



CSN-103: Fundamentals of Object Oriented Programming

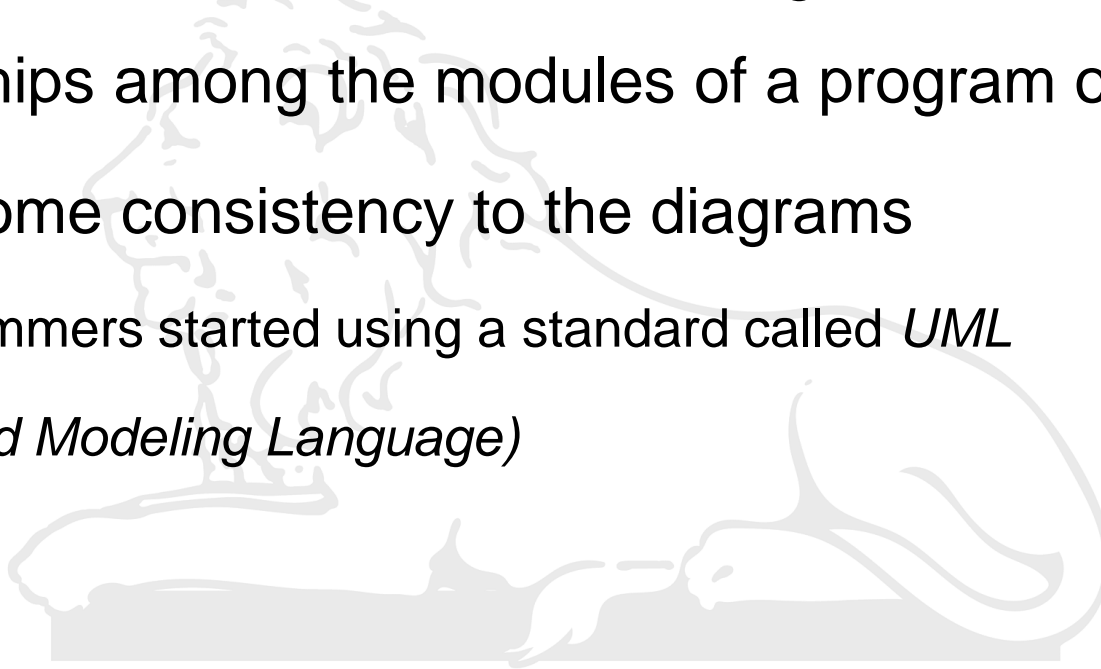
Instructor: Dr. Rahul Thakur

Assistant Professor, Computer Science and Engineering, IIT Roorkee



Unified Modeling Language

- Unified Modeling Language (**UML**)
- **Structure Charts:** To illustrate the organizational relationships among the modules of a program or system
- To add some consistency to the diagrams
 - Programmers started using a standard called *UML* (*Unified Modeling Language*)



Class Diagram

- UML class diagrams allow us to denote
 - Contents of classes
 - Member variables
 - Member methods of a class
 - Relationships between classes
 - One class inherits from another
 - One class contains an object of another class

We can depict all the source code dependencies between classes

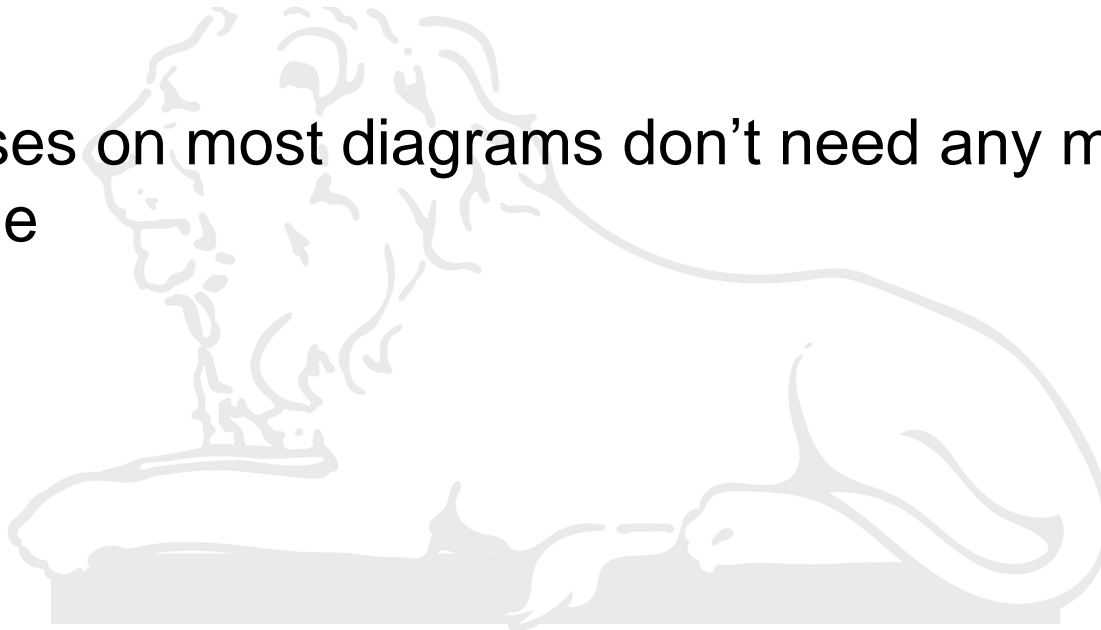
The Basics: Classes

- The simplest form of class diagram



```
public class Dialler  
{  
}  
}
```

- The classes on most diagrams don't need any more than their name



Compartments

- A class icon can be subdivided into compartments
 - The top compartment is for the name of the class
 - The second is for the variables of the class
 - The third is for the methods of the class

| Dialler |
|----------------------|
| digits nDigits |
| digit recordDigit |

```
Public class Dialler
{
    private double digits;
    int nDigits;
    public void digit(int n);
    protected boolean recordDigit(int n);
}
```

Visibility

- For class members
 - A dash (-) denotes private
 - A hash (#) denotes protected
 - A plus (+) denotes public
 - A plus (~) denotes package-private (default)

| Dialler |
|--------------------------|
| - digits ~ nDigits |
| + digit # recordDigit |

```
Public class Dialler
{
    private double digits;
    int nDigits;
    public void digit(int n);
    protected boolean recordDigit(int n);
}
```

Variable, Argument, and Return Type

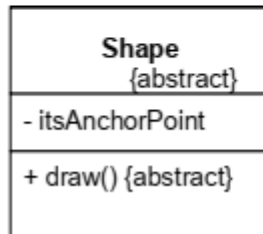
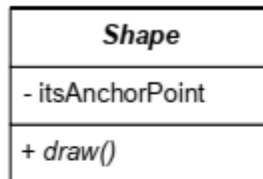
- The type of a variable/argument
 - Colon following the variable/argument name
- The return type of a method is shown after the colon following the function

| Dialler |
|--|
| - digits : double ~ nDigits : int |
| + digit (n : int) # recordDigit (n : int) : boolean |

```
Public class Dialler
{
    private double digits;
    int nDigits;
    public void digit(int n);
    protected boolean recordDigit(int n);
}
```

Abstract Classes

- Two ways to denote that a class/method is abstract
 - Italics: Class or method name (**Do Not Use in Exam**)
 - Use {abstract} word



```
public abstract class Shape
{
    private Point itsAnchorPoint;
    public abstract void draw();
}
```


«interface»

- «**interface**»: All the methods of classes marked with this stereotype are abstract
 - None of the methods can be implemented
- Moreover, «interface» classes can have no instance variables
- The only variables they can have are static variables.



```
interface Transaction
{
    public void execute();
}
```

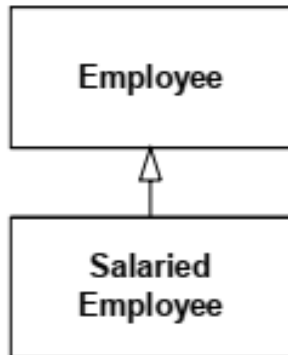
Relationships

- A relationship is a connection between two or more UML model elements
- Different kinds of relationships:



Inheritance

- Very careful with the arrowhead's **direction** in UML
- Example: **SalariedEmployee** is inherited from **Employee**

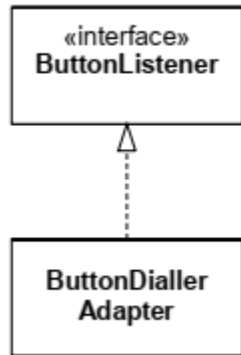


```
public class Employee
{
    ...
}
```

```
public class SalariedEmployee extends Employee
{
    ...
}
```

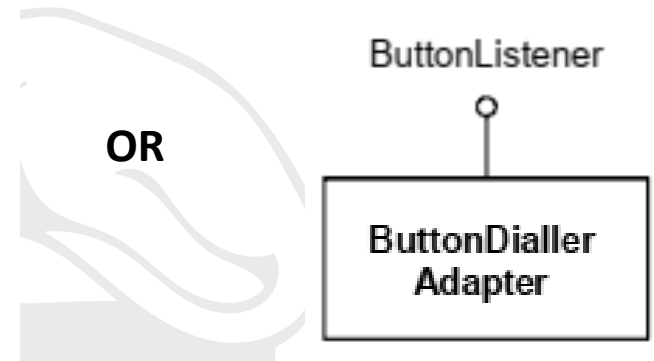
Interface

- UML has a special notation for inheritance
 - Used with a Java **interface**



```
interface ButtonListener
{
    ...
}

public class ButtonDiallerAdapter
    implements ButtonListener
{
    ...
}
```



Garbage Collection

Garbage Collection

- In Java, objects are dynamically operated by the **new** operator
- How objects are destroyed and memory is released for future reallocation?
- In some languages, dynamically allocated objects are **manually** released
- Java handles deallocation automatically using **Garbage Collection**

Garbage Collection

- When no reference to an object exists
 - Java assumes that the object is no longer needed
 - The memory occupied by object can be reclaimed
- No explicit need to destroy objects
- Garbage collection occurs rarely during program execution
 - `System.gc();`

- Example

```
Box b1=new Box();  
Box b2=new Box();  
b1=b2; OR b1=null;
```

- The first object referred by *b1* is available for garbage collection