Applying Inheritance to C# Types

Gill Cleeren
CTO Xpirit Belgium

@gillcleeren | www.xpirit.com/gill

Agenda

Understanding inheritance

Creating a base and a derived type

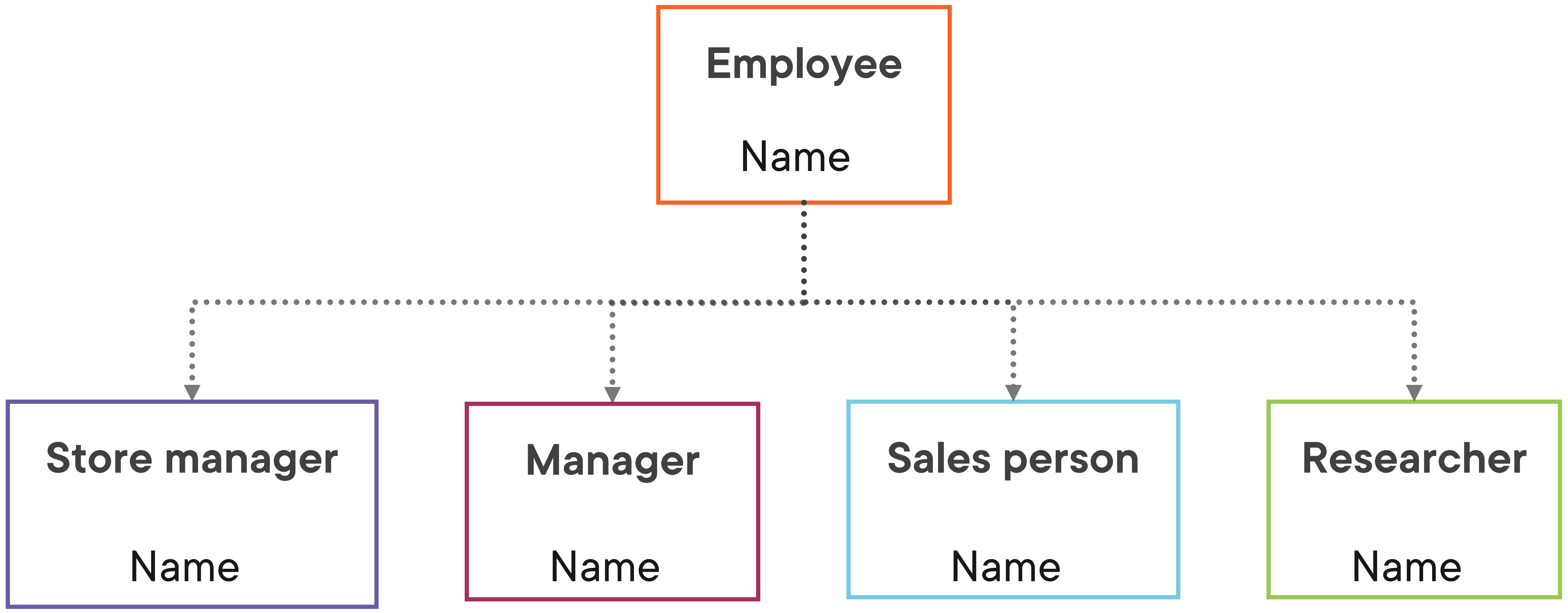
Understanding polymorphism

Working with abstract and sealed classes

Inheriting from System.Object

Understanding Inheritance

Different Types of Employees



Introducing inheritance

Important concept in object-oriented development

Object gets data and functionality from parent

Using Inheritance in C#

Parent (or base) and derived class

Reuse code

Easier to maintain

Can be one or more levels deep

Creating a Base and a Derived Type

```
public class BaseClass
{
}

public class DerivedClass: BaseClass
{
}
```

Base and Derived Classes

Creating the Base and Derived Class

Employee

```
public class Employee
{

}
```

Manager

```
public class Manager: Employee
{

}
```

Accessing the Base Class Members

```
public class Employee
{
   public string name;

   public void PerformWork()
   {

   }
}
```

```
public class Manager: Employee
{
   public void DisplayManagerData()
   {
      Console.WriteLine(name);
   }
}
```

Revisiting Access Modifiers

public

private

protected

Accessing the Base Class Members

```
public class Employee
{
   private string name;

   public void PerformWork()
   {

   }
}
```

```
public class Manager: Employee
{
   public void DisplayData()
   {
      Console.WriteLine(name);//error
   }
}
```

Accessing the Base Class Members

```
public class Employee
{
   protected string name;

   public void PerformWork()
   {

   }
}
```

```
public class Manager: Employee
{
   public void DisplayData()
   {
      Console.WriteLine(name);//ok
   }
}
```

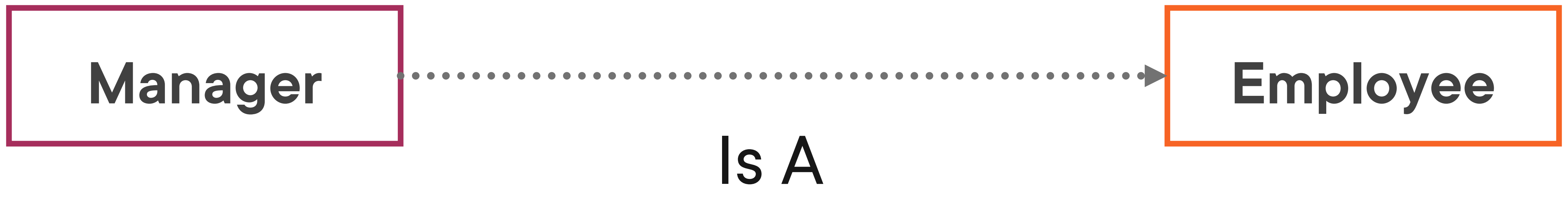
Demo

Creating a base class

Building a derived class

Access the base class' members

The “Is-A” Relation



```
Manager m1 = new Manager();//Manager derives from Employee

Researcher r2 = new Researcher();//Researcher derives from Employee

m1.PerformWork(); //will call PerformWork() on the base Employee class

r2.PerformWork(); //will call PerformWork() on the base Employee class
```

Using the Base Type

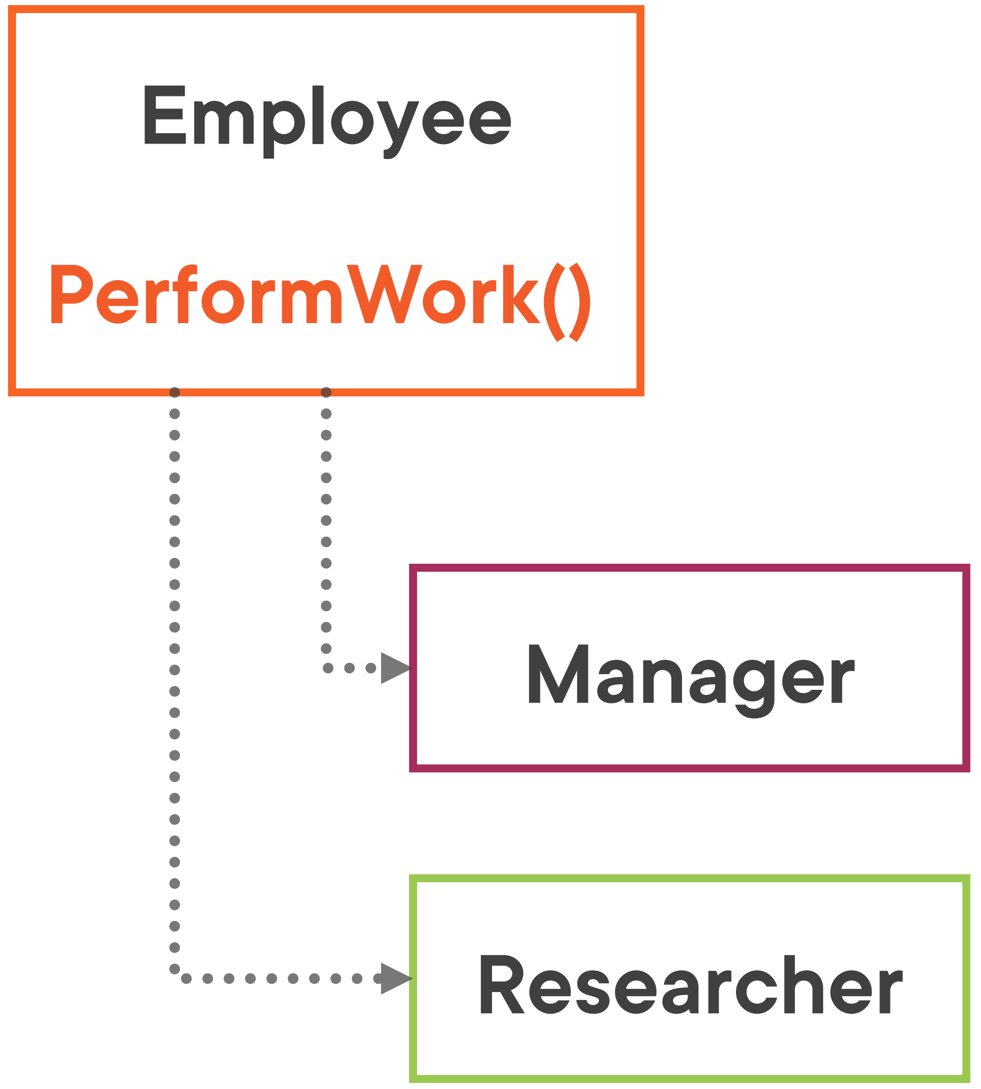
Using the Is-A relation

Demo

Using the “Is-A” relation

Using Polymorphism

Using a Base Method



```
public class Employee
{
    public void PerformWork()
    { ... }
}
```

```
public class Manager: Employee
{ }
```

```
public class Researcher: Employee
{ }
```

```
Manager m1 = new Manager();
m1.PerformWork();
Researcher r1 = new Researcher();
r1.PerformWork();
```

The invoked method will be the same for all inheriting types.

Introducing Polymorphism

Override a base class method with a new implementation

“Poly” & “morph”

Uses virtual and override keywords

Introducing Polymorphism

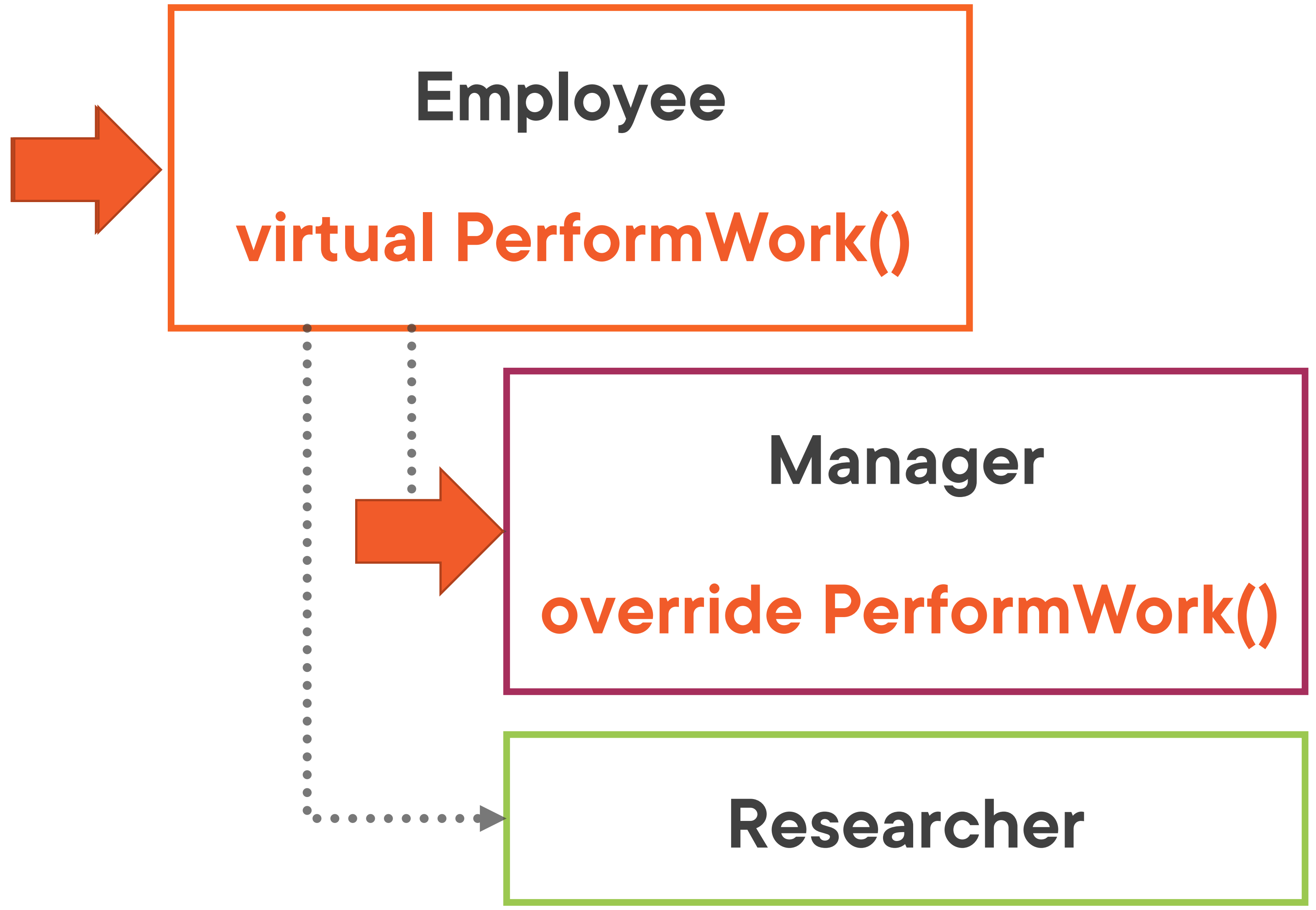
Employee

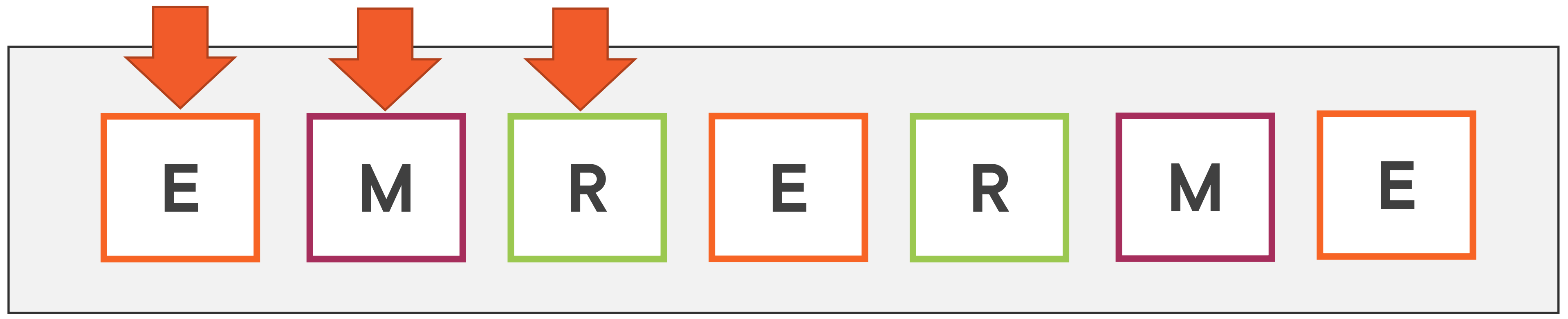
```
public class Employee
{

   public virtual void PerformWork()
   { ... }

}
```

Manager

```
public class Manager: Employee
{

   public override void PerformWork()
   { ... }

}
```

Using Polymorphism in C#

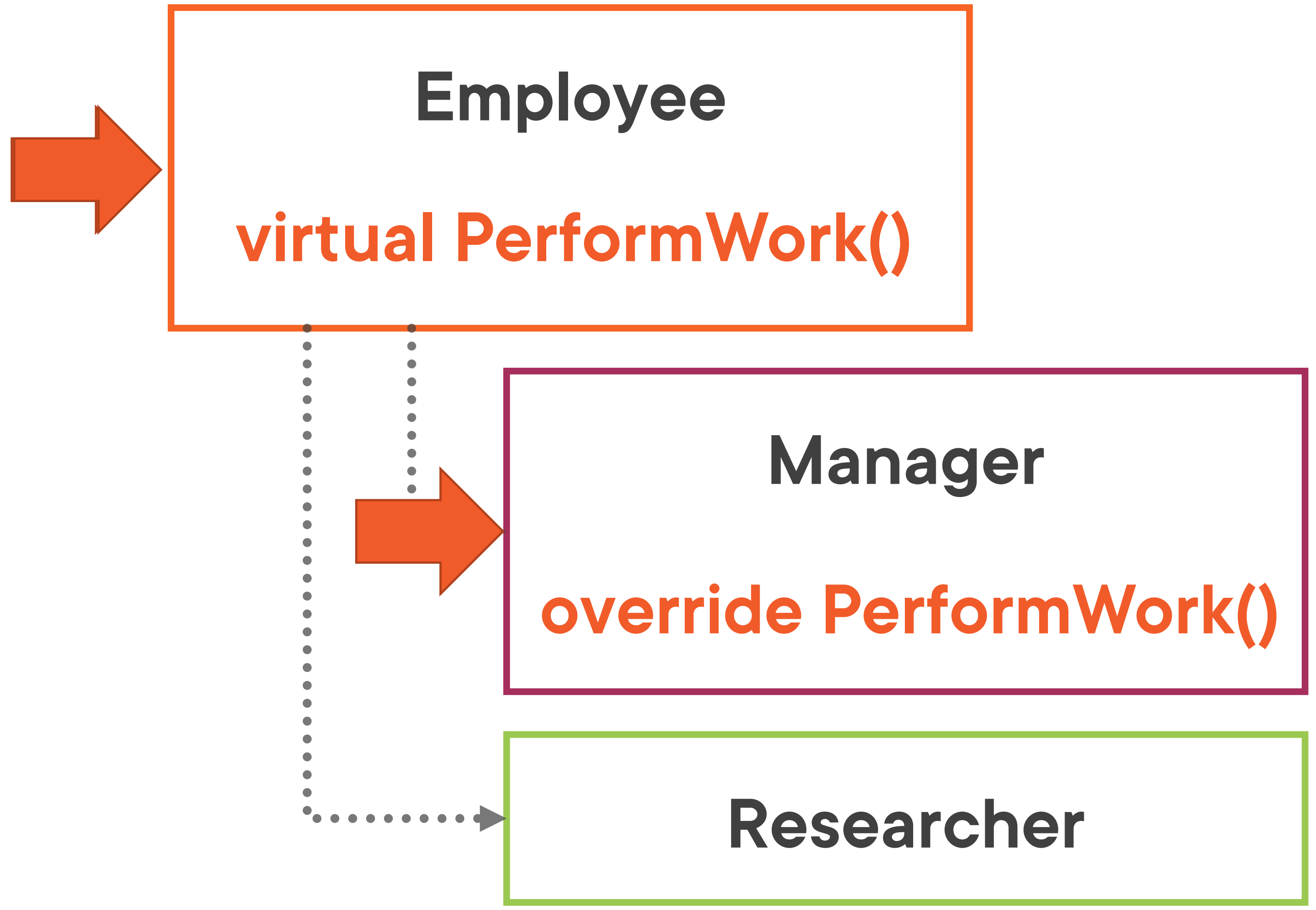


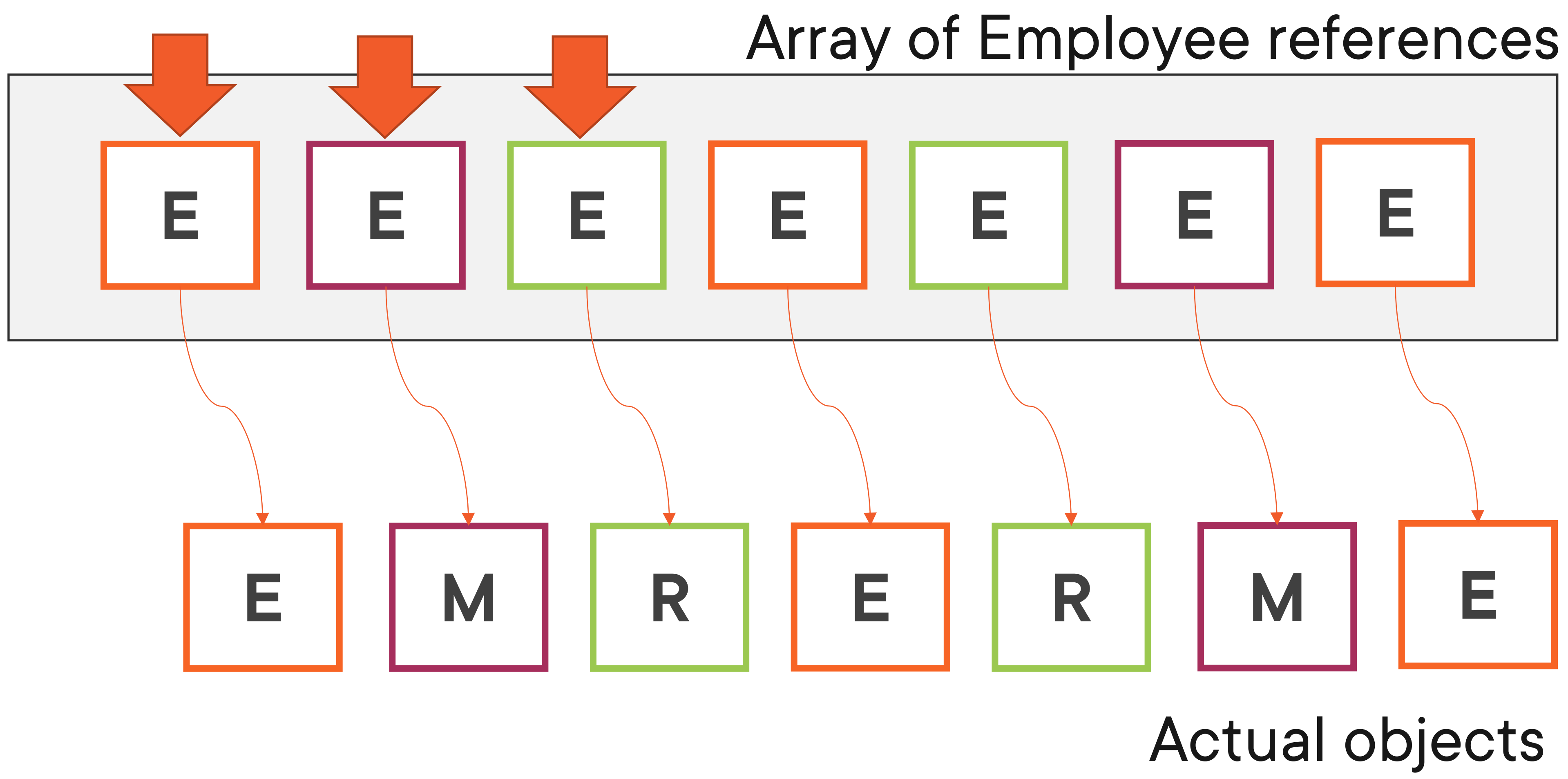


```
Employee e1 = new Manager();

Employee e2 = new Researcher();

e1.PerformWork();//will call the most specific version, so the one on Manager

e2.PerformWork();//will call the most specific version, so the one on Researcher

e1.AttendManagementMeeting(); //error if defined on Manager derived type
```

Using Polymorphism

Looping over an Array of Employee References





Demo

Adding virtual and override

Using polymorphism

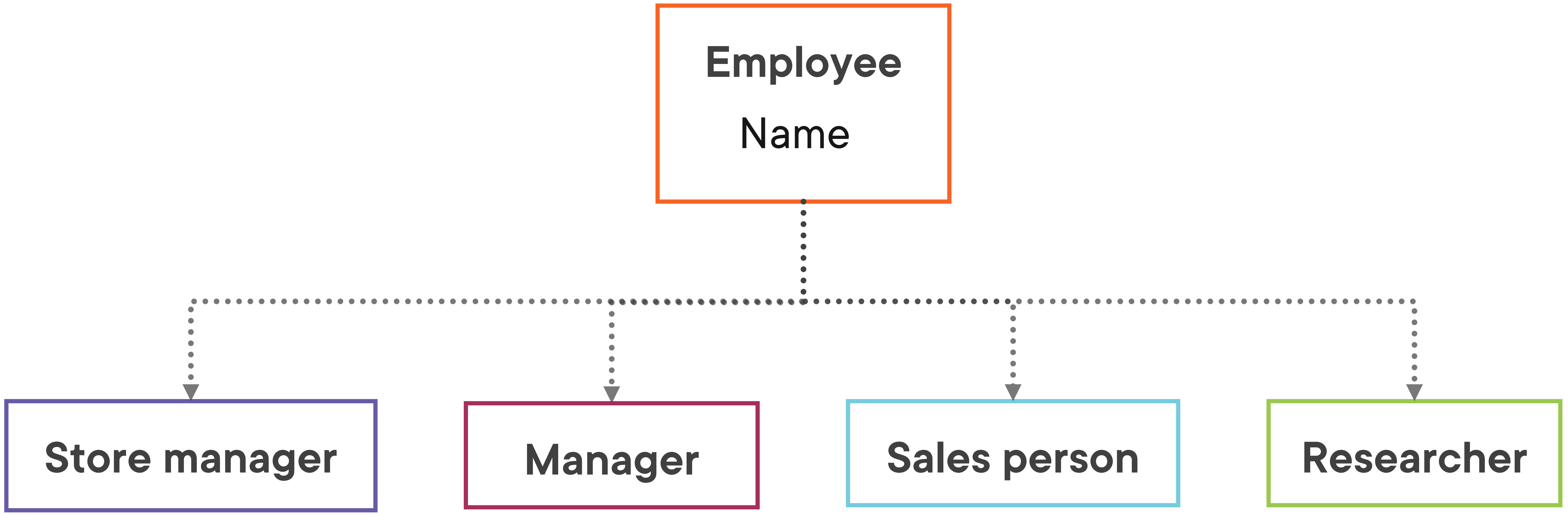
Working with Sealed and Abstract Classes

Demo

Creating a sealed class

Trying to inherit from a sealed class

The Idea behind Abstract Classes



```
Employee e1 = new Employee();
//Might not actually be a real
//concept and thus abstract
```

Introducing Abstract Classes

Used to model a concept, something abstract

Missing complete implementation

Can’t be instantiated

```
public abstract class Employee
{

    public abstract void ReceiveWage();

    public virtual void PerformWork
    {

        ...

    }

}
```

Creating an Abstract Class

```
Employee employee = new Employee();//won’t compile
```

Instantiating Abstract Classes

Will result in a compile-time error

Inheriting from an Abstract Class

Implementing Abstract Methods is Required

Employee.cs

```
public abstract class Employee
{

    public abstract void ReceiveWage();

}
```

Manager.cs

```
public class Manager: Employee
{

    public override void ReceiveWage()
    {

        ...

    }

}
```

Demo

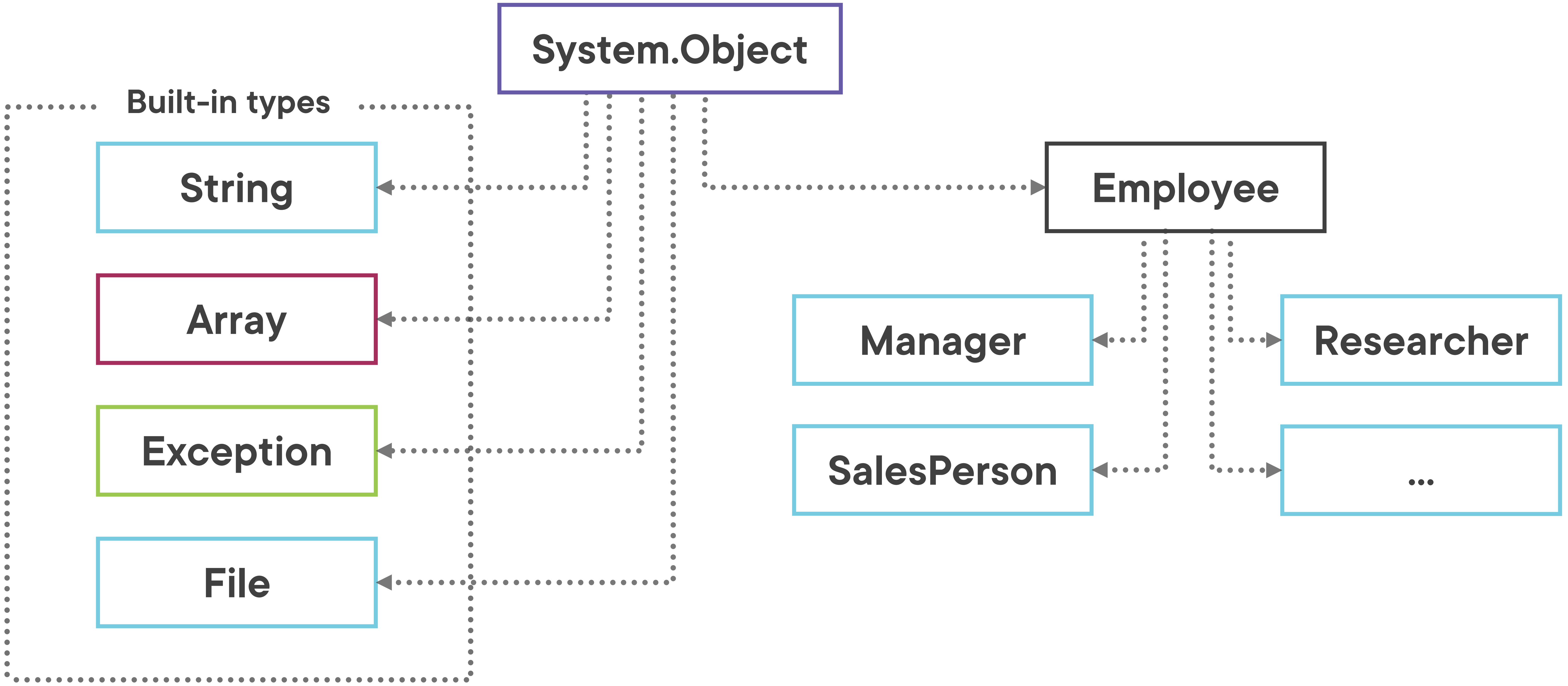
Introducing an abstract class

Adding an abstract method

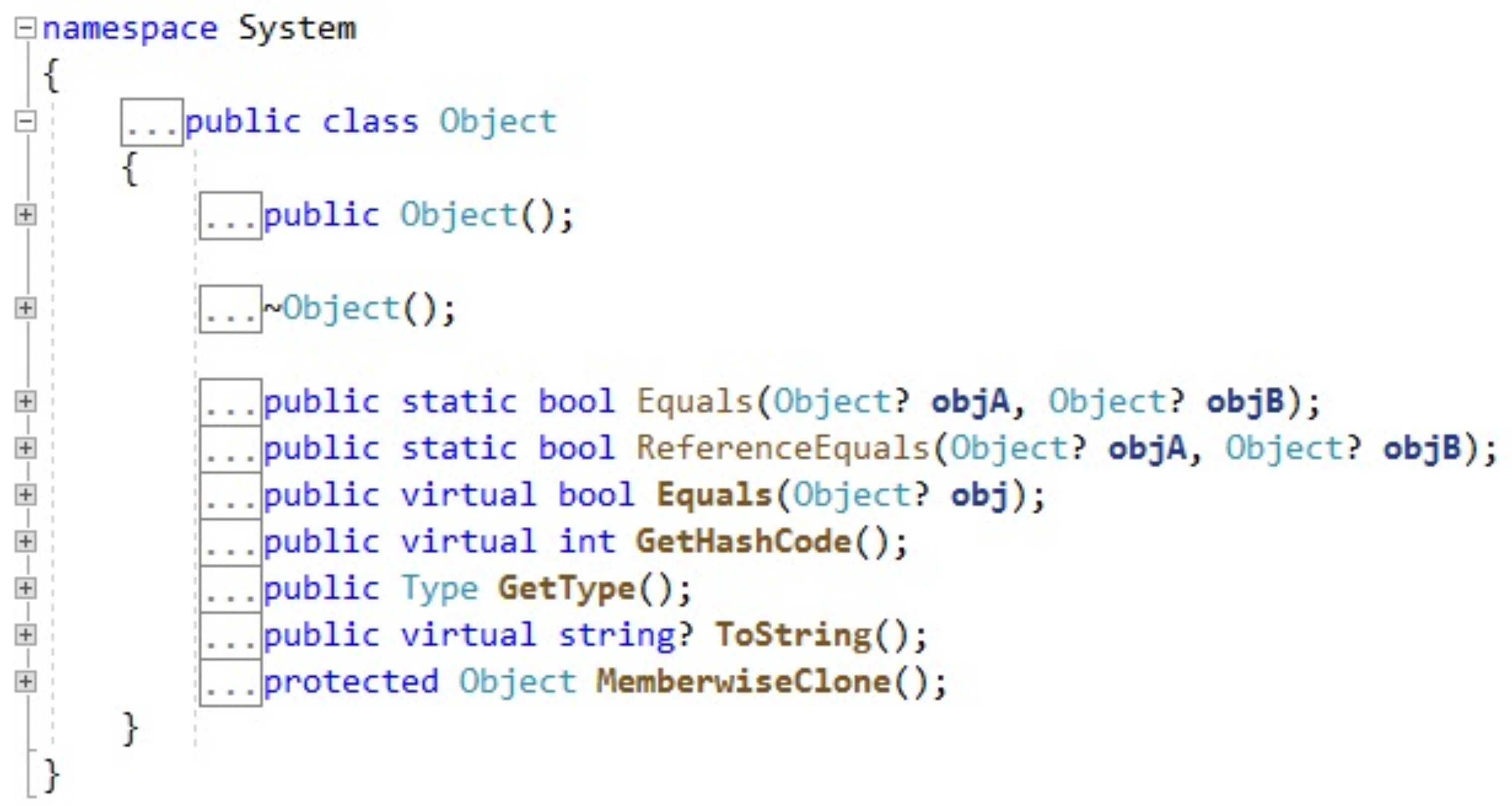
Deriving from the abstract type

Inheriting from System.Object

Going to the Base Object



Members Defined on System.Object



```
namespace System
{
    ...public class Object
    {
        ...public Object();

        ...~Object();

        ...public static bool Equals(Object? objA, Object? objB);
        ...public static bool ReferenceEquals(Object? objA, Object? objB);
        ...public virtual bool Equals(Object? obj);
        ...public virtual int GetHashCode();
        ...public Type GetType();
        ...public virtual string? ToString();
        ...protected Object MemberwiseClone();
    }
}
```

```
object a = new Employee();
```

Everything Is A System.Object

Demo

Inheriting from System.Object

Summary

Deriving from a base class will bring reuse

Access modifiers define what the derived class can access

Virtual and override introduce polymorphism

Everything inherits from System.Object

Resources

Other relevant courses in the C# path:

- Object Oriented development in C#
  - Deborah Kurata
- C# Generics
  - Thomas Huber
- Working with Arrays and Collections in C#
  - Simon Robinson

Up next:

Using interfaces