# Приемы обеспечения технологичности ПО

Юнит-тесты

Под технологичностью понимают качество проекта ПО, от которого зависят трудовые и материальные затраты на его реализацию и последующую модификацию

Хороший проект сравнительно быстро и легко кодируется, тестируется, отлаживается и модифицируется

Технологичность ПО определяется следующим:

Проработанностью его моделей

Уровнем независимости модулей

Стилем программирования

Степенью повторного использования кода

Проработанность модели – чем лучше проработана модель, тем четче определены подзадачи и структуры данных, проще проектирование и меньше вероятность ошибок

Независимость моделей – чем выше независимость модулей, тем их легче понять, реализовывать, модифицировать, а также находить ошибки и их исправлять

Стиль программирования – под этим понимают стиль оформления программы, ее структурность, что существенно влияет на читаемость кода и кол-во ошибок программирования

Увеличение степени повторного использования кода, хорошо если это использование ранее разработанных библиотек, а также если это унификация кодов текущей разработки

Если же степень исходного кода повышается искусственно, то технологичность снижается.

Высокая технологичность проекта особо важна, если продукты рассчитаны на многолетнее использование

# Модули и их свойства

Модулем называется автономно компилируемая программная единица

1. Сцепление – является мерой взаимозависимости модулей, которое определяет то, насколько хорошо модули отделены друг от друга

Различают 5 видов сцепления:

1. Сцепление по данным – предполагает, что модули обмениваются данными, представленными скалярными значениями. При небольшом количестве передаваемых параметров этот тип сцепления обеспечивает наилучшие технологические характеристики ПО (самое лучшее, что может быть)
2. Сцепление по образцу – предполагает, что модули обмениваются данными, объединенными в структуры. При изменении структуры данных необходимо переписать модули, которые используют ее, а также прозрачность связи между модулями становится меньше
3. Сцепление по управлению – предполагает, когда один модуль посылает другому некоторый информационный объект, предназначенный для управления внутренней логикой модуля. Таким образом очень часто выполнят настройку режимов работы ПО.
4. Сцепление по общей области данных – предполагает, что модули работают с общей областью данных – на текущий момент такое сцепление считается недопустимым
5. Сцепление по содержимому, когда один модуль содержит обращение к внутренним компонентам другого, например, читает или изменяет внутренние данные, коды, передает управление вовнутрь второго модуля
6. Связанность модулей – мера прочности соединения функциональных и информационных объектов внутри одного модуля. Она характеризует степень взаимосвязи элементов, реализуемых одним модулем. Различают следующие виды связи (даются в порядке убывания уровня):  
   1) Функциональная связанность – когда все объекты модуля предназначены для выполнения одной функции, например, операции, объединяемые для выполнения одной функции или данные, связанные с. одной функцией

2) Последовательная связанность - Когда выход одной функции служит исходными данными для другой. Как правило такой модуль реализует одну подпрограмму, выполняющую две функции. Функциональная или последовательная связанность.

3) Информационная связанность – когда функции внутри модуля обрабатывают одни и те же данные

4) Процедурная связанность – такая связь имеет функции и данные, которые являются данными одного процесса.

5) Временная связанность – подразумевает, что функции выполняются параллельно или в течении определенного периода времени, например при инициализации (запуске) некоторого процесса.

6) Логическая связанность – основывается на объединении функции данных в одну логическую группу, например, функция обработки текстовой информации и т.д.

7) Случайная связанность – связь между элементами была или отсутствует.

# Нисходящая и восходящая разработка ПО

Восходящий подход к ПО – при использовании такого подхода сначала проектируют и реализуют компоненты нижнего уровня, а затем всех последующих и осуществляют их сборку. Исторически такой подход появился раньше.

Нисходящий подход к программированию – проектирование и последующая реализация выполняется сверху вниз, нереализованные компоненты заглушаются.

Защитное программирование (защита от ошибок) – при таком программировании необходимо проверять прежде всего:

1. Правильность операций ввода/вывода
2. Допустимость промежуточных результатов (типы, индексы, переменные и т. д.)

Принято различать следующие виды ошибок:

1. Ошибки передачи (данные искажаются аппаратными средствами)
2. Ошибки преобразования – программа неверно преобразует исходные данные во внутренний формат.
3. Ошибки перезаписи – пользователи вводят лишний или другой символ
4. Ошибки данных – пользователь вводит неверные данные

Проверка допустимости промежуточных результатов – проверяем только те промежуточные результаты, проверка которых целесообразна и не особо сложна:

* Проверка правильности вычислений индекса массива
* Значения переменных, которые определяют управляющее значение во внутренних циклах
* Предотвращение накопления погрешностей, для предотвращения необходимо:  
  1) Необходимо избегать вычитания близких чисел  
  2) Избегать деления больших чисел на малые  
  3) Сложение длинной последовательности чисел  
  4) Не использовать условие равенства вещественных чисел

Сквозной структурный контроль – реализация:

Для всех этапов генерируются списки наиболее часто встречающихся ошибок, которые формируют из опыта предыдущих разработок и различных источников информации.

Эффективными считаются программы, требующие минимального времени выполнения и минимального объема оперативной памяти.

Способы уменьшения времени выполнения:

1. Избегать использования преобразования переменных и типов
2. Избегать длинных операций умножения и деления, заменяя их сдвигами: сложением и вычитанием
3. Оптимизировать запись условных выражений, исключая лишние проверки
4. Выносить вычисления константных, т.е. не зависящих от цикла за цикл выражений (ну ты понял, да)
5. Исключать многократные обращения к элементам массивов по индексам.

Способы сэкономить память:

1. Выбирать алгоритмы обработки, не требующие дублирования исходных данных структурных типов в процессе обработки (Класс String)
2. Следует помнить, что при передаче структурных данных по значению, копии этих данных различаются в оперативной памяти, при передаче по ссылке такого нет, так что используем ее.

Обеспечение эффективности не должно требовать серьезных трудовых и временных затрат, а также приводить к существенному ухудшению технологичности.

# Разработка лицензионного соглашения

Программные продукты считаются объектами интеллектуальной собственности. Все отношения, связанные с правами на них, регулируются нормами авторского права. Программам предоставляется правовая охрана, как произведениям в литературе. Авторское право возникает в силу создания программного продукта для его признания и осуществления не требуется регистрации, сертификации и т. д. Авторское право не связано с правом собственности на материальный носитель информации, на котором записан продукт. Любая передача прав на носитель не влечет за собой передачи прав на продукт.

При создании обучающих программ, группы владельцев авторских прав включаются: авторы учебного материала, часть специалистов по реализации этой программы, состав данной группы ограничивает важные условия.

Авторское право действует в отношении программ, являющиеся результатами творческой деятельности автора. Таким образом, владельцами авторских прав не могут быть лица, решавшие в рамках проекта чисто технические задачи, например – набиравшие текст. Все члены коллектива разработчиков считаются авторами программного продукта.

Если компоненты программы обладают самостоятельным назначением, то каждый из авторов располагает правами на разработанную им часть.

Права интеллектуальной собственности на программный продукт подразделяются на личные неимущественные и имущественные.

1. Личные неимущественные права: право авторства - право считаться автором, право на имя – определять форму указания своего имени, право на неприкосновенность – защита самой программы и/или ее названия от всякого рода искажений, способных нанести ущерб чести и достоинству автора. Личные неимущественные права принадлежат авторам независимо от их имущественных прав. Не могут передаваться другим лицам и охраняются бессрочно.
2. Имущественные права – связаны с владением, пользованием и распоряжением имущества. Исключительные имущественные права – означают полномочия осуществлять или разрешать осуществление следующих действий, выпуск программы – полное или частичное копирование программы. Распространение программы - предоставление к ней доступа бесплатно или за плату, модификация программы, включая перевод с одного языка на другой, иное использование программы. Исключительными имущественными правами могут владеть: авторы, т.е. создатели продукта, даже если они разработали его за счет грантов и спонсоров, автор совместно с заказчиком и только заказчик.

Распределение имущественных прав в рамках коллектива авторов, а также между авторами и заказчиками отражается в договорах заинтересованных сторон. Если в договоре между заказчиками и авторами не предусмотрено иное, то все исключительные имущественные права на программу созданы по заданию работодателю, принадлежат последнему. Имущественные права на программный продукт переходят по наследству. Они могут быть переданы полностью или частично. Срок их действия ограничен 50 годами после смерти последнего из авторов. По истечению этого срока программа становится общественным достоянием. Специфика распространения ПО состоит в том, что потребителю продается не сам продукт, а право на его применение в определенных условиях. Приобретаемое право относится к категории неисключительных имущественных прав. Договор о предоставлении права на использование объекта интеллектуальной собственности называется лицензионным соглашением. Его сторонами являются правообладатель и конечный пользователь.

В лицензионном соглашении необходимо четко определить права, предоставляемые владельцу лицензии и условия их реализации, а также запретив действия, выходящие за их границы.

К условиям реализации прав относится срок действия лицензии, кол-во установок продукта, возможность создания резервной копии дистрибутива и т.д. Все права интеллектуальной собственности сохраняются за правообладателем. В лицензионном соглашении обычно фиксируются меры по сопровождению ПО, а также состав и порядок оказания дополнительных услуг. Также целесообразно включить положение, снимающее с правообладателя ответственность за ущерб и убытки любого вида, связанные с использованием или невозможностью использования продукта. За невыполнение своих обязательств третьим лицам (например, фирма-распространитель, почта, провайдер), а также нарушение соглашения в результате форс-мажорных обстоятельств. Также необходимо добавить пункт о том, что оно может быть изменено только в другом договоре, подписанным обеими сторонами.

# Создание демонстрационной версии лицензионного соглашения – дома

# Работа с файлами в C#

String filePatch = @”C:\Temp\myFile.txt”;

Fileinfo file = new FileInfo(filePatch);

Console.Writeline(@“Hello

My

World!”);

Directory и Directory\_info – очень удобно просматривать содержимое каталогов, т.е. проходить по всем файлам и т. д.

Class Path – класс, который работает со строкой “путь к файлу”

GetDirectoryName()

GetExtension()

GetFileName()

GetFileNameWithoutExtension()  
GetRandomFileName()

# Чтение и запись файлов c использованием потоков

Используем класс Stream для работы

Класс BinaryWriter

Byte [] dataCollection = {1, 4, 6, 7, 12, 33, 26, 98, 82, 101 };

FileStream destFille = new FileStream(destinationFilePatch);

BinaryWriter writer = new BinaryWriter(destination);

# Юнит-тесты

int Calculate (int operandOne, int operandTwo)  
{

int result = 0;

//Perform some calculation

return result;

}

[TestMethod()]

Public void CalculateTest()

{

Program target = new Program();

int operandOne = 0;

int operandTwo = 0;

int expected = 0;

int actual;

actual = target.Calculate(operandOne, operandTwo);

Assert.AreEqual(expected, actual);

…..

}

Если тестируемый метод статистический

Actual = Program\_Accessor.GenerateRandomNumbers(min, max, numberOfRequirednumbers);

# Пользовательский тип данных

1. Перечисление

Еnum Days {Monday, Tuesday}

Enum Season {Spring, Summer, Fall, Autumn = Fall, Winter};

Или можно и так:  
 enum Season : sbyte {Spring = -3, Summer, Fall}

2. Структуры  
 int x = 99;

String xAsString = x.Tostring();

Класс object – это класс, от которого наследуются все остальные классы.

Структуры невозможно наследовать, они могут содержать конструкторы.

Возможно использования объединения (union)

int? d; - пример типа, допускающего значение null + int;

bool? K; - пример типа, допускающего значение null + bool;

int z = d ?? -1;

# Классы

Public Residence(): this(ResidenceType.House, 3, true, true){

}

….

# Инкапсуляция

По возможности закрывать все поля и методы. Для этого в C# есть определенные механизмы. В C# есть 5 спецификатора доступа:

* Public
* Private
* Protected
* Internal
* Protected internal

# Статические поля, методы и классы

Статические поля по умолчанию инициализируются нулями

Class MyClass{

Int a=g;

Double b=5;

String s = “My string”;

Static int index; //располагается отдельно в области глобальных данных

}

Статический конструктор существует отдельно, предназначен для инициализации статических полей

Public static int b;

Public int c;

Static Test(){

If(Console.Read()==0) b=30;

b=20;

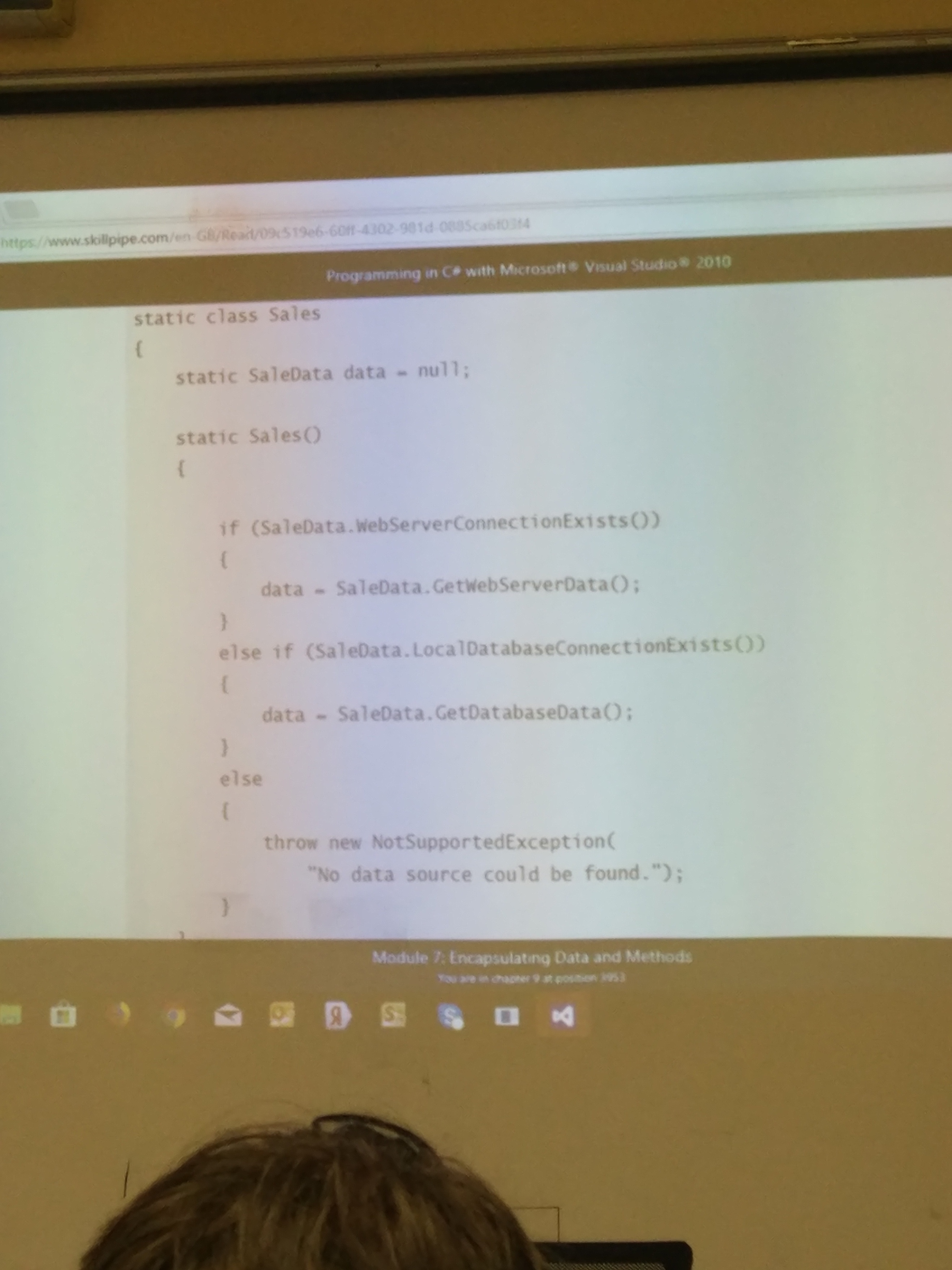
}

Public Test(){

b = 0;

c = 0;

}

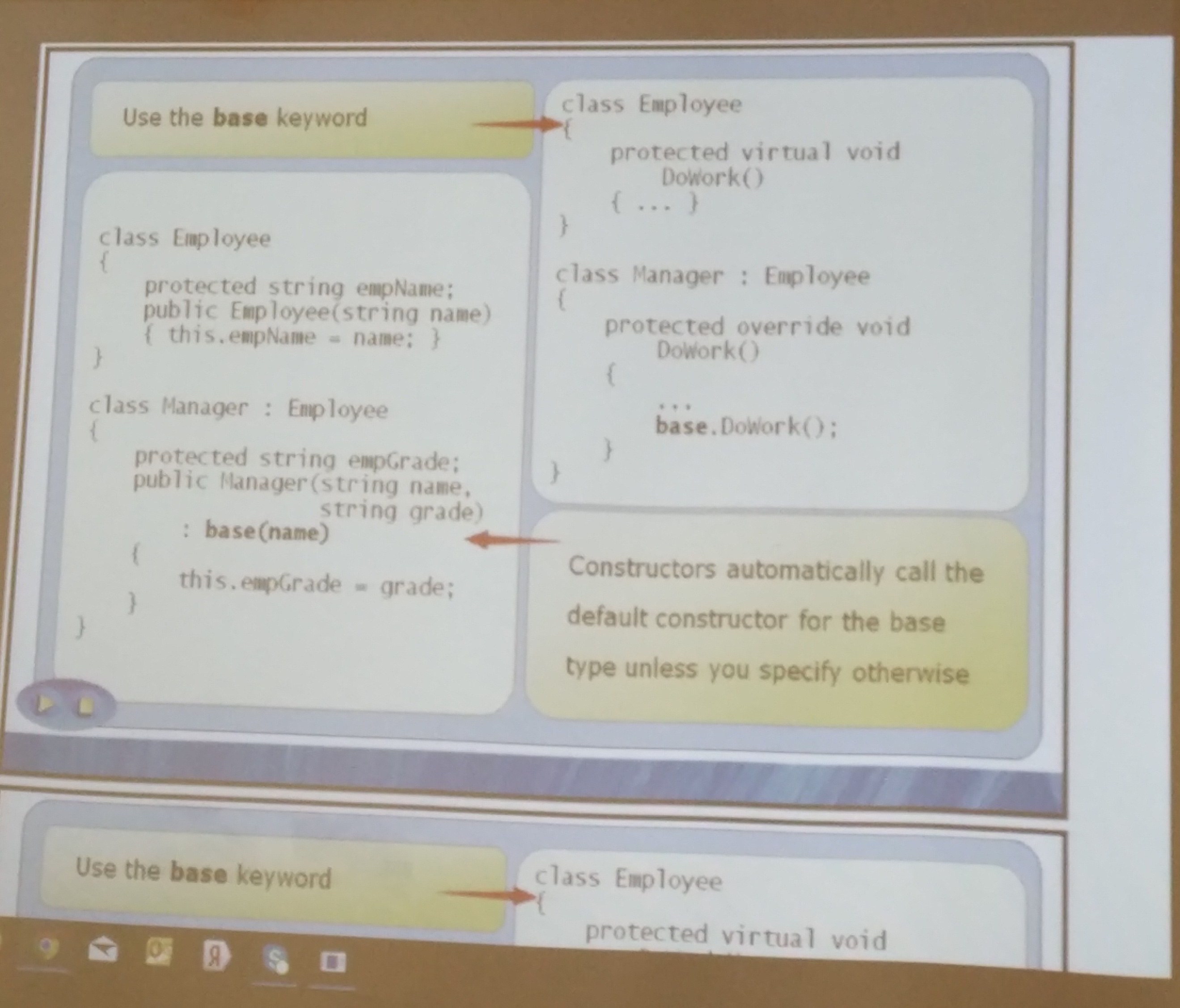
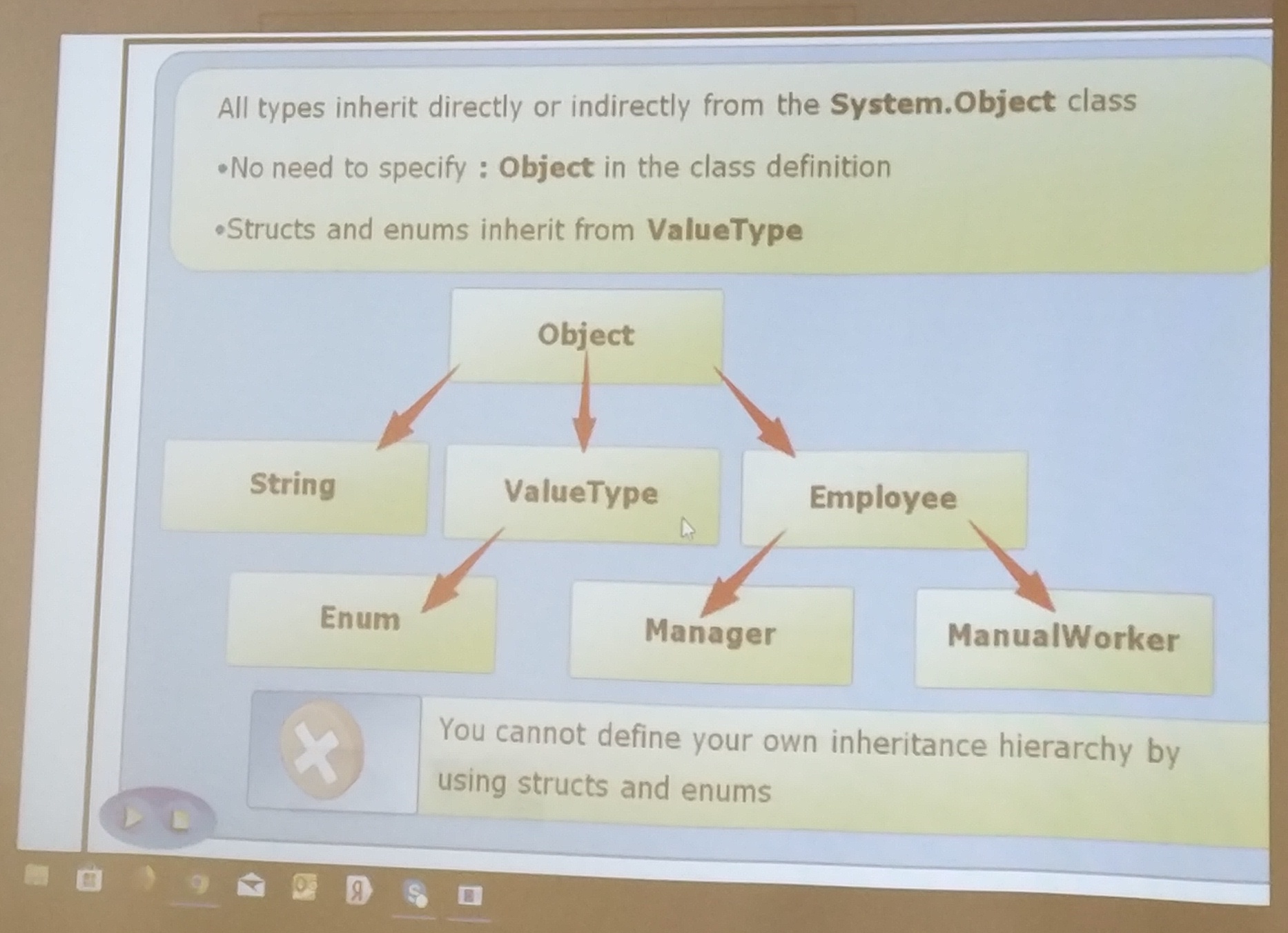


С# позволяет сделать просто все что угодно статическим. В статическом классе поля и методы тоже должны быть static

В C# есть возможность написания Extension-методов

# Наследование

В C# нет и не будет множественного наследования



# Полиморфизм

Пример механизма виртуальнвых функций

Class Employee

{

Protected virtual void

DoWork()  
{……}

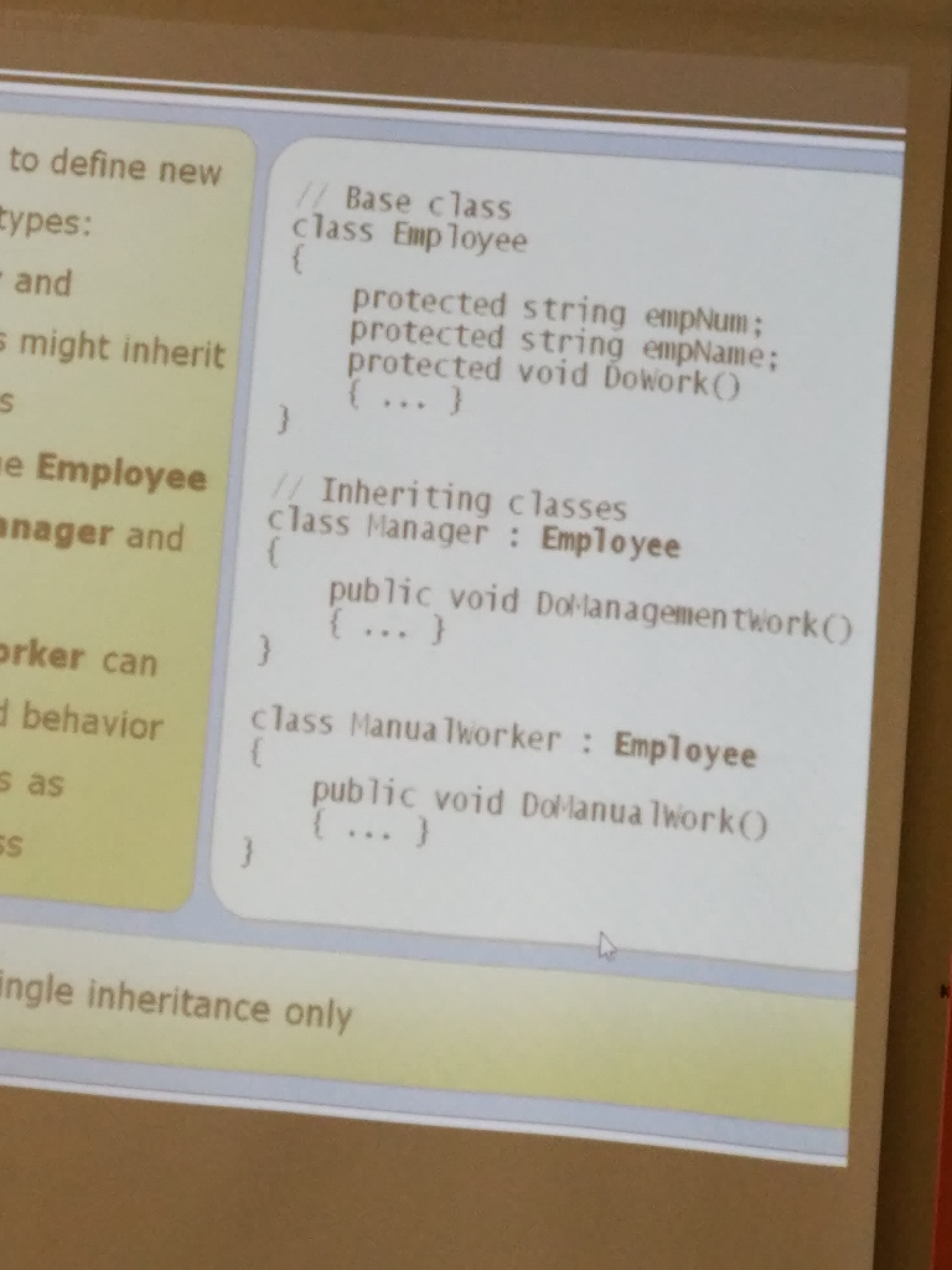
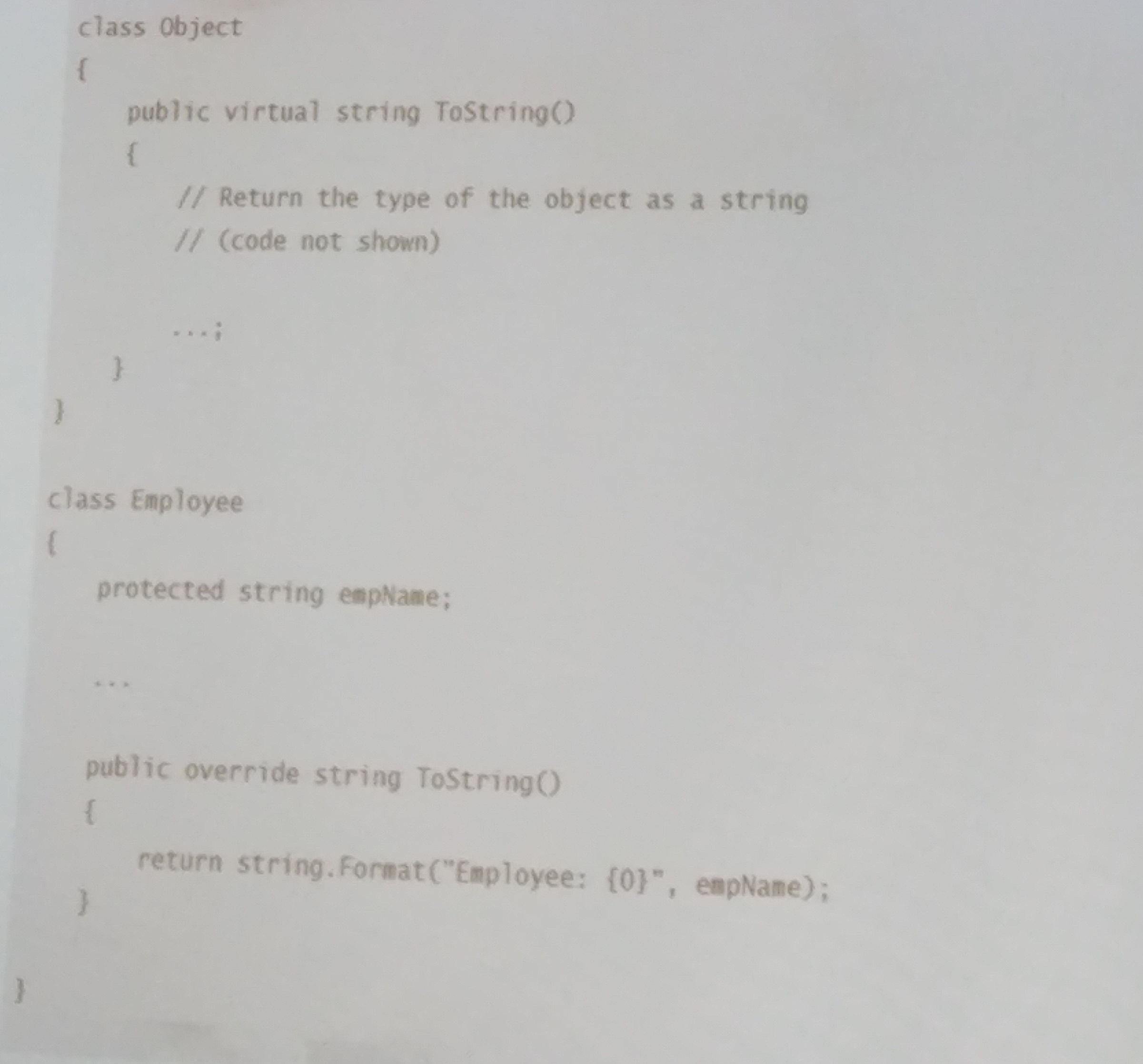
}

Class manager : Employee  
{

Protected override void

DoWork()  
 {……}

}



# Механизм скрытых методов

Manager myManager = new Manager(“Fred”,”VP”);

Employee myEmployee = myManager;

Manager myManagerAgain = myEmployee as Manager;

manualWorker myWorker = new ManualWorker(“Bert”);

myEmployee = myWorker;

myManagerAgain = myEmployee is a ManualWorker;

# Sealed классы и Sealed методы.

Sealed class Manager : Employee

{

………..  
}

# Интерфейсы

Пример:

Public interface ICalculator  
{  
 double Add();

Double Subtract():

Double Multipy();

Double DEV();  
}

В начале идет класс, от которого наследуется все, а затем методы:  
  
class Calculator: object, ICalculator, IMemory

{

………………..

}

Calculator myCalculator = new Calculator();

ICalculator iMyCalculator = myCalculastor;

В C# возможно указать данные на его родителя посредством base, например base.Dowork

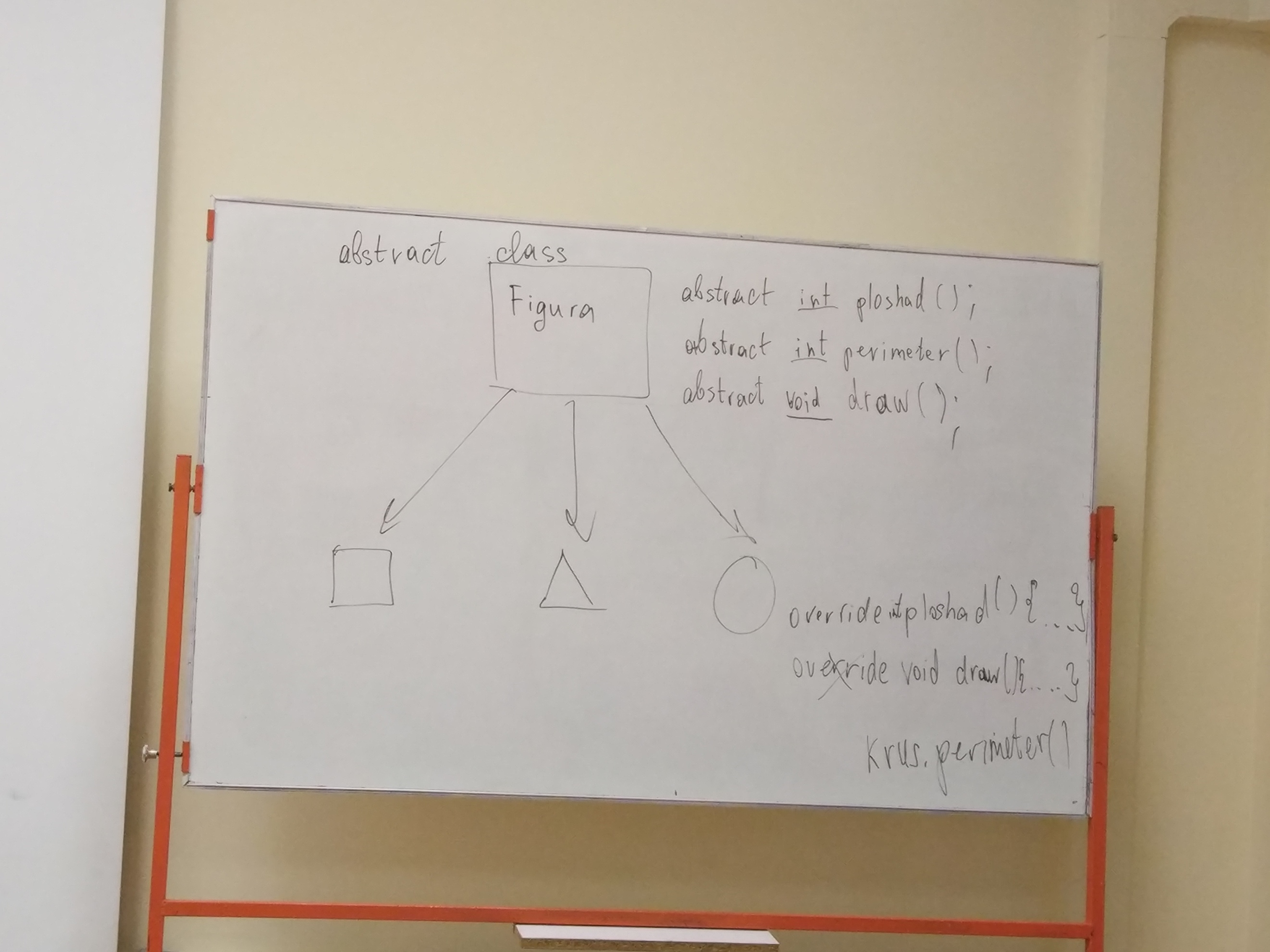
Base.DoWork() – можно

Base.base.DoWork() – нельзя

Передача параметров конструктора родительского класса.

Sealed class Manager : Employee – запрет на наследование

# Абстрактный класс



В абстрактном классе можно создавать абстрактные методы, код которых не пишется. Таким образом, мы должны определить функционал всех этих методов в наследниках. Причем это мы можем сделать без слова override

Создать объект абстрактного класса невозможно, только классы объектов-наследников