Protokoll Softwaredesign Aufgabe Debugging 11.04.2018

```
23 references
public class Person
{
    15 references
    public string FirstName;
    16 references
    public string LastName;
    15 references
    public DateTime DateOfBirth;

    8 references
    public Person Mom;
    8 references
    public Person Dad;
}
```

Man sieht anhand dieses Bildes aus dem Code die rekursive Datenstruktur der Klasse Person, da innerhalb jedes Objektes von Typ Person jeweils zwei Objekte vom Typ Person, hier „Mom“ und „Dad“ genannt, erstellt werden.

```
Locals
  args [string[]]: {string[0]}
  root: {Debugging.Person}
    Dad: {Debugging.Person}
      Dad: {Debugging.Person}
      DateOfBirth [DateTime]: {14.11.1948 00:00:00}
      FirstName [string]: "Charlie"
      LastName [string]: "Wales"
      Mom: {Debugging.Person}
    DateOfBirth [DateTime]: {21.07.1982 00:00:00}
    FirstName [string]: "Willi"
    LastName [string]: "Cambridge"
    Mom: {Debugging.Person}
  found [Person]: null
```

Wenn man sich den Inhalt von root im Debugger ansieht, nachdem man einen Breakpoint in Zeile 19 von Program.cs, sieht man, dass root die Person Willi ist. Des Weiteren sieht man die ganzen Eigenschaften von Willi, also sein Geburtsdatum, seinen Namen, sowie seine Eltern und deren Vorfahren.

```
21              Person ret = null;
22              if (person.LastName != "Battenberg")
23                  return person;
```

Die erste Person, die die Bedingung in Zeile 22 von FamilyTree.cs erfüllt, ist

```
Locals
  args [string[]]: {string[0]}
  root: {Debugging.Person}
  found: {Debugging.Person}
```

wieder unser Willi, da root von oben nach unten durchgegangen wird und Willi Cambridge der erste ist, dessen Nachname ungleich Battenberg ist, womit er die Bedingung erfüllt.

```
Person ret = null;
if (person.LastName == "Battenberg")
    return person;
```

Ich habe Bedingung von ungleich auf gleich geändert, daraufhin warf das Programm mir folgende NullReferenceException:

```
Unhandled Exception: System.NullReferenceException: Object reference not set to an instance of an object.
   at Debugging.Familytree.Find(Person person) in C:\Users\Philipp\Documents\Documents\Studium\4. Semester\SWDes\Aufgabe 2-Debugging\FamilyTree.cs:line 25
   at Debugging.Familytree.Find(Person person) in C:\Users\Philipp\Documents\Documents\Studium\4. Semester\SWDes\Aufgabe 2-Debugging\FamilyTree.cs:line 28
   at Debugging.Familytree.Find(Person person) in C:\Users\Philipp\Documents\Documents\Studium\4. Semester\SWDes\Aufgabe 2-Debugging\FamilyTree.cs:line 28
   at Debugging.Familytree.Find(Person person) in C:\Users\Philipp\Documents\Documents\Studium\4. Semester\SWDes\Aufgabe 2-Debugging\FamilyTree.cs:line 28
   at Debugging.Familytree.Find(Person person) in C:\Users\Philipp\Documents\Documents\Studium\4. Semester\SWDes\Aufgabe 2-Debugging\FamilyTree.cs:line 28
   at Debugging.Program.Main(String[] args) in C:\Users\Philipp\Documents\Documents\Studium\4. Semester\SWDes\Aufgabe 2-Debugging\Program.cs:line 19
```

```
3 references
public static Person Find(Person person)
{
    Person ret = null;
    if (person.LastName == "Battenberg")
        return person;

    ret = Find(person.Mom);
    if (ret != null)
        return ret;
    ret = Find(person.Dad);
    return ret;
}
```

Diese Exception kommt zustande, da das Programm, wie vorhin, die Person immer anhand des „Mom-Zweiges“ von oben nach unten durchgeht, da zuerst Find(person.Mom) aufgerufen wird. Zuerst ist person.Mom Diana Spencer(siehe unten 1.). Danach Franzi Roche (siehe 2.) und zuletzt Ruth Gill(3.). Und dann findet die Funktion keine weitere person.Mom und wirft die Exception.

```
public static Person BuildTree()
{
    return
        new Person { FirstName = "Willi", LastName = "Cambridge", DateOfBirth=new DateTime(1982, 07, 21),
 1.         Mom = new Person { FirstName = "Diana", LastName = "Spencer", DateOfBirth = new DateTime(1961, 07, 01),
                Mom = new Person {FirstName = "Franzi", LastName="Roche", DateOfBirth = new DateTime(1936, 01, 20),
                    Mom = new Person {FirstName = "Ruth", LastName="Gill", DateOfBirth = new DateTime(1908, 06, 07)},
      2             Dad = new Person {FirstName = "Moritz", LastName="Roche", DateOfBirth = new DateTime(1885, 07, 08)}
         3.     },
```

```
public static Person Find(Person person)
{
    Person ret = null;
    if (person.LastName == "Battenberg")
        return person;

    if(person.Mom != null){    1.
    ret = Find(person.Mom);
    if (ret != null)
        return ret;
    }

    if (person.Dad != null){   2.
    ret = Find(person.Dad);
    if (ret != null)
        return ret;
    }
    return null; 3.
}
```

Um dieses Problem müssen wir ein bisschen Code hinzufügen:

1. Für fügen eine Abfrage hinzu ob person.Mom ungleich null ist um die NullReferenceException zu vermeiden, somit überspringt er in diesem Durchlauf den Aufruf von Find(person.Mom).
2. Das gleiche machen wir vor Find(person.Dad), damit hier nicht der gleiche Fehler auftritt.
3. Letztendlich fügen wir ans Ende der Funktion Find() noch return null, für den Fall, dass die darüberliegenden Aufrufe aufgrund unserer Bedingungen alle übergangen werden und somit keinen Rückgabewert liefern.

```
Person ret = null;
if (100 >= (DateTime.Now.Year - person.DateOfBirth.Year) && (DateTime.Now.Year - person.DateOfBirth.Year) <=90)
    return person;
```

Ich habe die Bedingung so geändert, dass in der Bedingung überprüft wird ob das momentane Jahr (2018) minus dem person.DateOfBirth.Year, also dem Geburtsjahr der entsprechenden Person, einerseits weniger als oder gleich 100 entsprechen und andererseits größer oder gleich 90 entsprechen. Doch leider hat das noch nicht zum gewünschten Ergebnis geführt, da er mir nur Willi Cambridge zurückgibt, auf den diese Bedingung nicht zutrifft. Der Übersichtlichkeit halber ersetzte ich die langen Worte durch eine kurze Variable namens age.

```
int age = DateTime.Now.Year - person.DateOfBirth.Year;
if (age >= 90 && age <= 100)
```

Dann erstellte ich eine neue Funktion beziehungsweise verpackte meine bisherigen Änderungen in eine neue Funktion, damit beide Funktionalitäten erhalten bleiben und ich nicht die Eine mit der Anderen überschreibe.

```
public static Person WhoIsThisOld(Person person)
{
    Person ret = null;
    int age = DateTime.Now.Year - person.DateOfBirth.Year;
    if (age >= 90 && age <= 100)
        return person;
    if(person.Mom != null)
        ret = WhoIsThisOld(person.Mom);
    if (ret != null)
        return ret;
    if(person.Dad != null)
        ret = WhoIsThisOld(person.Dad);
    return ret;
}
```

1.

2.

3.

Die Funktion WhoIsThisOld() baute ich von der Vorgehensweise ähnlich wie die Funktion Find() auf.
1. Zuerst gibt man die Bedingung an, dass das age, also das Alter der momentan behandelten Person, größer gleich 90 und kleiner gleich 100 entspricht. Sobald diese Bedingung erfüllt wird, beziehungsweise true ist, wird person zurückgegeben.

2. Danach geht das Programm wie in der Find()-Funktion, immer zuerst den Mom-Zweig entlang und ruft, sofern person.Mom ungleich null ist, die Funktion nochmal mit person.Mom auf.

3. Wenn das Programm mit dem Mom-Zweig fertig ist, macht es das Gleiche mit dem Dad-Zweig.

Sofern, sowohl person.Mom als auch person.Dad gleich null sind und damit 2. Und 3. übergangen werden returned das Programm ret und springt damit wieder im Zweig zurück. Von der neuen Position im Zweig aus, macht das Programm immer wieder die Abfragen, beziehungsweise springt zurück, bis es ein Ergebnis gefunden hat oder einfach eine leere Zeile ausgibt, wenn es für die entsprechende Bedingung keine Lösung gibt.

```
if (age >= 150 && age <= 170)
```

```
PS C:\Users\Philipp\Documents\Documents\Studium\4. Semester\SWDes\Aufgabe 2-Debugging> dotnet run
Ruth Gill
PS C:\Users\Philipp\Documents\Documents\Studium\4. Semester\SWDes\Aufgabe 2-Debugging> dotnet run

PS C:\Users\Philipp\Documents\Documents\Studium\4. Semester\SWDes\Aufgabe 2-Debugging>
```

Zu guter letzt, rufen wir noch die Funktion WhoIsThisOld in der Program.cs Datei auf.

```
Person found = Familytree.WhoIsThisOld(root);
```