Tasarım Şablonları 2. Hafta

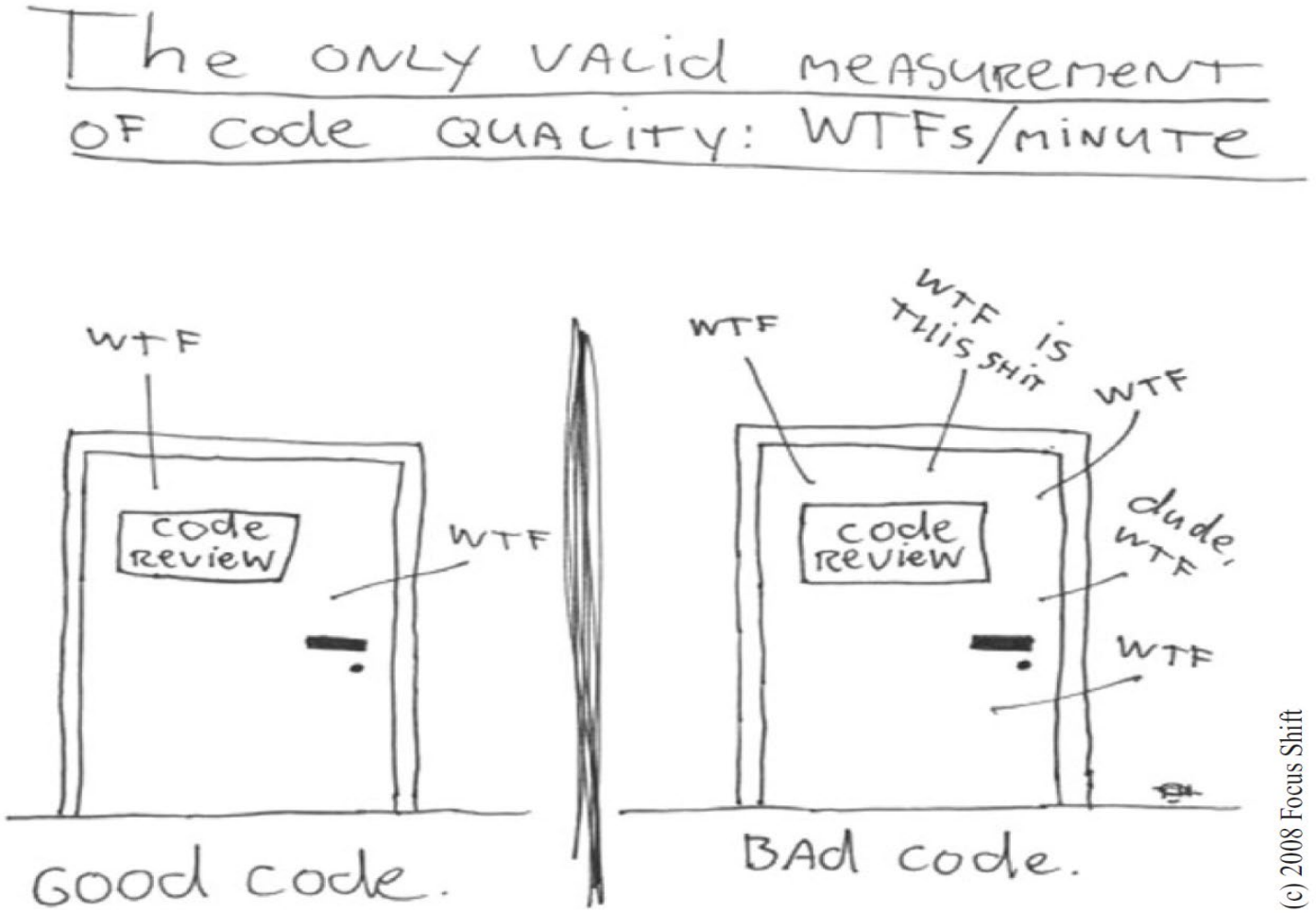
03.03.2025

Clean Code İlkeleri –Refactoring ve Şablonlara Giriş

Code Quality Measurement: WTFs/Minute

WTFs per minute means the **number of times** anyone says "What the heck?" when s(he) is trying to understand someone else's code.

- **WTF**: "Well actually why didn't you do it this way"
- Total Productive Maintenance (TPM) (1951)
- **Lean:** Focus on **maintenance** rather than on production
- 70 years ago making **readable** code is as important as making it executable.



Reproduced with the kind permission of Thom Holwerda.
http://www.osnews.com/story/19266/WTFs_m

Bad Code & Good Code

- Bad code tries to do too much, it has muddled intent and ambiguity of purpose.
- Clean code is focused.
- Each function, each class, each module exposes a **single-minded attitude** that remains entirely undistracted, and unpolluted, by the surrounding details.

One of the 5s of Lean: Seiso

- Seiso, or cleaning ("shine" in English): Keep the workplace free of hanging wires, grease, scraps, and waste (Çalışma ortamını sarkan tellerden, yağdan, hurdalardan ve atıklardan uzak tutun.

- Reviews on code: Get rid of commented-out code lines that capture history or wishes for the future?
- Importance of Names
  - Names play a large factor in coding.
    - ✓ They help us understand code and maintain it.
  - Writing clean code is not an easy task.
    - ✓ It takes discipline, practice, and implementation.
  - If we keep at it we'll easily **limit the amount of WTFs** we get in our code reviews.

The Importance of Refactoring

- **Bjarne Stroustrup**, inventor of C++ (first released 1985 and initially standardized in 1998 as ISO/IEC 14882:1998), author of The C++ Programming Language (third edition: 2024) https://www.stroustrup.com/programming.html

«*I like my code to be **elegant** and **efficient**. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease **maintenance**, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well».*

The Importance of Refactoring

- **Grady Booch**, one of author of Object Oriented Analysis and Design with Applications (third edition 2007)

«*Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control*»

- "Big" **Dave Thomas,** founder of OTI (Object Technology International (1988 after 1996 by IBM developed Visual Age, Smalltalk and Java tools) , also godfather of the Eclipse strategy , and Integrated Development Environment (IDE)

«Clean code can be read, and enhanced by a developer other than its original author. It has ***unit*** *and* ***acceptance*** *tests. It has meaningful names. It provides one way rather than many ways for doing one thing. It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API. Code should be literate since depending on the language, not all necessary information can be expressed clearly in code alone».*

The Importance of Refactoring

- **Michael Feathers,** author of Working Effectively with Legacy Code (2004)

  «*I could list all of the **qualities** that I notice in clean code, but there is one overarching quality that leads to all of them. Clean code always looks like it was written by someone who **cares.** There is nothing obvious that you can do to make it better. All of those things were thought about by the code's author, and if you try to imagine improvements, you're led back to where you are, sitting in appreciation of the code someone left for you—code left by someone who cares deeply about the craft».*

- **Ron Jeffries,** author of Extreme Programming(1996), Extreme Programming Adventures in C# (2004) « ……. **simple code**: runs all the tests; contains no duplication; expresses all the design ideas; minimizes the number of entities such as classes, methods, functions. **When the same thing is done over and over, it is not well represented in the code**. ……..

Refactoring: Extract method

- In poorly maintained code bases, methods tend to grow larger.
- People add logic to existing methods, and they just continue to grow.
- As this happens, methods can end up doing two or three different distinct things for their callers.
- In pathological cases, they can end up doing tens or hundreds.
- Extract Method is the remedy in these cases.
- Refactoring is the process of restructuring code without changing its behavior and the technique "Extract Method" is one of the most important building blocks of refactoring.
- If the same code is found in two or more methods in the same class: use Extract Method and place calls for the new method in both places.

Steps to create an Extract Method from the Feather's book Working Effectively with Legacy Code

The first thing to do is a set of tests.

If tests have done thoroughly exercising a method, it is possible to extract methods from it using these steps:

1. Identify the code to be extracted, and comment it out.

2. Think of a name for the new method and create it as an empty method.

3. Place a call to the new method in the old method.

4. Copy the code to be extracted into the new method

5. *Lean On the Compiler* to find out what parameters have to be passed and what values have to be returned

6. Adjust the method declaration to accommodate the parameters and return value (if any).

7. Run your tests.

8. Delete the commented-out code.

Java Code Example: from the book of Feather's

```
public class Reservation
{
public int calculateHandlingFee(int amount) {
int result = 0;
if (amount < 100) {
result += getBaseFee(amount);
}
else {
result += (amount * PREMIUM_RATE_ADJ) + SURCHARGE;
}
return result;
}
...
}
```

- The logic in the **else-statement** calculates the handling fee for premium reservations.
- It will be useful to use that logic someplace else in the system.
- Instead of *duplicating the code*, it is possible *to extract it* from here and then use it in *the other place*.

The First and Second Steps

```
public class Reservation
{
public int calculateHandlingFee(int amount) {
int result = 0;
if (amount < 100) {
result += getBaseFee(amount);
}
else {
// result += (amount * PREMIUM_RATE_ADJ) + SURCHARGE;
}
return result;
}
...
}
```

```
public class Reservation
{
public int calculateHandlingFee(int amount) {
int result = 0;
if (amount < 100) {
result += getBaseFee(amount);
}
else {
// result += (amount * PREMIUM_RATE_ADJ) +
                                        SURCHARGE;
result += getPremiumFee();
}
return result;
}
int getPremiumFee() {
}
...
}
```

Copy the Current Code into the New Method

```
public class Reservation   {
public int calculateHandlingFee(int amount) {
int result = 0;
if (amount < 100) {
result += getBaseFee(amount);
}       else {
// result += (amount * PREMIUM_RATE_ADJ) +
                                    SURCHARGE;
result += getPremiumFee();       }
return result;           }
int getPremiumFee() {
result += (amount * PREMIUM_RATE_ADJ) +
                                SURCHARGE;       }
 ...
```

- ❑ The code cannot be compiled.
- ❑ The code uses variables named **result** and **amount** that aren't declared.
  - ❖ Because only a portion of the result will be computed , it is required to return
- ❑ What have been computed.
  - ❖ It is required to get hold of the amount,
    - ✓ **amount** may be a parameter to the method and add it to the call

```
public class Reservation  {
public int calculateHandlingFee(int amount) {
int result = 0;
if (amount < 100) {
result += getBaseFee(amount);
                        }    else {
// result += (amount * PREMIUM_RATE_ADJ) + SURCHARGE;
result += getPremiumFee(amount);
}
return result;
}
int getPremiumFee(int amount) {
return (amount * PREMIUM_RATE_ADJ) + SURCHARGE;
}
...
```

} **Finally it is required to test the refactored code defined with an extract method**

A core technique for Working with Legacy Code

```
public class Reservation  {
public int calculateHandlingFee(int amount) {
int result = 0;
if (amount < 100) {
result += getBaseFee(amount);
}   else {
result += getPremiumFee(amount);
}
return result;
}
int getPremiumFee(int amount) {
return (amount * PREMIUM_RATE_ADJ) + SURCHARGE;
}
...             }
```

It is also possible to
extract duplication,
separate responsibilities
break down long methods
defining an extract method.

Lean on the Compiler

- The primary purpose of a compiler is to translate source code into some other form, but in **statically typed languages**, you can do much more with a compiler.
- we can take advantage of its ***type checking*** and use it to identify changes we need to make.
- This practice is called **leaning on the compiler.**

Following is an example of how to do, in other words lean on the compiler;

```
double domestic_exchange_rate;
double foreign_exchange_rate;
```

Lean on the Compiler

- A set of methods in the same file uses these variables
- We want to find someway to change them under test so we use the **Encapsulate Global References** technique.
- We write a class around the two declarations (variable names) and declare a variable of that class.

```
class Exchange
{
public:
double domestic_exchange_rate;
double foreign_exchange_rate;
};
Exchange exchange;
```

Lean on the Compiler

- Compiler can't find the followings while compiling:

domestic_exchange_rate and *foreign_exchange_rate*

- They are changed, they are accessed off the ***exchange*** object.

The code line,

*total = domestic_exchange_rate * instrument_shares;*

becomes

*total = **exchange.**domestic_exchange_rate * instrument_shares;*

- Bu gösterim tekniği **compiler guide** olarak adlandırılır vebu tür değişikliklerde oldukça etkindir.
- Değişken bildirimindeki bu değişiklik geliştiricinin artık değişimlerle ilgili karar almayacağı anlamına kesinlikle gelmez.
  - Kodu yazanın sadece bazı durumlarda bazı temel işleri derleyiciye yaptırması anlamına gelmektedir.

Two works of Lean on the Compiler

1. Altering a declaration to cause compile errors
2. Navigating to those errors and making changes

- It is possible to lean on the compiler to make **structural changes** to the program, as done in the **Encapsulate Global References** example.
  It is also possible to use it to initiate type changes.
- One common case is changing the *type of a variable declaration* from a *class* to *an interface*
  - using the errors to determine which methods need to be on the interface.

Encapsulate Global References (A Refactoring Style)

```
bool AGG230_activeframe[AGG230_SIZE];
bool AGG230_suspendedframe[AGG230_SIZE];
void AGGController::suspend_frame()   { suspend_frame method needs to access the active and suspended frames
frame_copy  (AGG230_suspendedframe,  AGG230_activeframe)make the frames, members of the AGGController class
   clear(AGG230_activeframe);
   flush_frame_buffers();                    }
void AGGController::flush_frame_buffers()
{          for (int n = 0; n < AGG230_SIZE; ++n) {     aynı sınıfta farklı bir metot da aynı global değişkenleri kullanıyor
               AGG230_activeframe[n] = false;
              AGG230_suspendedframe[n] = false;      }          }
```

the code here has two global arrays.

Fakat farklı sınıflar da bu çerçeveleri, böylece değişkenleri kullanabilir O zaman ne yapılabilir? i) global değişkenler metoda , **suspend_frame(),** parametre olarak geçebilir. Global değişkenli bu fonksiyonu çağıran (call) diğer fonksiyonlara da parametreleri geçmelidir.

ii)Global değişkenler (iki dizi), **AGGController** sınıfının yapıcı fonksiyonunun argümanları olabilir.

Sonuç olarak: farklı global değişkenler aynı sınıfın içerisinde yoğun olarak farklı yerlerde kullanılarak sürekli değişiyor ve farklı metotların değerlerini değiştiriyor.

Introduction to Design Patterns

Christoper Alexander and Design Patterns

- The first idea of using patterns was for building and proposed by the architect *Christopher Alexander.*
  - He found recurring themes in architecture, and captured them into descriptions
    - ✓He called them patterns.
- The term 'pattern' appeals to the replicated similarity in a design
  - The similarity makes room for variability and customization in each of the elements

- Alexander defines: «Each pattern is a three part rule which express a relation between a certain context, a problem and a solution.
- Each pattern is a relationship between

  a certain context,

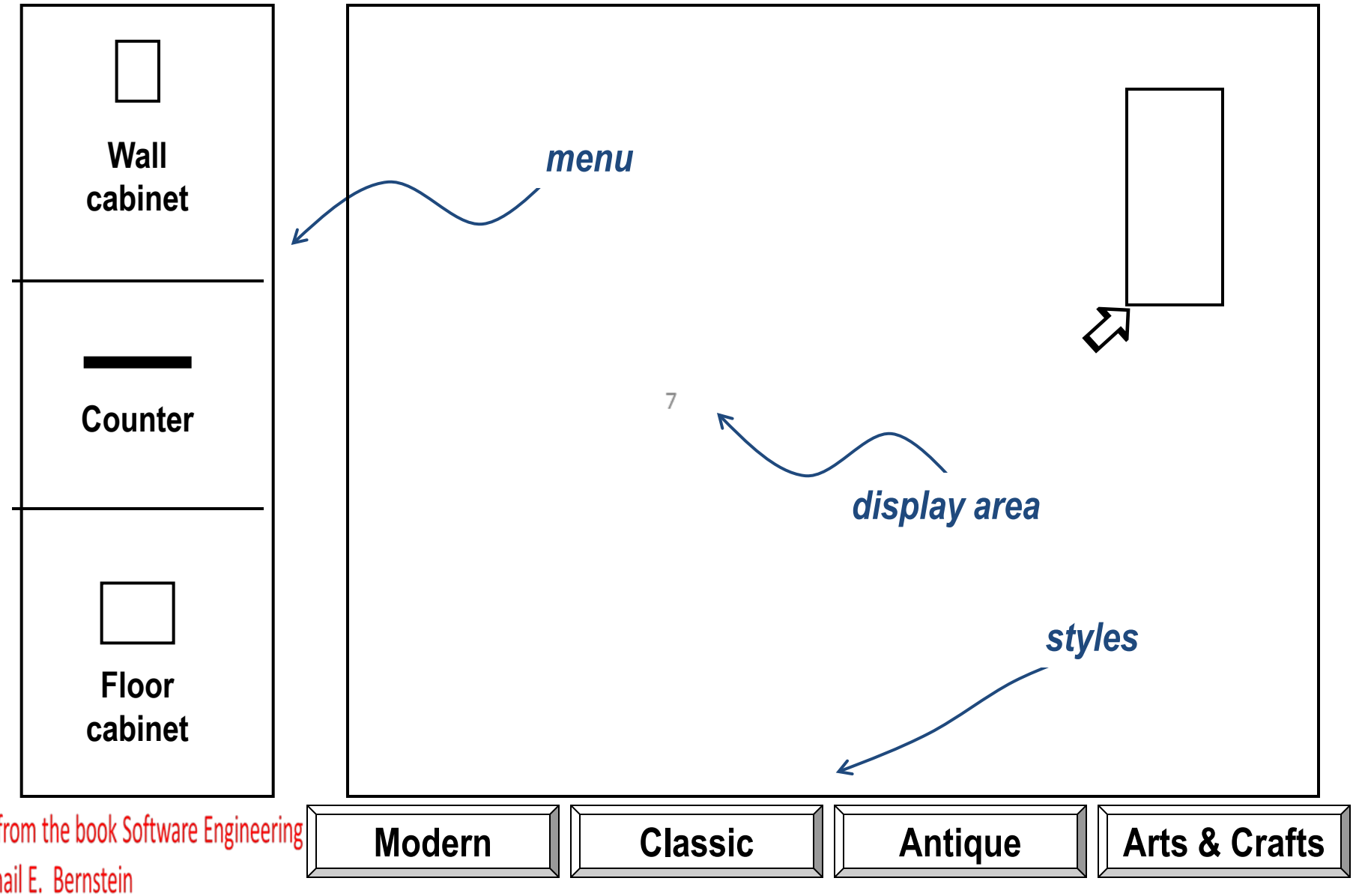
  a certain system of forces which occurs repeatedly in that context

  a certain spatial configuration which allows these forces to resolve themselves.

- A pattern is an instruction and shows how this configuration can be used over and over again.

- The pattern is a thing that happens in the world
- The rule which tell us how to create that thing and when we must create it.

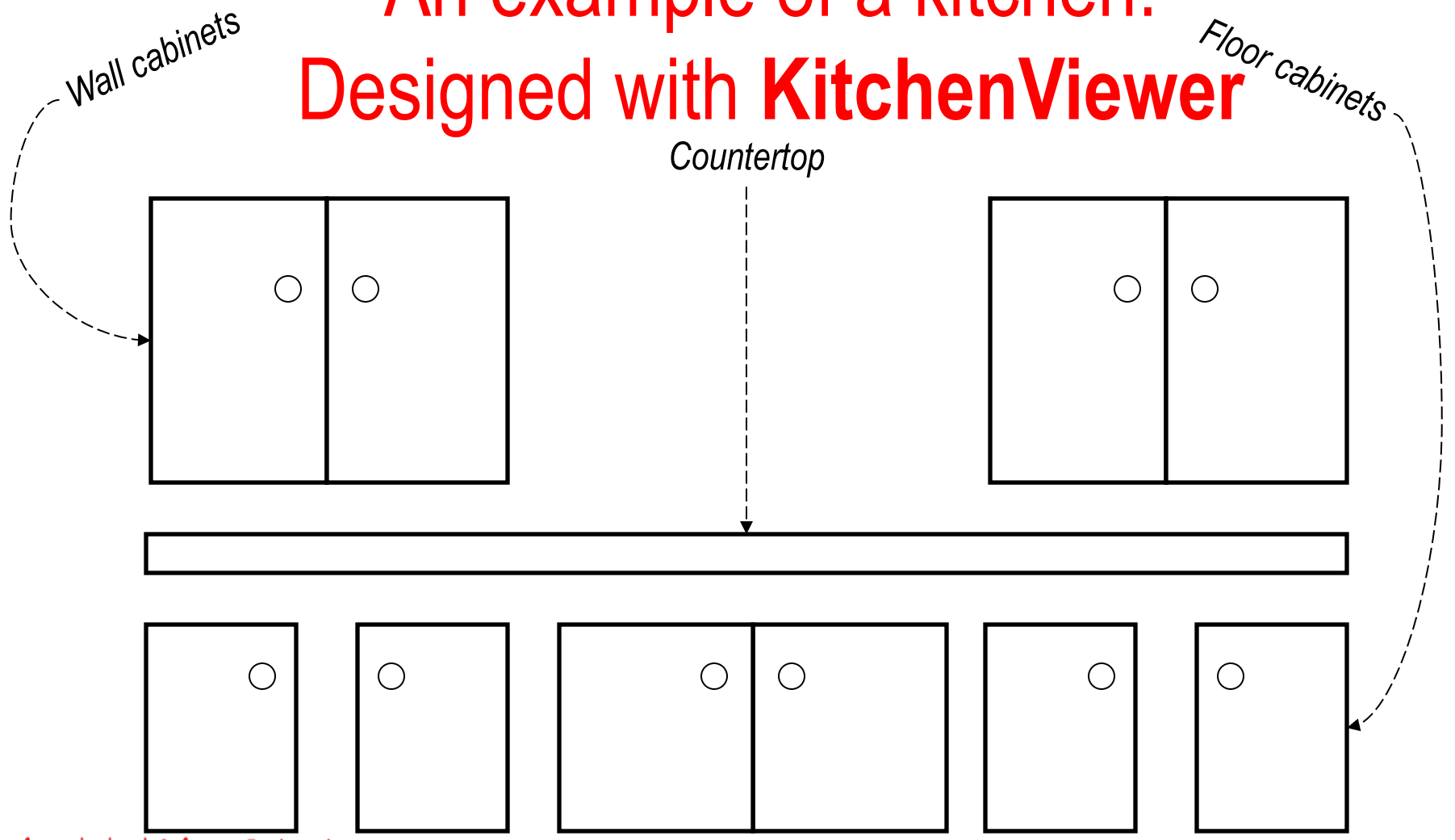
Kitchen Viewer *(problem is called as KitchenViewer)* Interface:



Kitcher Viewer example is taken from the book Software Engineering written Eric J. Braude and Michail E. Bernstein

An architectural pattern example

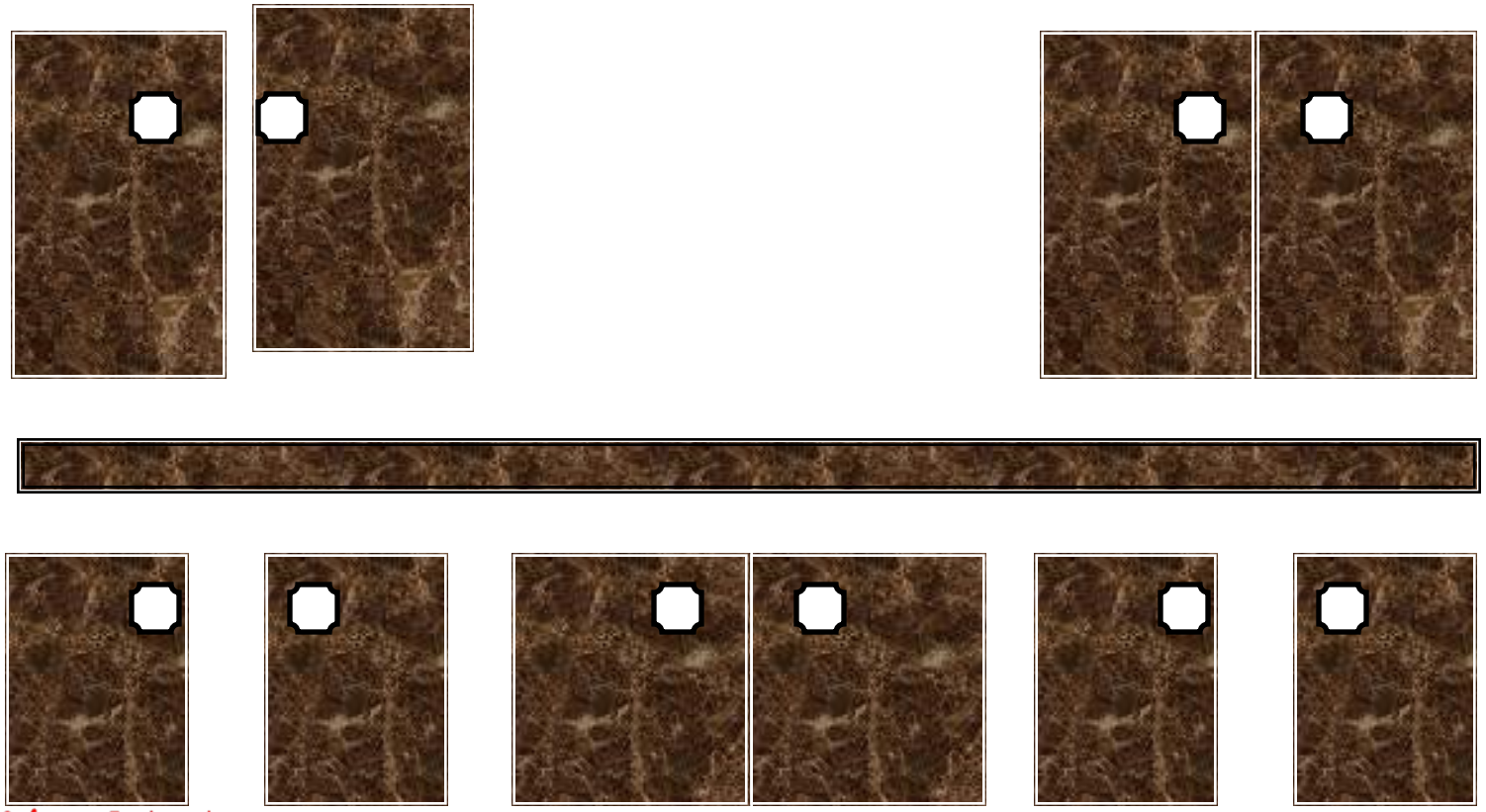
An example of a kitchen: Designed with **KitchenViewer**



Kitcher Viewer example is taken from the book Software Engineering written Eric J. Braude and Michail E. Bernstein

Modern	Classic	Antique	Arts & Crafts

Selecting *Antique* Style using KitcherViewer



Kitcher Viewer example is taken from the book Software Engineering written Eric J. Braude and Michail E. Bernstein

Modern | Classic | **Antique** | Arts & Crafts

Specific Design Purposes for KitcherViewer

- Farklı stillerde tasarlanan dolaplar aslında aynı süreçleri içeriyorlar.Bu nedenle:

Counter counter =new Counter();

draw (counters);

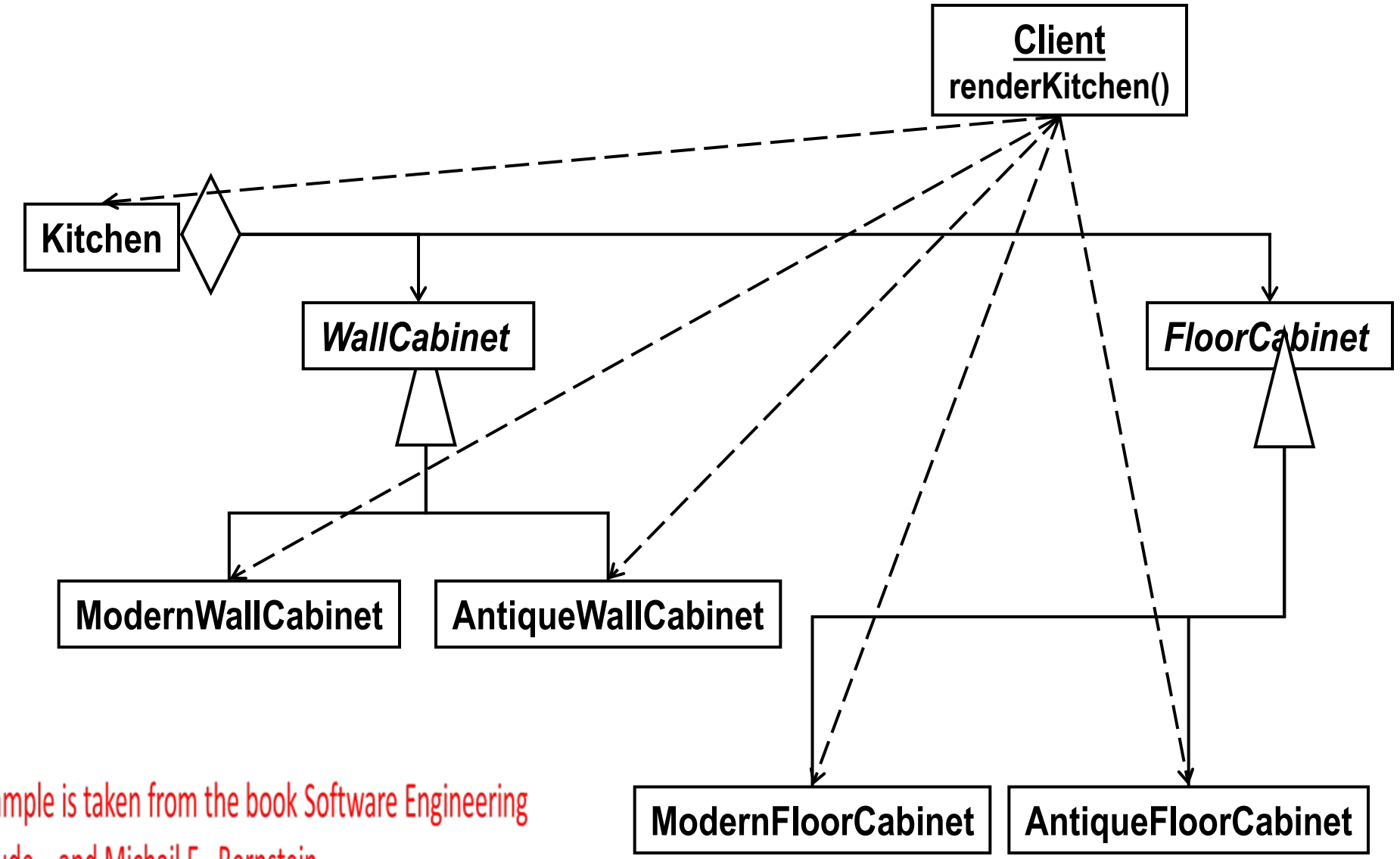
yazılacaktır.

- Tek bir kod bloğu, bağlama göre farklı olası yapılardan birini çalıştıracaktır → *Polymorphism*

Böylece:

- An application must construct a family of objects at runtime.
- The design must enable choice among several families of styles

KitchenViewer Without Design Patterns



Kitcher Viewer example is taken from the book Software Engineering written Eric J. Braude and Michail E. Bernstein

An Example: Without Applying a Design Pattern

- Örnek Uygulama *Without applying a Design Pattern*
  - renderKitchen() metodu Kitchen, WallCAbinet ve Floor Cabinet sınıflarını kullanır; Client sınıfının bir fonksiyonudur.
    - ✓Bu kod her dolap stili için tekrarlanacaktır (repeated)
    - ✓Bu kod farklı dolap yerleşimleri için farklı olacağı için duplicated olarak kabul edilir.

- Repeated code for every style results in more prone-error, and far less maintainable code and complicated .
- It is inflexible, hard to prove correct, and hard to reuse

The method renderKitchen() in KitchenViewer without applying Design Patterns

```
// Assume that the antique style was selected .
// Create the antique  wall cabinets
AntiqueWallCabinet    antiqueWallCabinet1 = new  AntiqueWallCabinet () ;
AntiqueWallCabinet    antiqueWallCabinet2 = new  AntiqueWallCabinet () ;
// Create the antique floor cabinets
AntiqueFloorCabinet ant iqueFloorCabinet1 = new AntiqueFloorCabinet () ;
AntiqueFl.oorCabinet antiqueFloorCabinet2 = new AntiqueFloorCabinet () ;
// Create the kitchen object, assuming the existence of add() methods
Kitchen antiqueKitchen = new Kitchen();
antiqueKitchen.add( antiqueWallCabinetl, ... ); // rest of  parameters specify location
antiqueKitchen.add ( antiqueWallCabinet2, ... );
antiqueKitchen.add ( antiqueFloorCabinetl, ... ) ;
antiqueKitchen. add ( antiqueFloorCabinet2, ... ) ;
// Render antiqueKitchen
```

Applying a Design Pattern

KitchenViewer design problem is implemented by applying Abstract Factory design pattern.

- The object will have responsibility for ***creating the kitchen*** .
- Doğrudan *AntiqueWallCabinet nesnelerinin oluşturulması yerine*, renderKitchen() fonksiyonu parametreli olarak uyarlanır ve

new AntiqueWallCabinet ( ) //applies only to antique style:

myStyle. getwallCabinet (); //applies to the style chosen at run time.

yazılır.

- myStyle *run time* sırasında getWallCabinet() ya da getFloorCabinet() metodunu çalıştırır.
  - Bunu gerçekleştirmek için **KitchenStyle** isimli yeni bir sınıf tanımlanır. Bu yeni sınıf ilgili fonksiyonları desteklemek üzere oluşturulmuştur.

New Class KitchenStyle

- KitchenStyle class has subclasses as ModernKStyle, AntiqueKStyle and other types
- Each subclass support separate implementation of getWallCabinet(), getFloorCabinet() and other functions
- Due to polymorphism, executing myStyle.getFloorcabinet()has different effects when **myStyle** is an object of **ModernKStyle** versus an object of **AntiqueKStyle**
- **Client code** references **Kitchen, KitchenStyle, WallCabinet** and **FloorCabinet,** but does not appear in the client code.
- For example, the class AntiqueWallCabinet does not appear in client code.

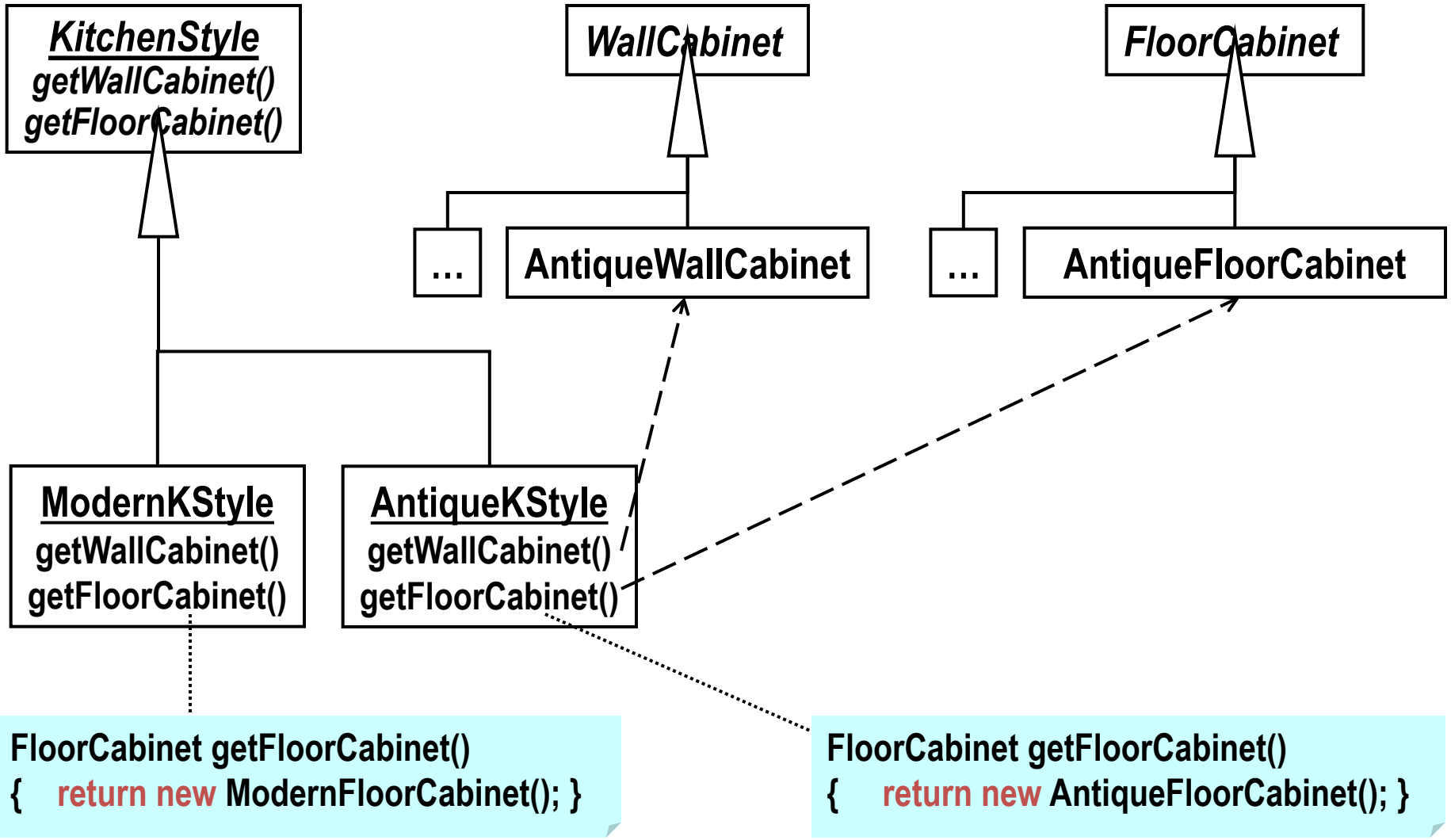
New Class KitchenStyle

- At runtime myStyle is an object of the class ModernStyle
- When the method renderKitchen() executes the following statement:

```
WallCabinet wallCabinet7 = myStyle.getWallCabinet();
```

- The method getWallCabinet() is the version defined in the class ModernStyle, and it returns a ModernWall object.

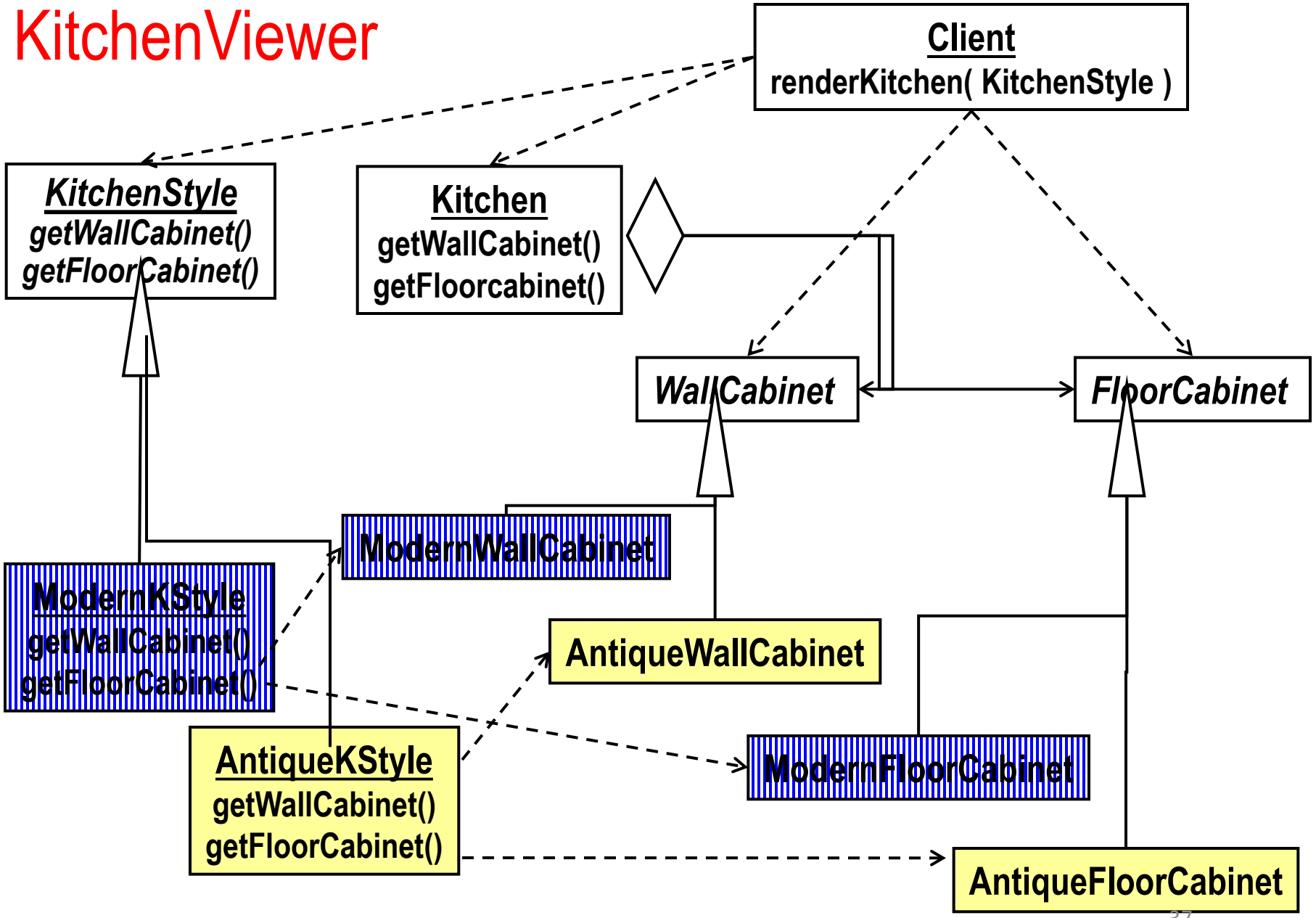
The Idea behind the Abstract Factory Design Pattern



renderKitchen(KitchenStyle myStyle) in KitchenViewer

```
// Create the wall cabinets : Type determined by the class of myStyle
WallCabinet wallCabinetl = myStyle.getWallCabinet () ;
WallCabinet wallCabinet2 = myStyle.getWallCabinet ( ) ;
// Create the floor cabinets : Type determined by the class of myStyle
// Create the kitchen object (in the style required)
FloorCabinet  floorCabinetl = myStyle. get FloorCabinet () ;
FloorCabinet  floorCabinet2 = myStyle.get FloorCabinet ( ) ;
  ..............
Kitchen kitchen = new Kitchen();
kitchen.add( wallCabinetl , ... } ;
kitchen.add( wallCabinet2 , ... } ;
 ..................
kitchen.add( floorCabinetl , ... } ;
kitchen.add (floorCabinet2 , . . . J ;.....
```

Abstract Factory Design Pattern Applied to KitchenViewer



10 Mart Dersinin İnteraktif Çalışması

3 Mart Pazartesi özetlenen video ile birlikte (Martin Fowler «Working with Refactoring») tartışılacak konuşmanın linki

Martin Fowler «Not Just Code Monkeys»

https://www.youtube.com/watch?v=Z8aECe4lp44&t=25s

Videoları dinlemeniz ve not almanız istenmektedir.

Ders sırasında videoya dönüş için yapabilmek için alacağınız notlarda zaman kaydı yapabilirsiniz.

Derste öğrencilerden izlenilen videolar ve dersin kapsamı dahilinde görüşleri tartışılacaktır.