

# Creating the Classes



**Gill Cleeren**  
CTO Xpirit Belgium  
[@gillcleeren](https://twitter.com/gillcleeren)

# Overview



**Creating the Product class**

**Using composition**

**Working with class-level members**

**Splitting into multiple partial classes**



# Creating the Product class



# The Class Template

```
public class MyClass
{
    private int counter;

    public void MyMethod()
    {
        //Do something great here
    }
}
```



```
public class Product  
{  
    ...  
}
```

## The Product Class



# The Class Access Modifiers

**public**

**protected and  
private**

**internal**





**Can I leave out the  
access modifier?**

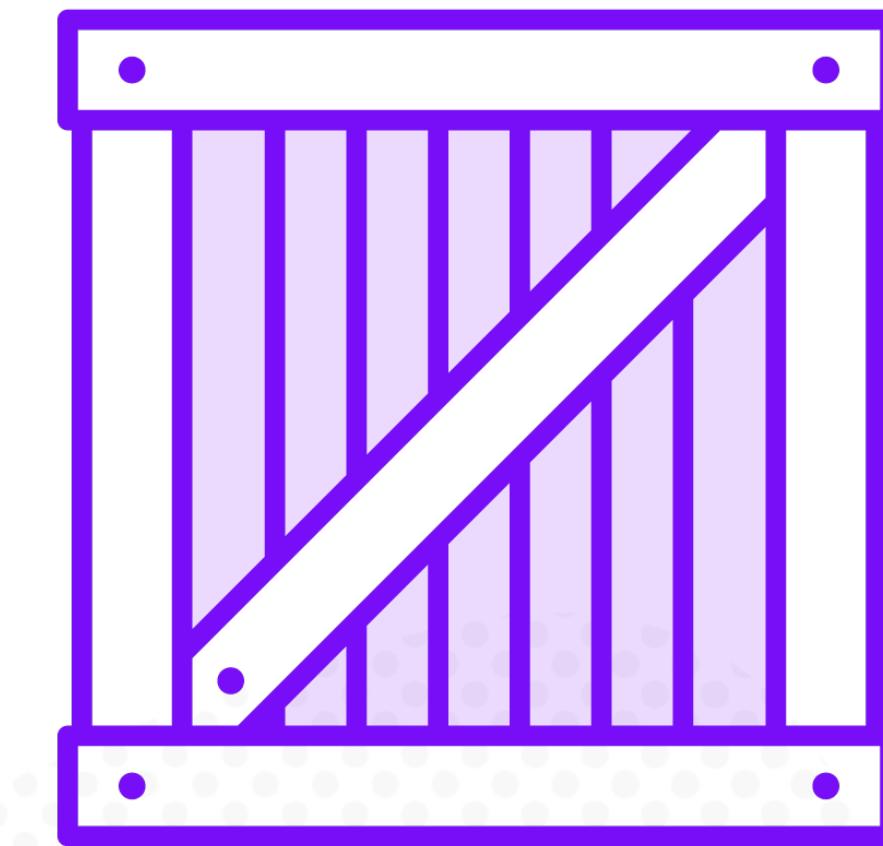
**internal by default**

**Also now default when creating a  
new class using Visual Studio**



# The Product Class: Data

**Id**  
**Name**  
**Description**  
**Maximum in stock**  
**Price & currency**  
**Unit type**  
**Current amount in stock**  
**Low on stock?**



```
public class Product
{
    private int id;
    private string name = string.Empty;
}
```

## **Adding Fields (aka Instance Variables )**

**Typically, private (also default)**  
**Hiding the information inside the class > Encapsulation**  
**Must be accessed through methods or properties**



# Possible Access Modifiers on a Field

**private**

**public**

**protected**

**internal**





## Using encapsulation

**Change and access through public interface**

**Some data is never exposed and thus only used for inner workings**

**Through access modifiers, we apply encapsulation**



## Demo

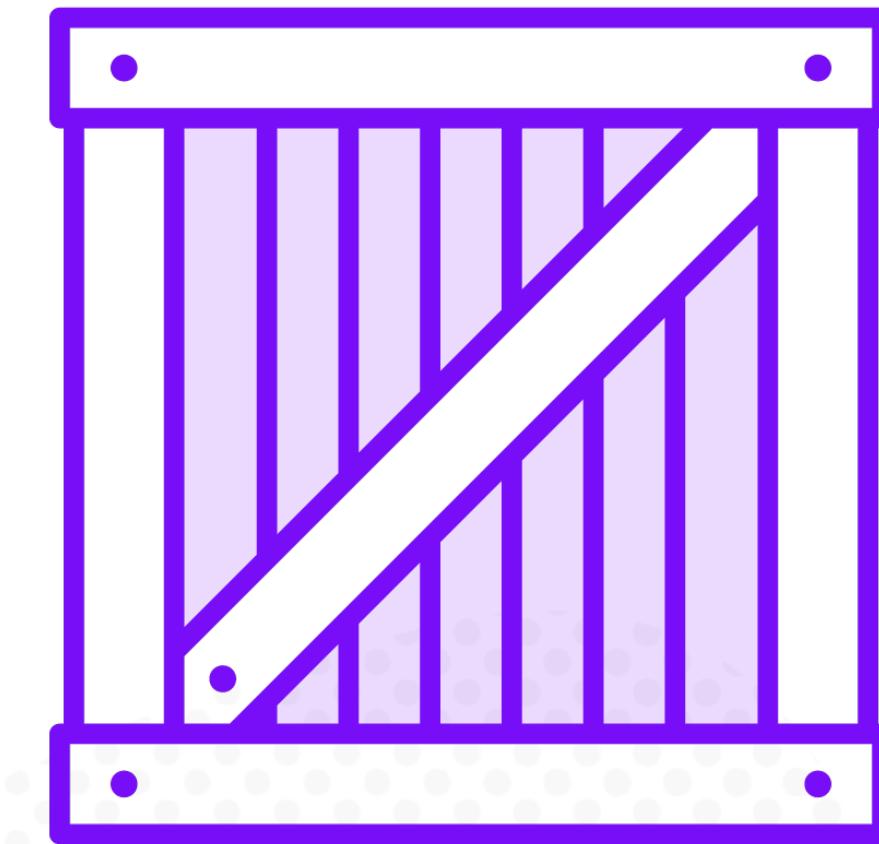


**Creating the class**  
**Adding fields**



# The Product Class: Functionalities

- “Use” product
- Add new product to inventory
- Alert if low on stock
- Display the details of a product (short & long)
- Increase stock when order arrives



```
public class Product
{
    public void IncreaseStock()
    {
        amountInStock++;
    }
}
```

## Adding a Method

**Public if part of the public interface**

**Can work with private fields to change state of object they belong to**



# The Method Template

```
<access modifier> <return type> Method_Name (Parameters)
{
    //method statements
}
```



# Demo



## Adding methods



# Using “getter and setter” Methods

```
public class Product
{
    private string name;

    public string GetName()
    {
        return name;
    }

    public void SetName(string n)
    {
        name = n;
    }
}
```





## **Can we only use methods?**

**Some data might need to be part of public interface of class**

**Properties wrap data and can control its value**



```
public class Product
{
    public string Name
    {
        get { return name; }
        set
        {
            name = value; //custom logic can be included here
        }
    }
}
```

## Using a Property

**Combination of private field and public method**

**Contain get and set accessors**

**Can contain logic to check value**



# Analyzing a Full Property

```
public string Name
{
    get { return name; }
    set
    {
        name = value;
    }
}
```



# Converting to Auto-Implemented Properties

Also known as automatic properties

Product.cs

```
public string Name  
{  
    get { return name; }  
    set  
    {  
        name = value;  
    }  
}
```

Product.cs

```
public string Name { get; set; }
```



```
public string Name  
{  
    get {return name;}  
}
```

◀  **Readonly property**

```
public string Name { get; private set; }
```

◀  **Private set**



# Demo



**Adding properties**

**Using properties inside the class**



# Creating Objects

```
Product product = new Product();
```



```
public Product()
{
    maxAmountInStock = 100;
    name = string.Empty;
}
```

## Adding a Constructor

**Used to instantiate an object with initial values**

**Numeric to 0**

**Char to '\0'**

**Boolean to false**

**References to null**





## What happens if we omit the constructor?

A default constructor is created for us

Only if we don't include another constructor!



```
public Product()  
{ }
```

## The “Default” Constructor

**Not visible**

**Not created when any other constructor is created**



```
public Product(int id, string name)
{
    Id = id;
    Name = name;
}
```

## Passing Parameters to the Constructor



# Demo



## Adding constructors





**Do we always need to  
find unique names?**

Combination of parameters (number,  
order, data types) must be unique

Names can be reused

Method overloading



```
public void IncreaseStock()  
{  
    AmountInStock++;  
}  
  
public void IncreaseStock(int amount)  
{  
    //implement method here  
}
```

## Using Method Overloading



```
public Product(int id) : this(id, string.Empty)  
{ }
```

```
public Product(int id, string name)  
{  
    Id = id;  
    Name = name;  
}
```

## Using Constructor Overloading



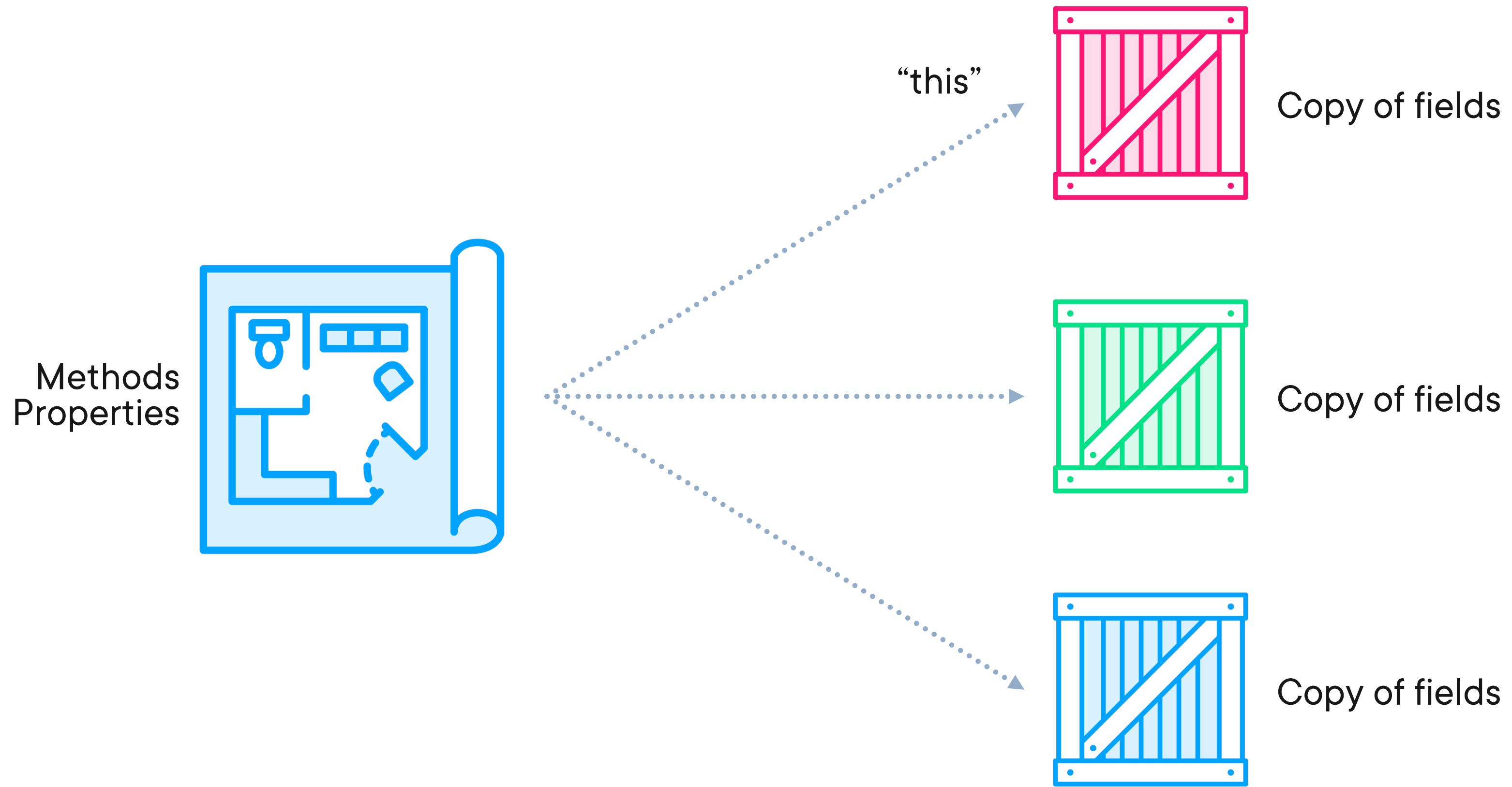
# Demo



**Using method and constructor overloading**



# How Objects Are Created



```
public Product(int id, string name)
{
    this.Id = id;
    this.Name = name;
}
```

## Working with this



# Demo



**Using the this reference**



```
public class Product(int id, string name)
{
    public int Id
    {
        get
        {
            return id;
        }
    }

    public string Name { get; set; } = name;
}
```

## Primary Constructors

Introduced in C# 12

Way of declaring a constructor whose parameters are available in body of the type



```
public class Product(int id, string name)
{
    public Product(int id, string name, decimal price)
        : this(id, name)
    {
    }
}
```

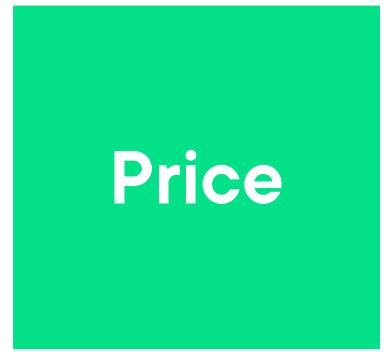
## Adding an Overloaded Constructor



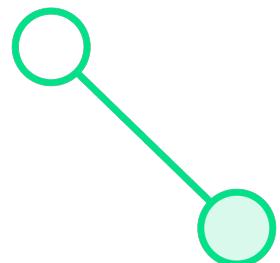
# Using Composition



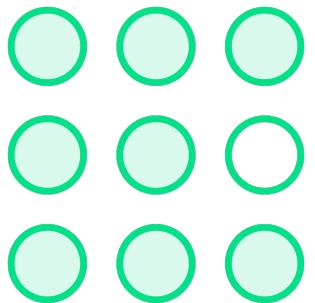
# A Product “Has-A” Price



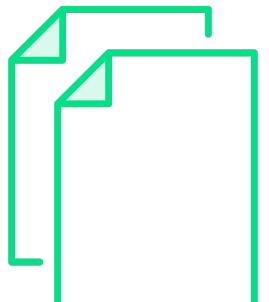
# Using Composition



**Association of objects of different classes**



**Type may not be able to exist standalone**



**Can be reused in multiple classes**



## Demo



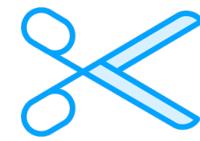
**Adding the Price class**  
**Applying composition**



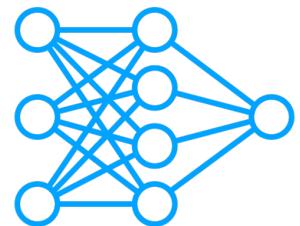
# Splitting into Partial Classes



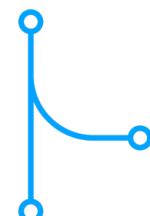
# Using partial



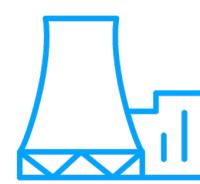
**Physically splitting up over multiple files**



**Classes, structs and interfaces support partial**



**Combined upon compilation**



**Useful if parts of the class are generated**



# Using the partial Keyword

Product.cs

```
public partial class Product  
{  
}
```

Product2.cs

```
public partial class Product  
{  
}
```



# Demo



**Creating a second Product class using  
partial**



# Adding the Order Class



# Demo



## Adding the Order class



# Summary



**Classes are the main building block in C#**

**Contain**

- Fields
- Methods
- Properties
- Constructors

**Encapsulation is applied using access modifiers**

**Composition creates the “Has-A” relation**

**Can be partial**



**Up Next:**

# **Using and Testing the Classes**

---

