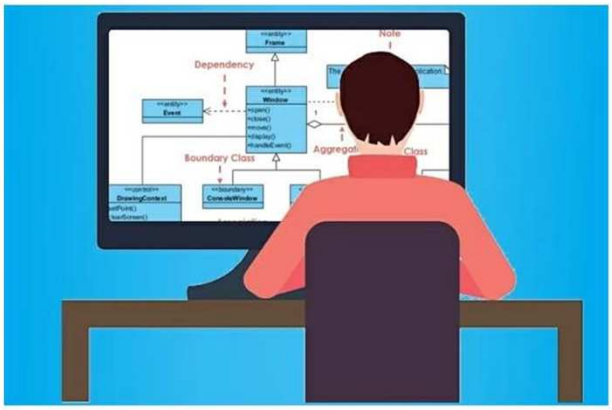
Software Design & Architecture
Spring 2022 - Week-03



مدرس: مہندس ماجد کلیم

جامعہ بحریہ، واقعگاہ کراچی

Engr. Majid Kaleem

WEEKLY AGENDA

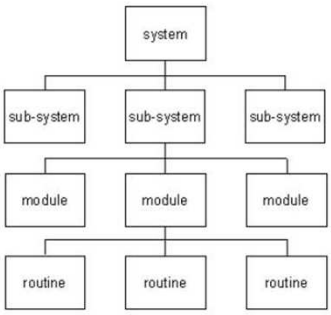
TENTATIVE WEEKLY DATES		TENTATIVE TOPICS
1	Mar 7th – Mar 11th	INTRODUCTION TO THE COURSE; DEFINING SOFTWARE ARCHITECTURE & DESIGN CONCEPTS
2	Mar 14th – Mar 18th	DESIGN PRINCIPLES; OBJECT-ORIENTED DESIGN WITH UML
3	Mar 21st – Mar 25th	SYSTEM DESIGN & SOFTWARE ARCHITECTURE; OBJECT DESIGN, MAPPING DESIGN TO CODE
4	Mar 28th – Apr 1st	FUNCTIONAL DESIGN; UI DESIGN; WEB APPLICATIONS DESIGN **ASSIGNMENT & QUIZ #1**
5	Apr 4th – Apr 8th	MOBILE APPLICATION DESIGN; PERSISTENCE LAYER DESIGN
6	Apr 11th – Apr 15th	CREATIONAL DESIGN PATTERNS
7	Apr 18th – Apr 22nd	STRUCTURAL DESIGN PATTERNS **ASSIGNMENT & QUIZ #2**
8	Apr 25th – Apr 29th	BEHAVIORAL DESIGN PATTERNS
← MID TERM EXAMINATIONS →		
9	May 9th – May 13th	INTERACTIVE SYSTEMS WITH MVC ARCHITECTURE; SOFTWARE REUSE
10	May 16th – May 20th	ARCHITECTURAL DESIGN ISSUES; ARCHITECTURE DESCRIPTION LANGUAGES (ADLS)
11	May 23rd – May 27th	ARCHITECTURAL STYLES/PATTERNS & DESIGN QUALITIES
12	May 30th – Jun 3rd	ARCHITECTURAL STYLES/PATTERNS & DESIGN QUALITIES **ASSIGNMENT & QUIZ #3**
13	Jun 6th – Jun 10th	QUALITY TACTICS; ARCHITECTURE DOCUMENTATION
14	Jun 13th – Jun 17th	ARCHITECTURAL EVALUATION TECHNIQUES
15	Jun 20th – Jun 24th	MODEL DRIVEN DEVELOPMENT **ASSIGNMENT (PRESENTATIONS) & QUIZ #4**
16	Jun 27th – Jul 1st	REVISION WEEK
← FINAL TERM EXAMINATIONS →		

SYSTEM DESIGN

- **System** is a group/collection of interacting or interrelated entities that form a unified whole.
- **System design** is the process of designing the elements of a *system* such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system.
- **System Analysis** is the process that *decomposes* a system into its component pieces for the purpose of defining how well those components interact to accomplish the set requirements.

PURPOSE OF SYSTEM DESIGN

- The purpose of the System Design process is to provide sufficient *detailed data* and *information* about the system and its system elements to enable the implementation consistent with architectural entities as defined in models and views of the system architecture.



ELEMENTS OF A SYSTEM

1. **Architecture** - This is the *conceptual model* that defines the structure, behavior and more views of a system. For instance, we can use flowcharts to represent and illustrate the architecture.
2. **Modules -** These are components that handle *one specific task* in a system. A combination of the modules make up the system.
3. **Components -** This provides a *particular function* or group of related functions. They are made up of modules.
4. **Interfaces -** This is the *shared boundary* across which the components of a the system exchange information and relate.
5. **Data -** This the *management* of the information and data flow.

FLOW OF EVENTS

- The use cases begin to describe what your system will do. To actually build the system, though, you'll need more specific details.
- These details are written as the ***flow of events***. *The purpose of the flow of events is to* document the flow of logic through the use case.
- This document will describe in detail ***what*** the user of the system will do and what the system itself will do.
- Although it is detailed, the flow of events is still implementation−independent. You can assume as you are writing the flow that there will be an automated system.
- However, you shouldn't yet be concerned with whether the system will be built in C++, C#, or Java.
- The goal here is describing *what the system* will do, not *how the system will do it. The flow of events typically includes:*

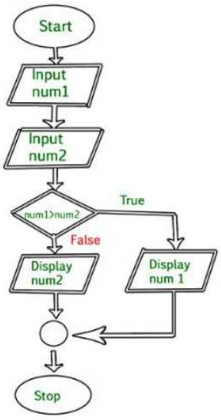
Flow of Events

- A brief description
- Preconditions
- Primary flow of events
- Alternate flow of events
- Postconditions

- https://www.projectmanagementdocs.com/template/project-documents/use-case-document/#axzz6pIVLZHni

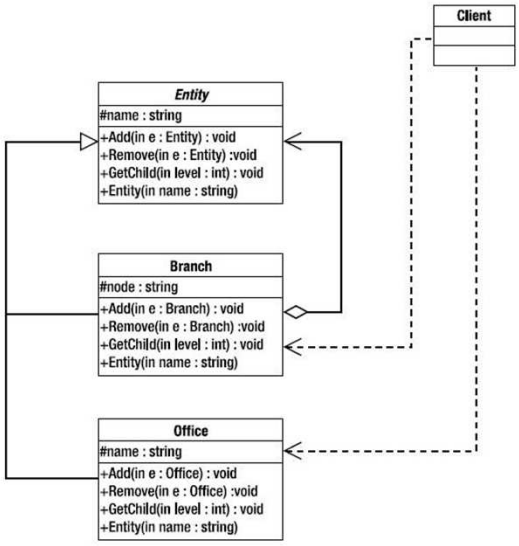
Mapping Design to Code

- *Design is nothing but collection of illustrations and drawings*.
- Those must be implemented and transformed into code (using programming language /pseudo code).
- *This part is actually covered in Lab where UML/other diagrams are represented in C#. For example:*

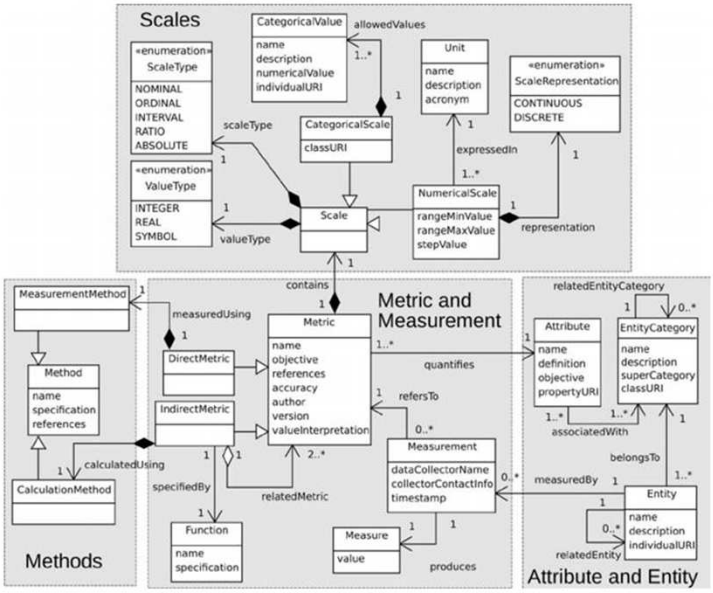



```c
int main()
{
    int num1, num2, largest;

    /*Input two numbers*/
    printf("Enter two numbers:\n");
    scanf("%d%d", &num1, &num2);

    /*check if a is greater than b*/
    if (num1 > num2)
        largest = num1;
    else
        largest = num2;

    /*Print the largest number*/
    printf("%d", largest);

    return 0;
}
```

EXAMPLE-UML TO CODE (DESIGN)



EXAMPLE-UML TO CODE (DESIGN)



EXAMPLE-UML TO CODE (SYMBOLS)

Program element	Diagram element	Meaning
Class	Class –attribute +operation( )	Types and parameters specified when important; access indicated by + (public), (private), and # (protected).
Interface	<<interface>> IClass +operation( )	Name starts with I. Also used for abstract classes.
Note	descriptive text	Any descriptive text.
Package	Package	Grouping of classes and interfaces.

EXAMPLE-UML TO CODE (SYMBOLS)

Inheritance	A △ B	B inherits from A.
Realization	A △ B	B implements A.
Association	A —— B	A and B call and access each other's elements.
Association (one way)	A ——> B	A can call and access B's elements, but not vice versa.
Aggregation	A ◇—— B	A has a B, and B can outlive A.
Composition	A ◆—— B	A has a B, and B depends on A.

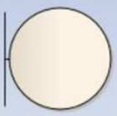
UML TO CODE (TYPES OF RELATIONSHIPS)

- *Association* - If two classes in a model need to communicate with each other, there must be a link between them, and that can be represented by an association (connector).
- *Inheritance* is an *"is-a" relationship and is a coding element in which a class makes it possible to define* subclasses that share some or all of the main class characteristics.
- *Interfaces* is a *"behaves-as" or "looks-like" relationship and is an element of coding where you define a common* set of properties and methods for use with the design of two or more classes.

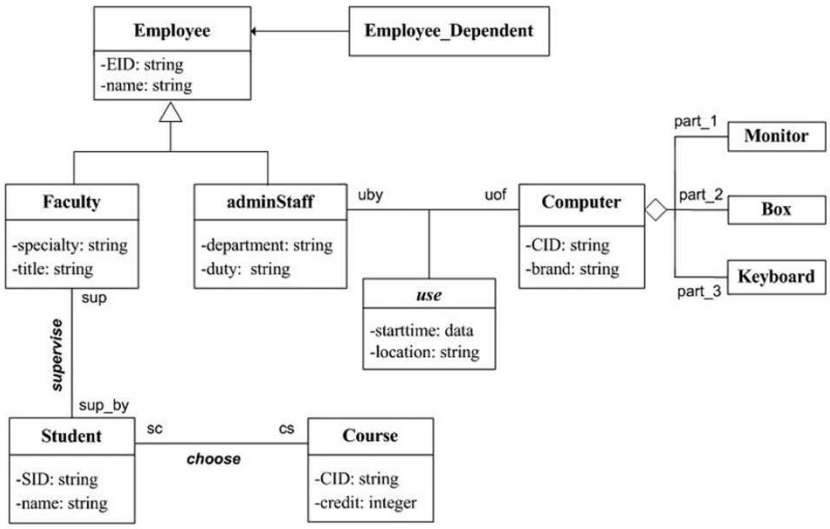
UML TO CODE (TYPES OF RELATIONSHIPS)

- *Aggregation* is an *"is-part-of" or "has-a" relationship and simply indicates a whole-part relationship.*
- *Composition* (a.k.a. Composite Aggregation) is a *"uses-a" relationship and is a strong type of aggregation and* means that a class cannot exist by itself. It must exist as a member of another class. For example, a button class must exist as part of a container such as a form.

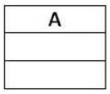
EXAMPLE-UML TO CODE (SYMBOLS)

Stereotype	What does it represent?
 Entity	Represents information that is important and persisted within the business/problem domain. This stereotype is used most often in domain modeling.
 Controller	Represents a portion/component of the software that is responsible for implementing business logic and overall control over certain processes.
 Boundary	Represents a means by which an actor is able to interact with the system.

EXAMPLE-UML TO CODE (DESIGN)



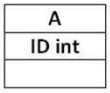
SIMPLE CLASS




```
public class A
{
 public A()
 {
  ...
 }
}
```

17

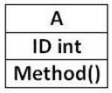
SIMPLE CLASS




```
public class A
{
 int ID;
 public A()
 {
  ...
 }
}
```

18

SIMPLE CLASS

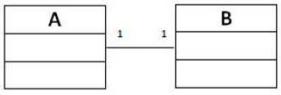



```
public class A
{
 int ID;
 public A()
 {
  ...
 }
 public int Method()
 {
  ...
 }
}
```

19

BI-DIRECTIONAL ASSOCIATION

Association	A ———— B	A and B call and access each other's elements.

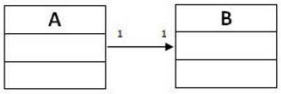



```
public class A
{
 Public B objB;
 public A()
 {
  ...
 }
}

public class B
{
 Public A objA;
 public B()
 {
  ...
 }
}
```

20

UNIDIRECTIONAL ASSOCIATION

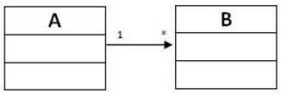
Association (one way) A ⟶ B A can call and access B's elements, but not vice versa.




```
public class A
{
 Public B objB
 public A()
 {
  ...
 }
}

public class B
{
 public B()
 {
  ...
 }
}
```

21

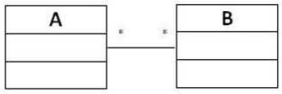
ASSOCIATION – ONE TO MANY




```
public class A
{
 public B objB[];
 public A()
 {
  ...
 }
}

public class B
{
public B()
 {
  ...
 }
}
```

22

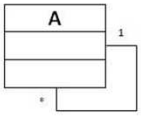
ASSOCIATION – MANY TO MANY




```
public class A
{
 Public B objB[];
 public A()
 {
  ...
 }
}

public class B
{
 Public A objA[];
 public B()
 {
  ...
 }
}
```

23

REFLEXIVE ASSOCIATION

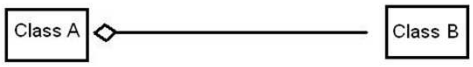



```
public class A
{
 Public A objA[];
 public A()
 {
  ...
 }
}
```

24

AGGREGATION

Aggregation A ◇—— B A has a B, and B can outlive A.




```
public class B
{
 public B()
 {
  ...
 }
}
public class A
{
 B objB;
 public A(B objB)
 {
  this.objB = objB;
 }
}
...
B objB = new B();
A objA = new A(objB);
```

25

COMPOSITION

Composition A ◆—— B A has a B, and B depends on A.

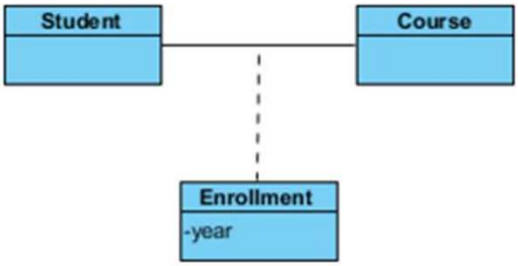

```
public class B
{
 public B()
 {
  ...
 }
}

public class A
{
 public A()
 {
  B objB = new B();
 }
}
```

26

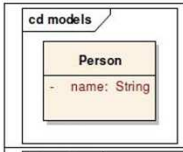
Association Class

- https://www.codeproject.com/Tips/596709/Implementation-of-Type-of-Association

	Association	Aggregation	Composition
Owner	No owner	Single owner	Single owner
Life time	Have their own lifetime	Have their own lifetime	Owner's life time
Child object	Child objects all are independent	Child objects belong to a single parent	Child objects belong to a single parent

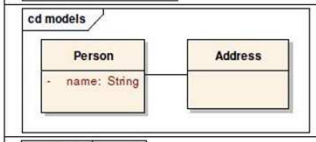


Example-UML to Java Code



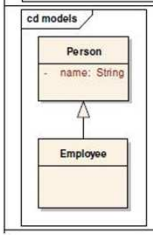
Class

```
public class Person{
   private String name;
}
```



Association

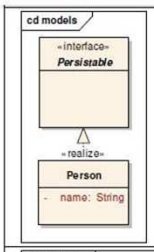
```
public class Person{
  private String name;
  private Address address;
}

public class Address{}
```



Inheritance

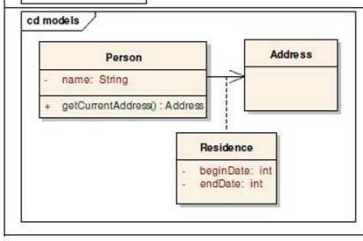
```
public class Employee
extends Person {}
```

EXAMPLE-UML TO JAVA CODE



Interface

```
public interface Persistable{
  public void Save();
}

public class Person
implements Persistable{
  public void Save(){}
}
```



Association class

```
public class Person{
  private Residence[] residences;
  public Address getCurrentAddress(){}
}

public class Residence{
  private Address address;
  private Date beginDate;
  private Date endDate;
}

public class Address{
}
```


EXAMPLE-UML TO C# CODE

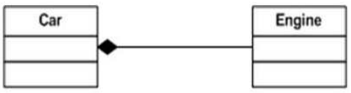


Figure 1 - Composition

```
public class Engine
{
 . . .
}
Public class Car
{
   Engine e = new Engine();
   .......
}
```

EXAMPLE-UML TO C# CODE

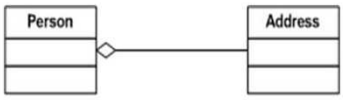
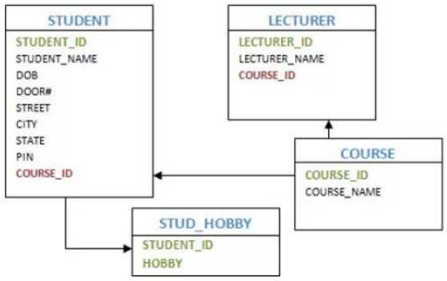


Figure 2 - Aggregation

```
public class Address
{
 . . .
}
Public class Person
{
private Address address;
public Person(Address address)
    {
this.address = address;
    }
    . . .
}
. . .
Address address = new Address();
Person person = new Person(address);
```

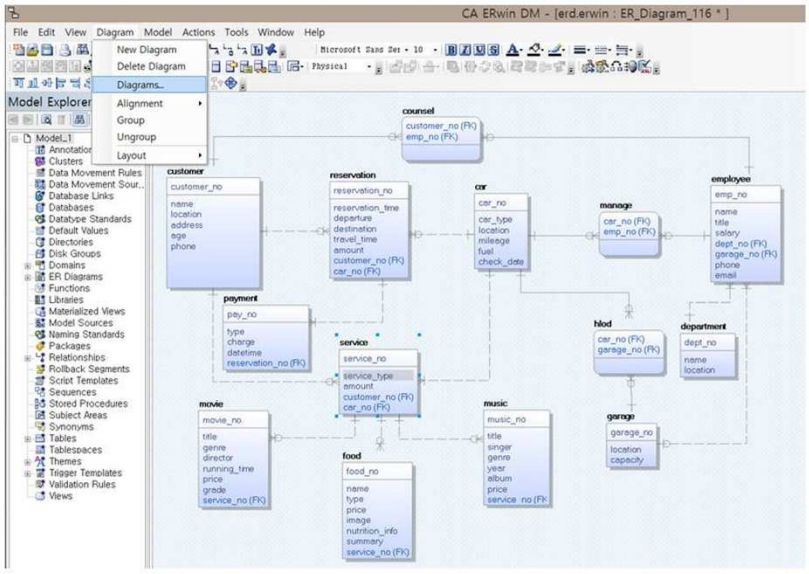

ERWIN DATA MODELER

- ***Learn ERWIN to convert database logical model to physical model.***



- https://slideplayer.com/slide/12053782/

ERWIN DATA MODELER

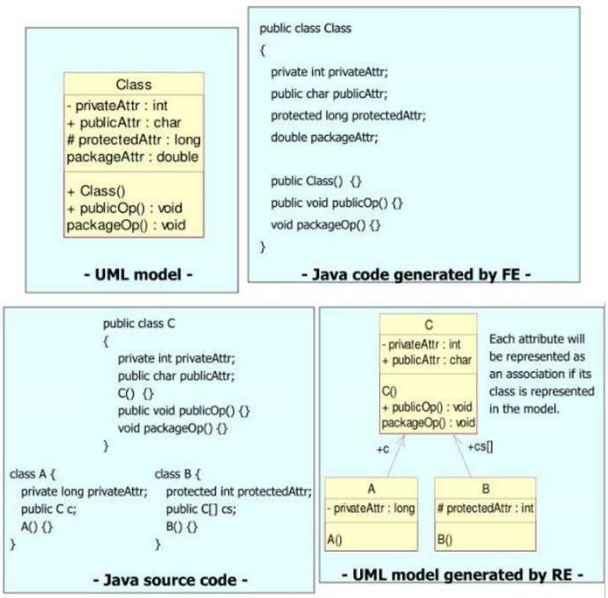


FORWARD & REVERSE ENGINEERING

- **Search UML Modeling Tools**
- **IMPORTANT URL TO VISIT**
- https://www.slideserve.com/lel/forward-and-reverse-engineering

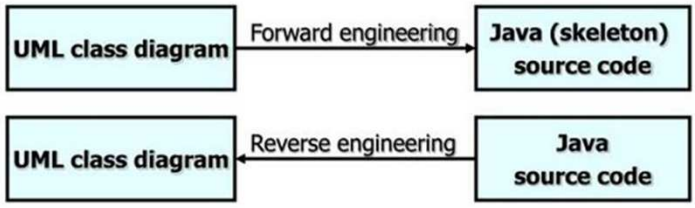
Rational Rose

FORWARD & REVERSE ENGINEERING



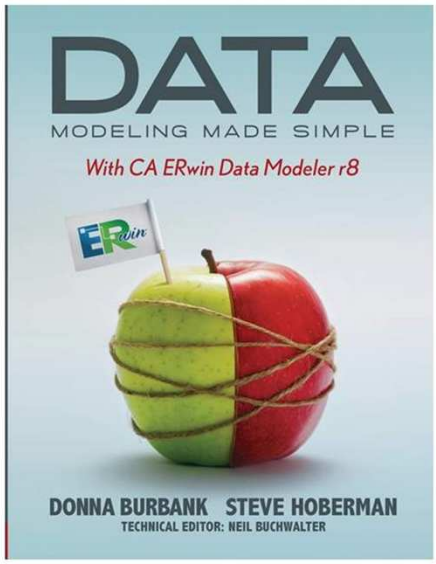
FORWARD & REVERSE ENGINEERING

- Forward Engineering – Produces code from diagram.
- Reserve Engineering – Produces diagram from code.



ERWIN DATA MODELER

- **Download pdf**




```
If(anyQuestions)
{
 askNow();
}
else
{
 thankYou();
 submitAttendance();
 endClass();
}
```

REFERENCES

1. *Software Architecture, Perspectives on an Emerging Discipline By Mary Shaw & David Garlan*
2. *The Art of Software Architecture, Design Methods & Techniques By Stephen T. Albin*
3. *Essential Software Architecture By Ian Gorton*
4. *Microsoft Application Architecture Guide By Microsoft*
5. *Design Patterns, Elements of Reusable Object-Oriented Software By by Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides*
6. *Refactoring, Improving the Design of Existing Code By Martin Fowler & Kent Beck*