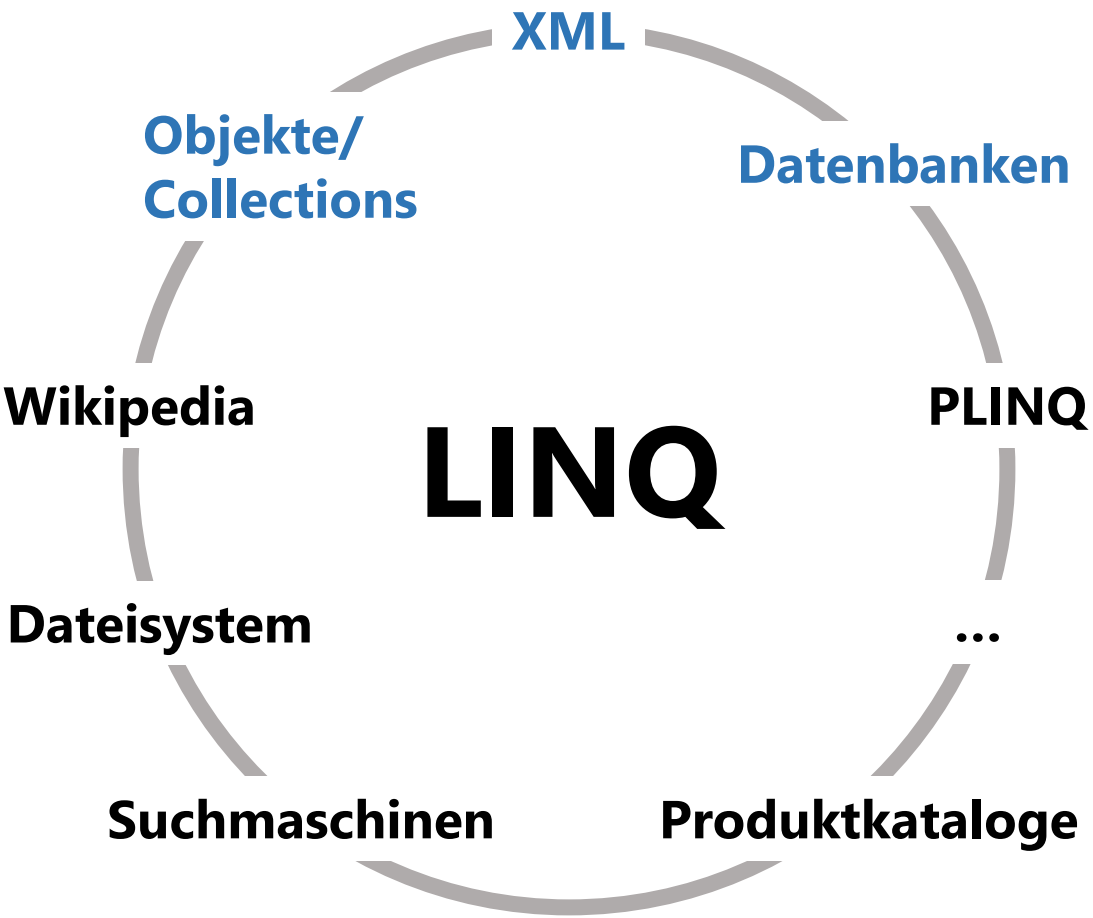
Aufgabe

Erstellung einer Liste mit dem Nachnamen und dem Vornamen aller **Personen**, die an einem **bestimmten Ort** wohnen und einen **bestimmten Beruf** ausüben. **Sortiert** werden soll die Liste zuerst nach dem Nachnamen und dann nach dem Vornamen.

```
List<Person> personen = new ArrayList<Person>();

String wohnort = "Stuttgart";
String beruf = "Entwickler";

List<Suchergebnis> liste = new ArrayList<Suchergebnis>();
for (Person p : personen) {
   if (wohnort.equals(p.getWohnort()) && beruf.equals(p.getBeruf())) {
      liste.add(new Suchergebnis(p.getNachname(), p.getVorname()));
   }
}

Collections.sort(liste, new Comparator<Suchergebnis>() {
   public int compare(Suchergebnis s1, Suchergebnis s2) {
      if (s1.getNachname() != null && s2.getNachname() != null) {
         int compare = s1.getNachname().compareTo(s2.getNachname());
         if (compare == 0) {
            if (s1.getVorname() != null && s2.getVorname() != null) {
               return s1.getVorname().compareTo(s2.getVorname());
            }
         } else {
            return compare;
         }
      }
      return 0;
   }
});
```

```
SELECT nachname, vorname
FROM person
WHERE wohnort = 'Stuttgart'
AND beruf = 'Entwickler'
ORDER BY nachname, vorname
```

```
List<Person> personen = ... ;

String wohnort = "Stuttgart";
String beruf = "Entwickler";

var liste = from p in personen
            where p.Wohnort == wohnort && p.Beruf == beruf
            orderby p.Nachname, p.Vorname
            select new { p.Nachname, p.Vorname };
```


XML
Objekte/
Collections
Datenbanken
Wikipedia
LINQ
PLINQ
Dateisystem
...
Suchmaschinen
Produktkataloge

LINQ ★★★★★

Mein Fazit: **Super**

4 Lambda Expressions

Bekannt in Java:

public class Person { … }

public interface Sortierbar { … }

public enum Geschlecht { … }

Unbekannt in Java:

public delegate bool **Personenfilter**(Person person);

```
public delegate bool Personenfilter(Person person);
```

```
public bool IstWeiblich(Person person) { ... }

public bool IstMaennlich(Person person) { ... }

public bool IstVolljaehrig(Person person) { ... }
```

```
Personenfilter filter = IstWeiblich;
...

if (filter(person)) {
    ...
}
```

```
public void Verarbeiten(List<Person> personen, Personenfilter filter) {
    foreach (Person person in personen) {
        if (filter(person)) {
        }
    }
}
```

```
List<Person> personen = ... ;

Verarbeiten(personen, IstWeiblich);

Verarbeiten(personen, IstMaennlich);

Verarbeiten(personen, IstVolljaehrig);
```

public delegate **bool** **Personenfilter**(**Person** person);

Personenfilter filter = **p => p.Alter > 30 && p.Alter < 70;**

Verarbeiten(personen, **filter**);

Verarbeiten(personen, **p => p.Alter > 30 && p.Alter < 70**);

```
int minalter = 30;
int maxalter = 70;

Verarbeiten(personen,
                    p => p.Alter >= minalter
                         && p.Alter <= maxalter);
```

Lambda Expressions

Mein Fazit: **Mächtig**

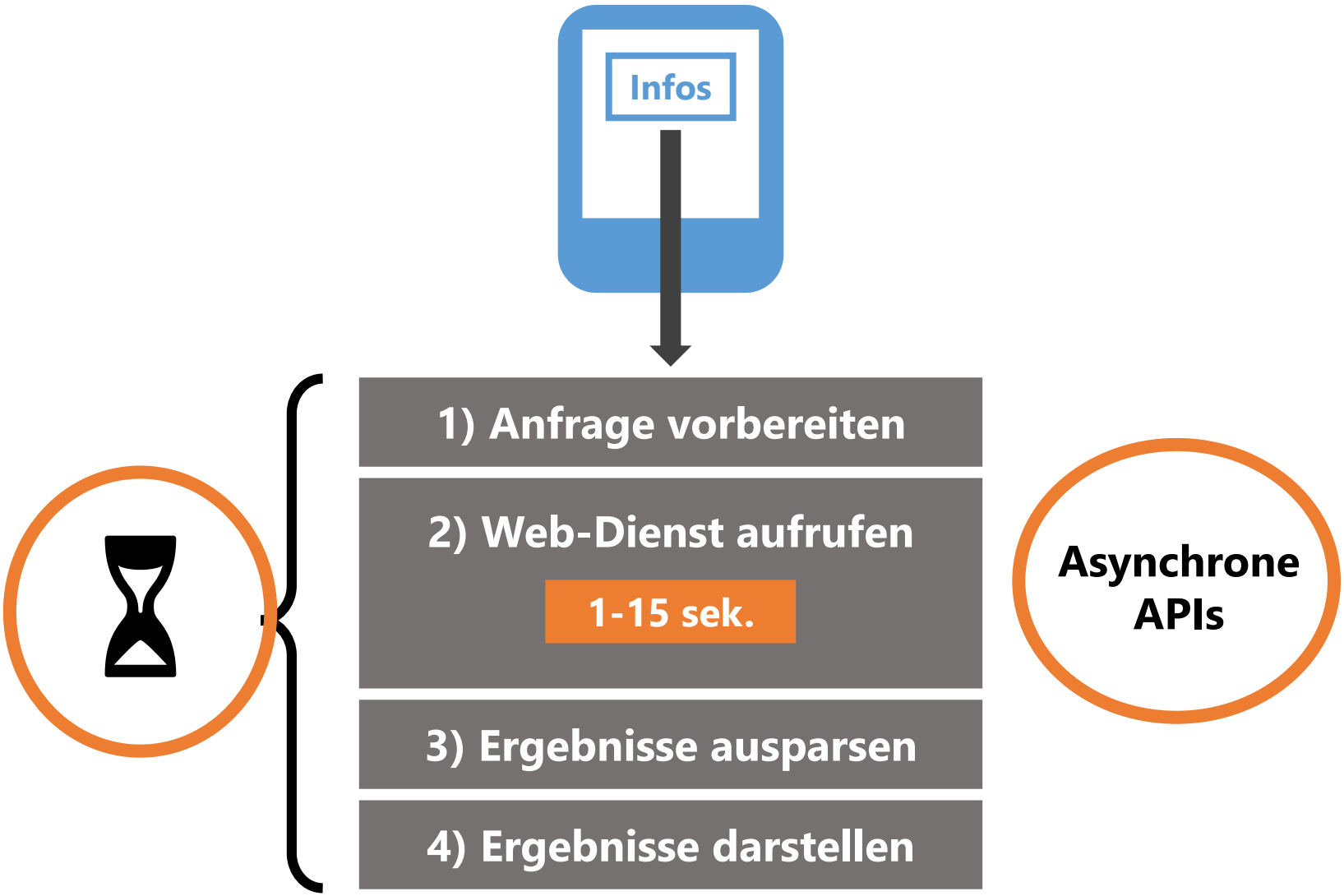
5 Asynchronous Functions

1) Anfrage vorbereiten

2) Web-Dienst aufrufen

3) Ergebnisse ausparsen

4) Ergebnisse darstellen


Infos
1) Anfrage vorbereiten
2) Web-Dienst aufrufen
1-15 sek.
3) Ergebnisse ausparsen
4) Ergebnisse darstellen
Asynchrone APIs

```
private async void OnClick() {
1   Anfrage anfrage = Vorbereiten();
2   DienstResult result = await AufrufenDienstAsync(anfrage);
3   Ergebnis ergebnis = Ausparsen(result);
4   Darstellen(ergebnis);
}
```

Asynchronous Functions

★★★★

Mein Fazit: **Richtiges Thema, Richtige Zeit**

Ich liebe Java && Ich liebe C#

Danke