Doing More with Classes and Custom Types

Gill Cleeren

CTO Xebia Microsoft Services Belgium

@gillcleeren

Overview

Grouping classes in namespaces

Introducing static data

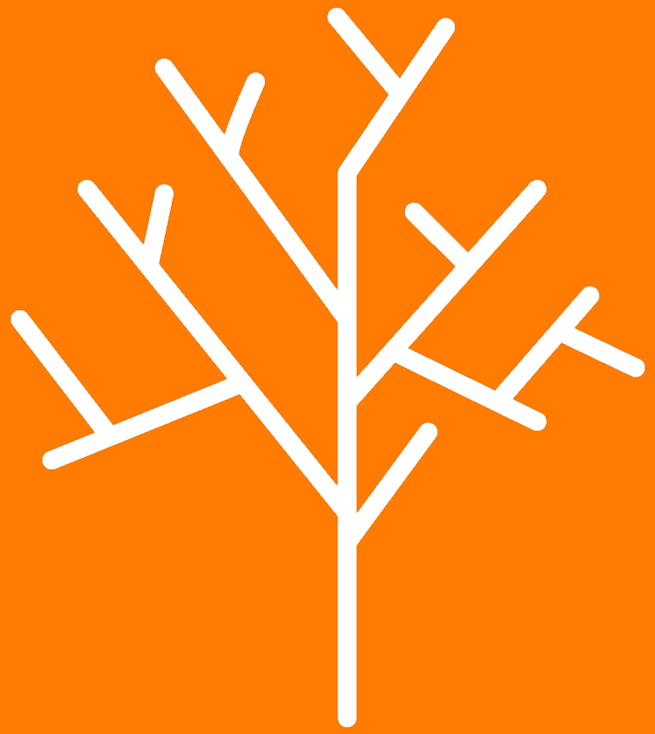
Working with null

Understanding garbage collection

Using a class library

Introducing records

Grouping Classes in Namespaces



There are a lot of types...

Organized in “folders”: namespaces

Avoids naming collisions

Namespaces

- Keep class names separate
- Used throughout .NET
- Organize our own classes in custom namespaces
- Make namespace available through using directive

```
namespace BethanysPieShop.HR
{
    public class Employee
    { }
}
```

Putting a Class into a Namespace

Using File-scoped Namespaces

Introduced with C# 10

Employee.cs

```
namespace BethanysPieShop.HR
{
    public class Employee
    { }
}
```

Employee.cs

```
namespace BethanysPieShop.HR;

public class Employee
{ }
```

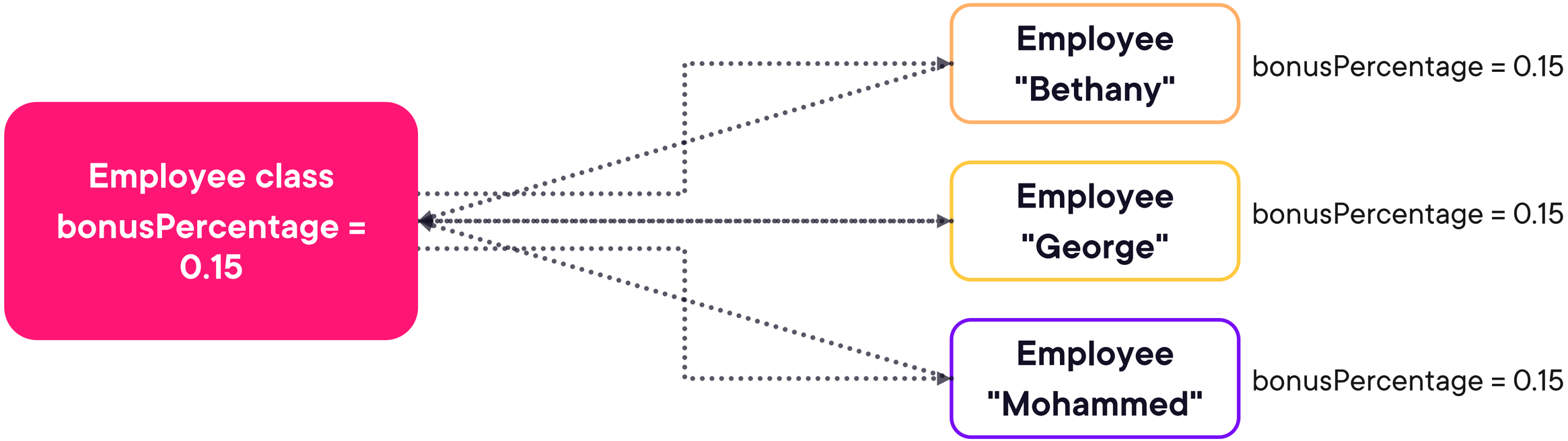
Demo

Grouping multiple classes into namespaces

Introducing the using directive

Introducing Static Data

Objects and Their Data



```
public class Employee
{
    public static double bonusPercentage = 0.15;
}
```

Adding Static Data

```
public class Employee
{
    public static double bonusPercentage = 0.15;
    public static void IncreaseBonusPercentage(double newPercentage)
    {
        bonusPercentage = newPercentage;
    }
}
```

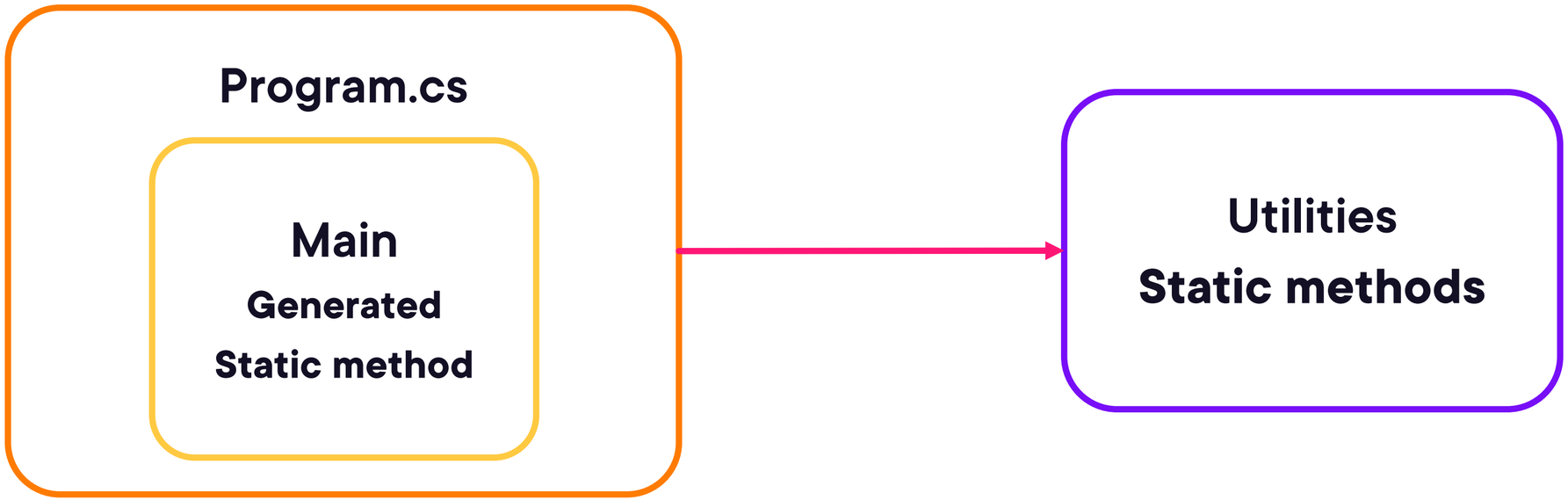
Changing Static Data with a Static Method

```
static void Main(string[] args)
{
    Employee.IncreaseBonusPercentage(0.2);//Note the class name, not an object!
}
```

Invoking a Static Method

Not on an object but on the class instead

Calling Static Methods




```
static void Main(string[] args)
{
  PrintAllEmployeeList();
}
```

```
public static
  void PrintAllEmployeeList()
{
  ...
}
```

Demo

Adding static data

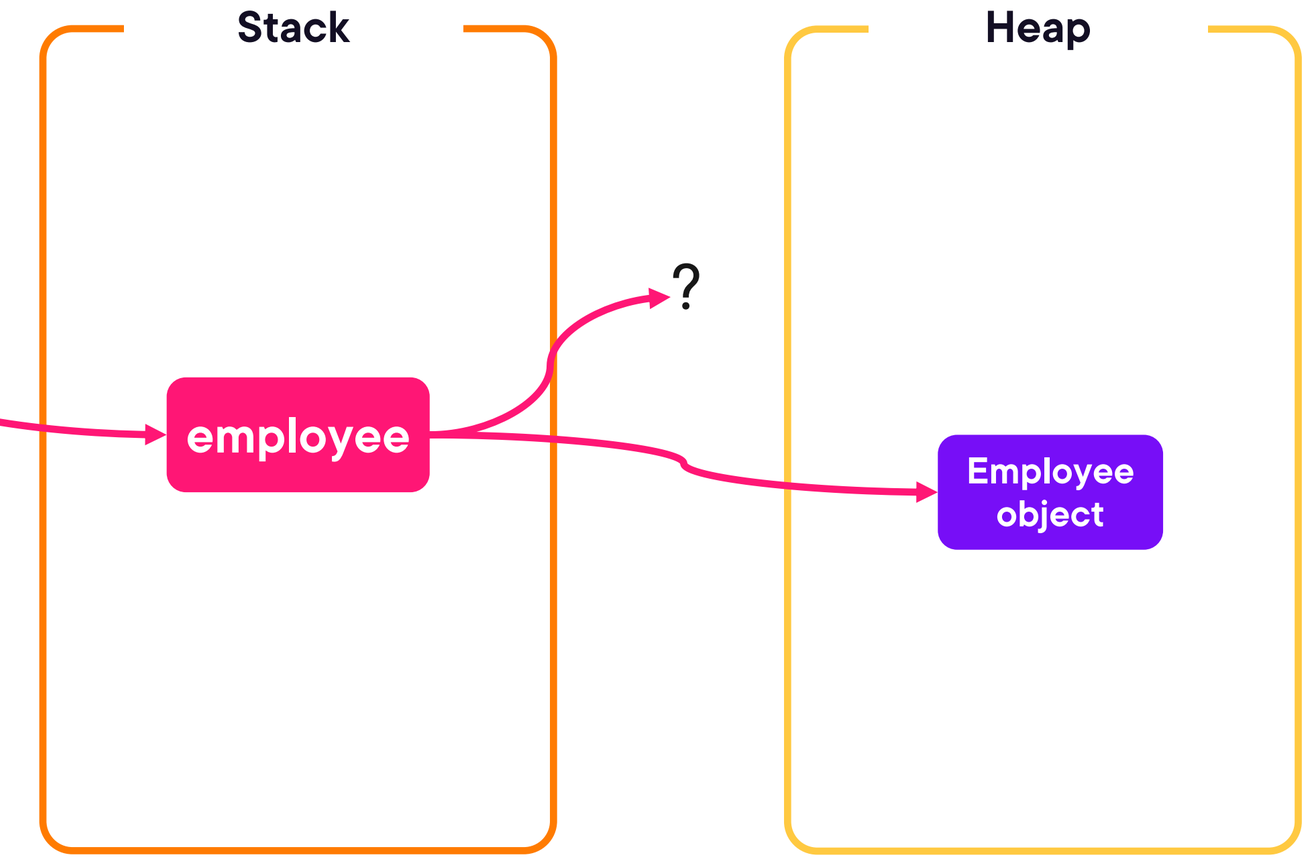
Creating a static method

Using the static functionality from our class

Working with null

Understanding null

```
Employee employee;
//employee is null
employee = new Employee();
```



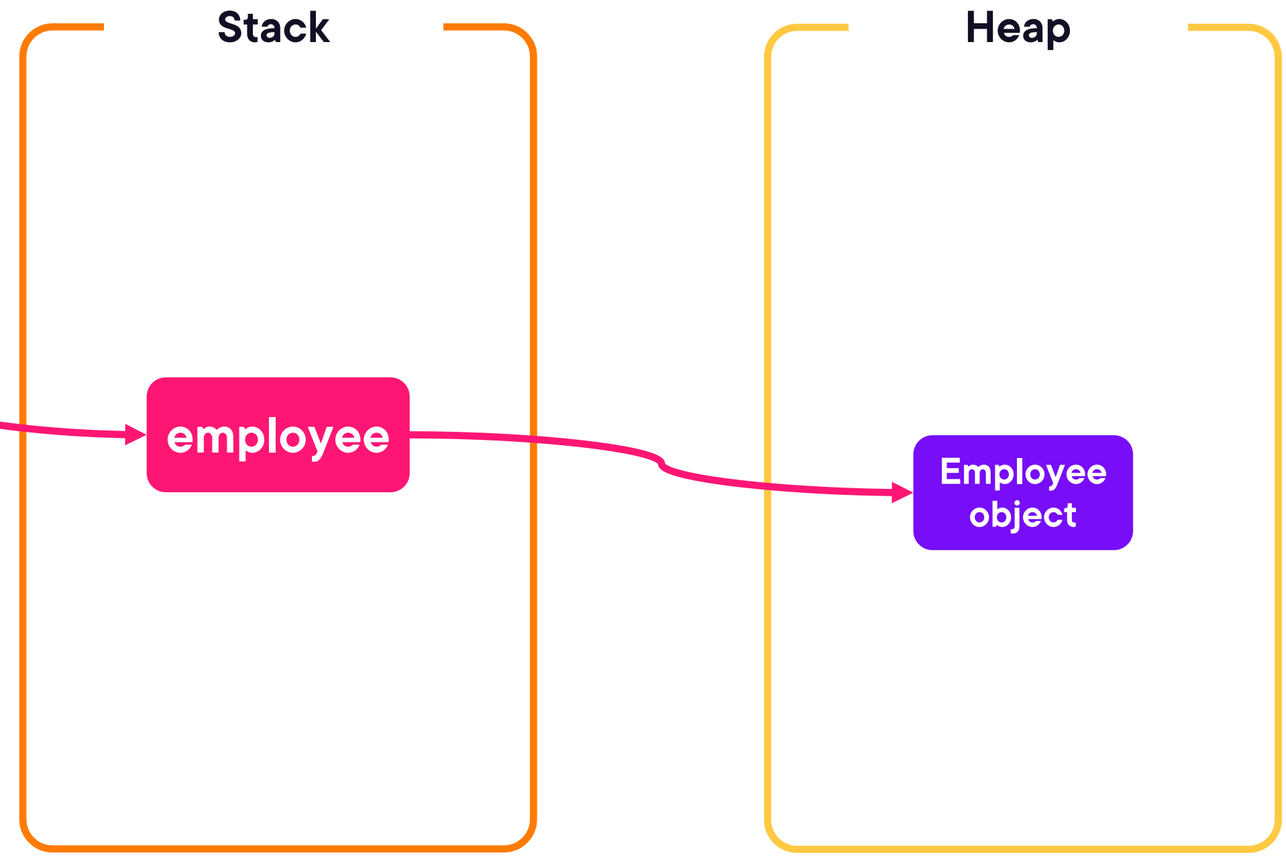
Using a Non-initialized Value

```
Employee employee = null;
employee.PerformWork();//runtime error
```

Setting the Reference to null

```
Employee employee;
//employee is null
employee = new Employee();

employee = null;
```



```
int? a = 10;
int? b = null;
if (b.HasValue)
{
    Console.WriteLine("We have a value");
}
```

Introducing Nullable Value Types

Demo

Handling null references at runtime

Working with nullable types

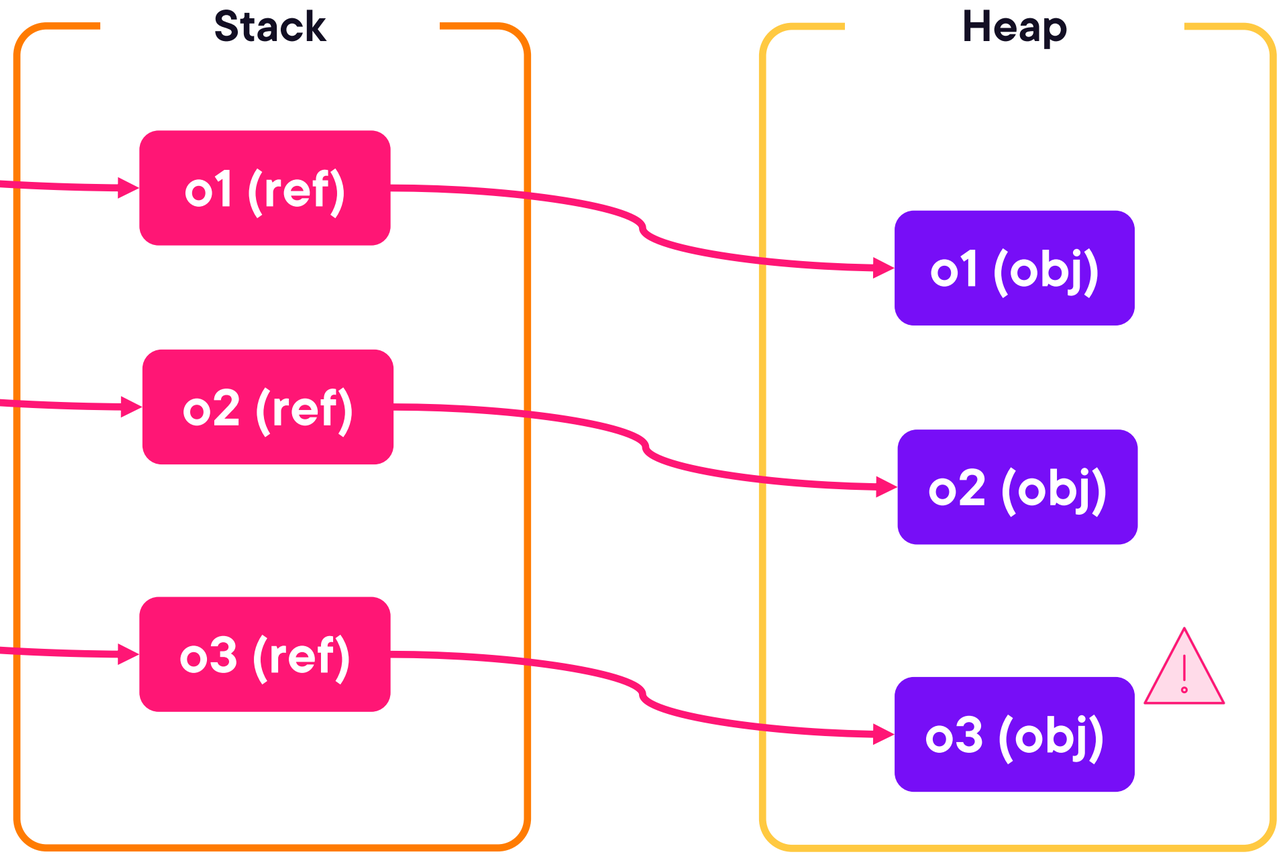
Understanding Garbage Collection

Working with Objects

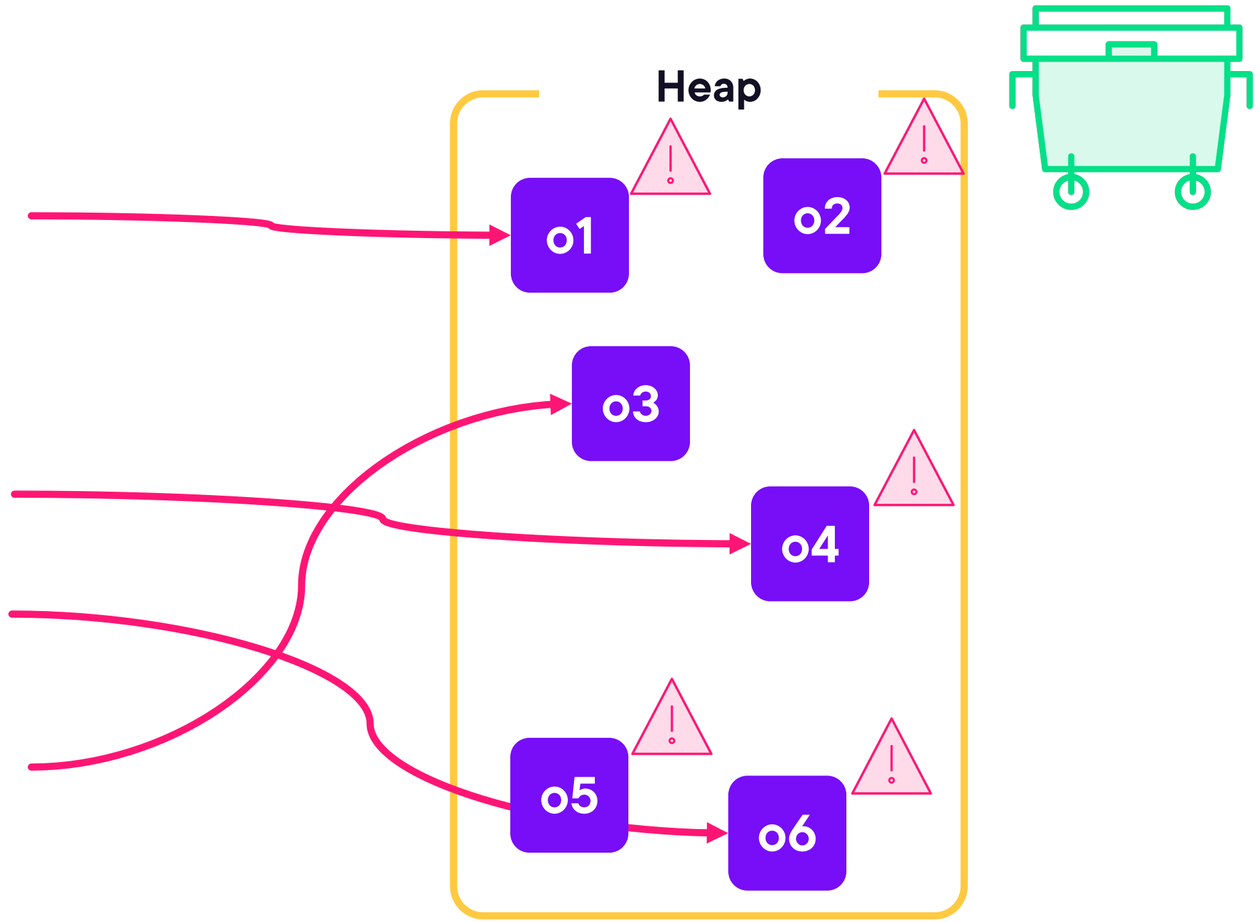
```
object o1 =
  new object();

object o2 =
  new object();

object o3 =
  new object();
```

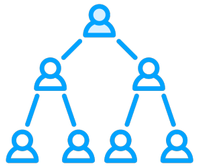


Understanding Garbage Collection



Understanding Garbage Collection

Automatic process, part of CLR



Works with several generations

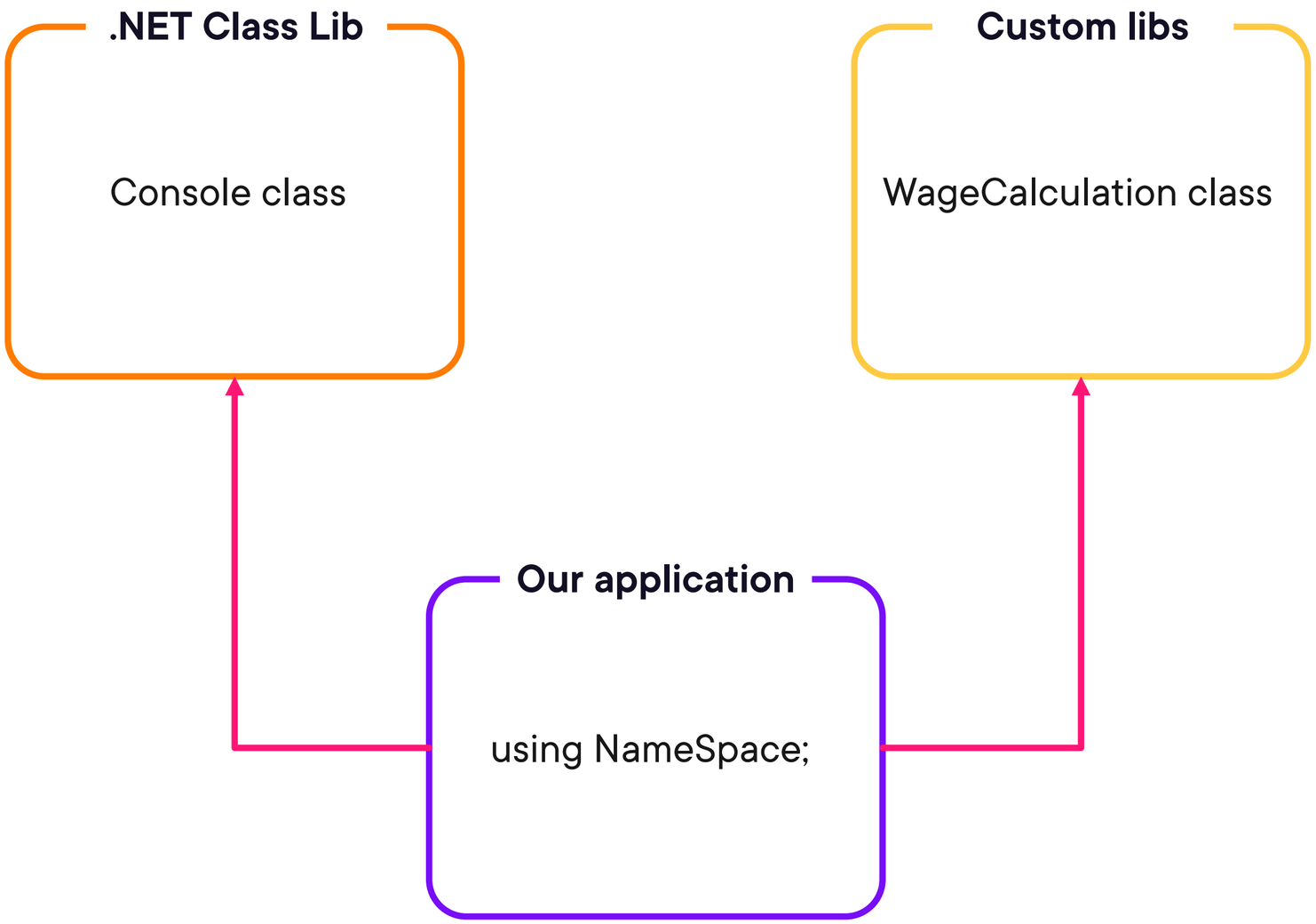
Can be triggered using GC.Collect(), often not required

Demo

Looking at garbage collection

Using a Class Library

Using a Class from an External Library



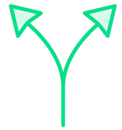
Demo

Using a class library

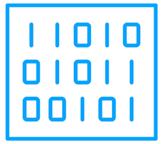
Introducing Records

Introducing C# Records

New reference type



Can replace class



Aimed at “just” containing data, can contain other members though

Comes with additional functionality built-in (generated)

```
public record Account;
public record class Account;
public record struct Account;
```

Creating a Record

```
public record Account(string AccountNumber);
```

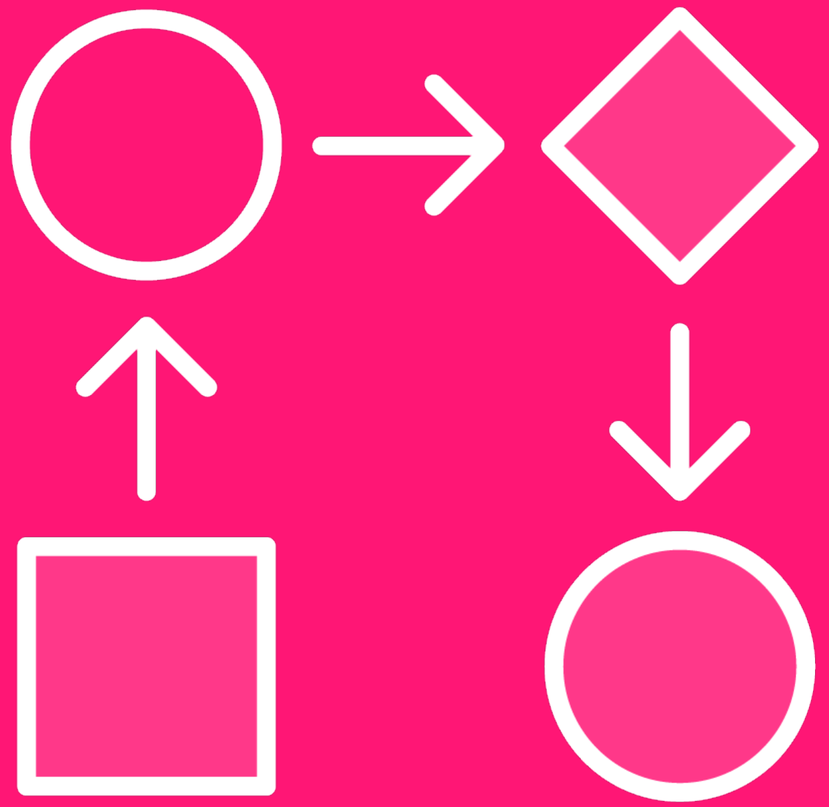
Revisiting Primary Constructors

Positional record

```
Account newAccount = new("123-456");
```

Instantiating a Record

Arguments are required to be passed in



Why records?

Passing around “just” data

Used for data that shouldn’t be changed after creation

Advantages of Using Records

Immutability

Value-based equality

Concise

```
Employee emp1 = new Employee("Bethany");
emp1.FirstName = "Gill";//error
```

Immutable Records

Positional records
Data can be safely read but not changed

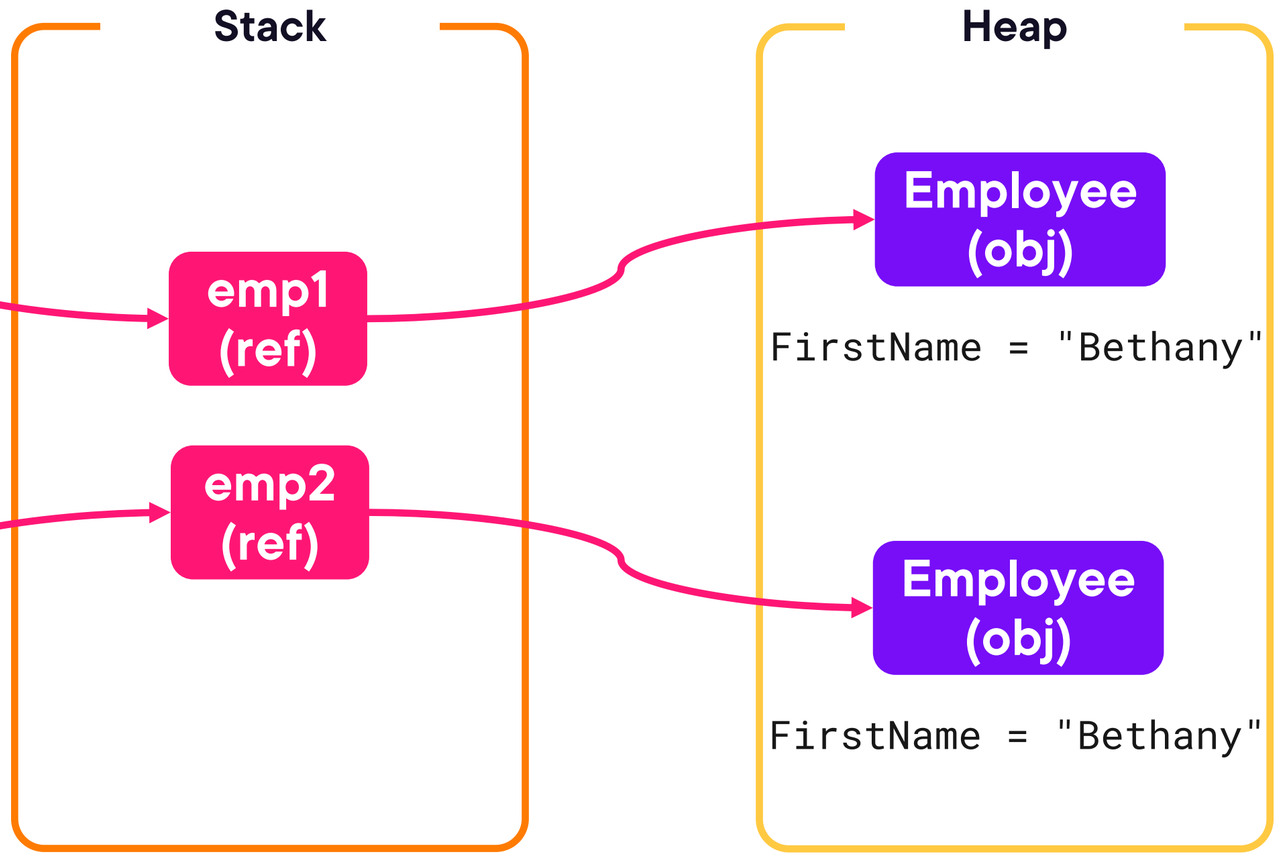
Understanding Equality with Classes

```
Employee emp1 =
new Employee();

emp1.FirstName =
"Bethany";


Employee emp2 =
new Employee();

emp2.FirstName
= "Bethany";
```



```
Employee emp1 = new Employee();
emp1.FirstName = "Bethany";

Employee emp2 = new Employee();
emp2.FirstName = "Bethany";

bool areEqual = emp1 == emp2;//false
```

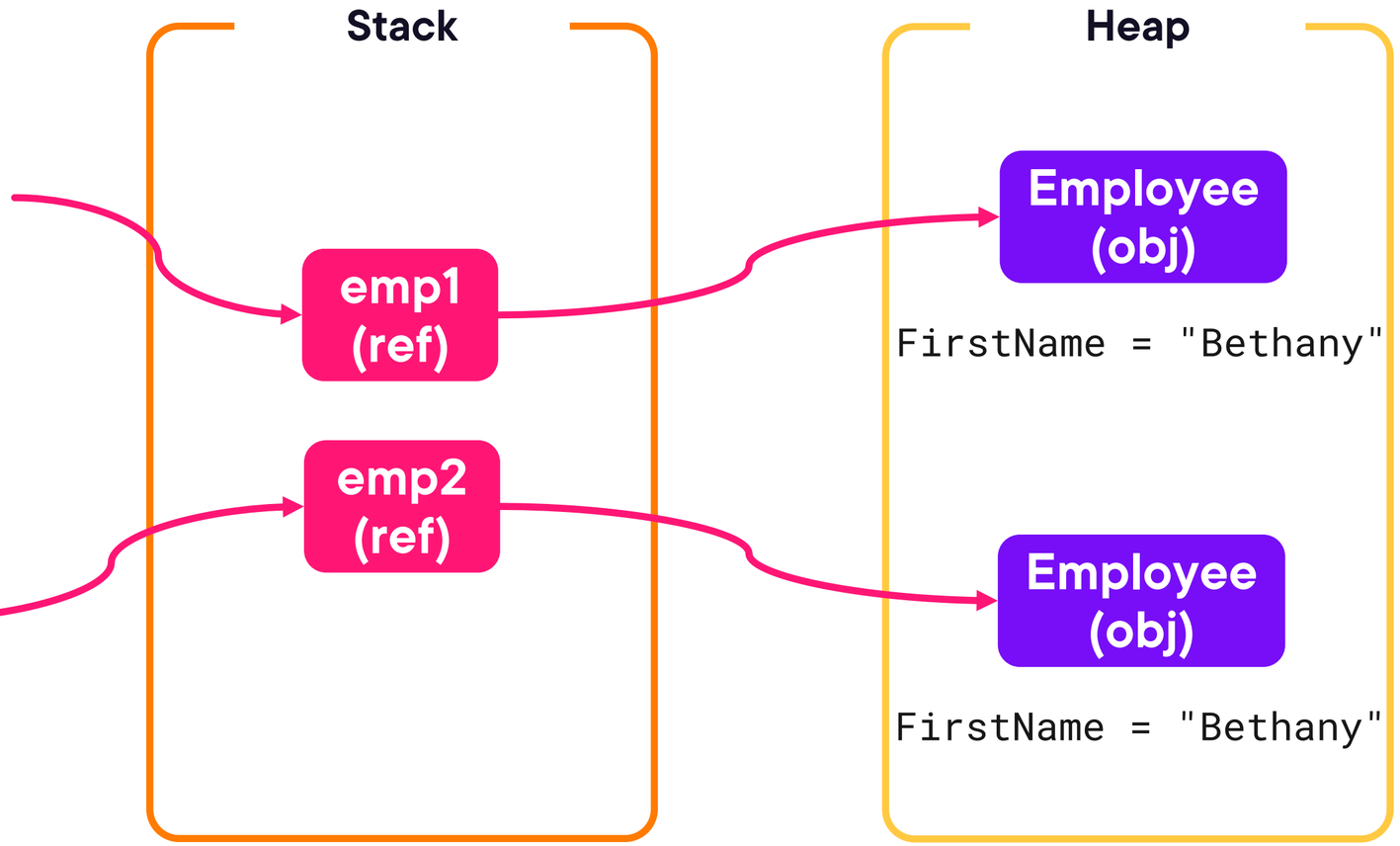
Classes Use Memory-based Equality

Types need to be the same
Variables point to same object in memory

Understanding Equality with Records

```
Employee emp1 = new
Employee("Bethany");
```

```
Employee emp2 = new
Employee("Bethany");
```



```
Employee emp1 = new Employee("Bethany");
Employee emp2 = new Employee("Bethany");

bool areEqual = emp1 == emp2;//true
```

Records Use Value-based Equality

Type definitions are the same
Values for those are equal

Demo

Using a record

Summary

Namespaces are used to group classes

Static data is class-level data

References can be null

- Can cause null reference exceptions
- Garbage collection will clean up unused objects

Use classes from an external library

Up Next:

Working with arrays