Object Oriented Methodology

Week – *7*, Lecture – *1*

Creating Class Diagrams

SAURABH SRIVASTAVA

VISITING FACULTY

IIIT LUCKNOW

Designing the Class Model

An advantage of using Object-Oriented Methodology is that it resembles well with the real world

In the real world, we have entities, and they have different relationships with each other

The Class Model represents a view of the system, where these entities are represented as classes

Classes have properties, exhibited behaviour and inter-relationships

- The properties become the fields of the class and the behaviour is implemented via its methods
- Good OO practices recommend keeping the fields private and methods public

Usually, the Domain Model provides enough information to create the Class Model

- This is because Domain Models describe these domain entities, terminologies and other relationships
- It is thus, usually the first step of the design process, to come up with a Class Model for the product

The common way to showcase the Class Model is with the help of Class Diagrams

Representing Classes

Classes are represented in a Class Diagram by Rectangular boxes

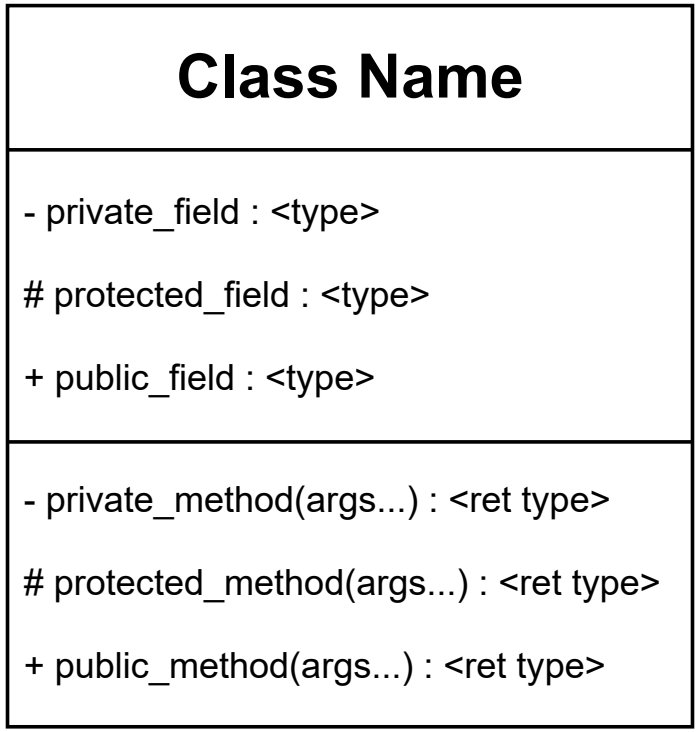
Usually, the box has three compartments dividing it horizontally

- The top compartment holds the class name
- The middle compartment is commonly used to show the details of the fields
- The bottom compartment describes details of the methods

The access type for any member can be shown by prepending following characters –

- **-** means the member is private
- **#** means the member is protected
- **+** means the member is public

Representing a Class



A Class in a Class Diagram

Representing Classes

Classes are represented in a Class Diagram by Rectangular boxes

Usually, the box has three compartments dividing it horizontally

- The top compartment holds the class name
- The middle compartment is commonly used to show the details of the fields
- The bottom compartment describes details of the methods

The access type for any member can be shown by prepending following characters –

- **-** means the member is private
- **#** means the member is protected
- **+** means the member is public

In addition, abstract methods (as well as classes) are shown in italics

Static members are shown in underlined style

Representing Relationships (1/3)

The classes involved in a typical software project seldom work in isolation …

- … they have relationships

The general term for representing a relationship between two (or more) classes is *Association*

- Usually, it means that one or more objects of a type, are contained in an object of the other type
- It is shown by connecting the classes with a solid line

Showing Simple Associations

An *Association* between two classes

Representing Relationships (1/3)

The classes involved in a typical software project seldom work in isolation … they have relationships

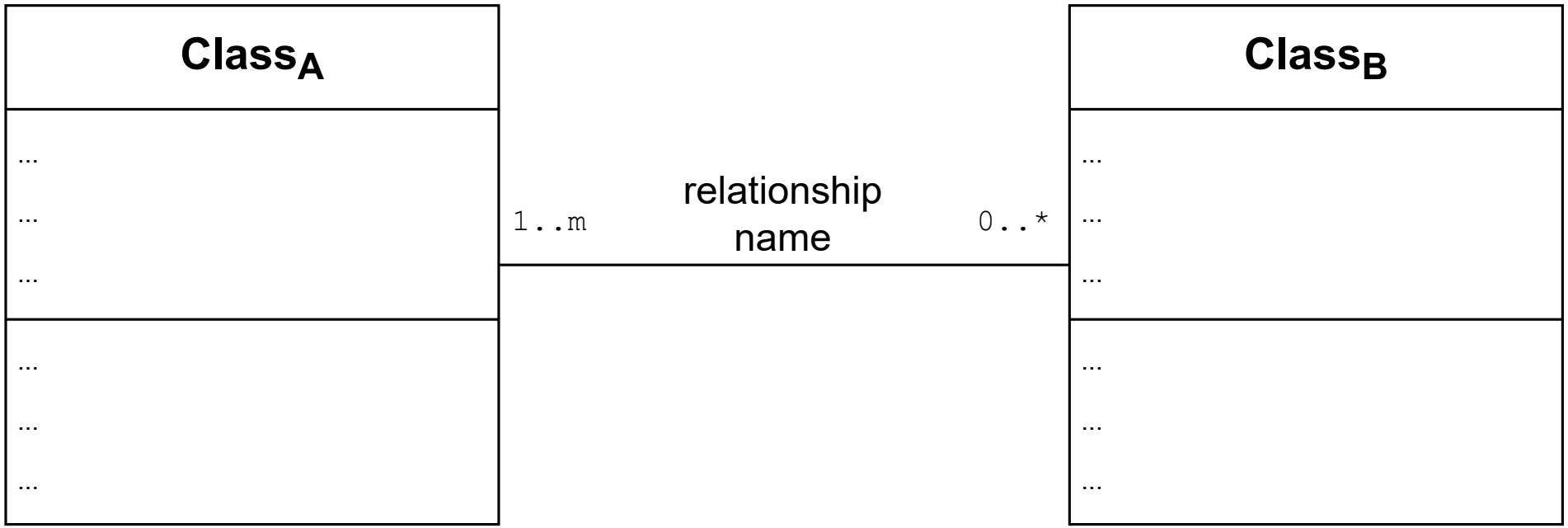
The general term for representing a relationship between two (or more) classes is *Association*

- Usually, it means that one or more objects of a type, are contained in an object of the other type
- It is shown by connecting the classes with a solid line

The association line can have attached decorations which can convey additional information, e.g.,

- A text written over or below the centre of the line represents a name for the relationship
- Texts written over or below on either side of the line shows the *multiplicity* of the classes in the association …
- … which represents the number of instances of the classes that participate in the relationship

Associations with details



An *Association* between two classes with the name of the relationship and multiplicities

Representing Relationships (1/3)

The classes involved in a typical software project seldom work in isolation … they have relationships

The general term for representing a relationship between two (or more) classes is *Association*

- Usually, it means that one or more objects of a type, are contained in an object of the other type
- It is shown by connecting the classes with a solid line

The association line can have attached decorations which can convey additional information, e.g.,

- A text written over or below the centre of the line represents a name for the relationship
- Texts written over or below on either side of the line shows the *multiplicity* of the classes in the association …
- … which represents the number of instances of the classes that participate in the relationship

Multiplicity is provided in a fixed format which looks like $\texttt{<l}_{\texttt{bound}}\texttt{>..<u}_{\texttt{bound}}\texttt{>}$

- $\texttt{l}_{\texttt{bound}}$ and $\texttt{u}_{\texttt{bound}}$ show the minimum and maximum number of objects that could be involved in the association
- A wildcard character – `*` – can replace either $\texttt{l}_{\texttt{bound}}$ or $\texttt{u}_{\texttt{bound}}$, and is interpreted as “zero or more”
- If both $\texttt{l}_{\texttt{bound}}$ and $\texttt{u}_{\texttt{bound}}$ are the same, e.g., the multiplicity is exactly `n`, then just writing `n` is also fine

Representing Relationships (2/3)

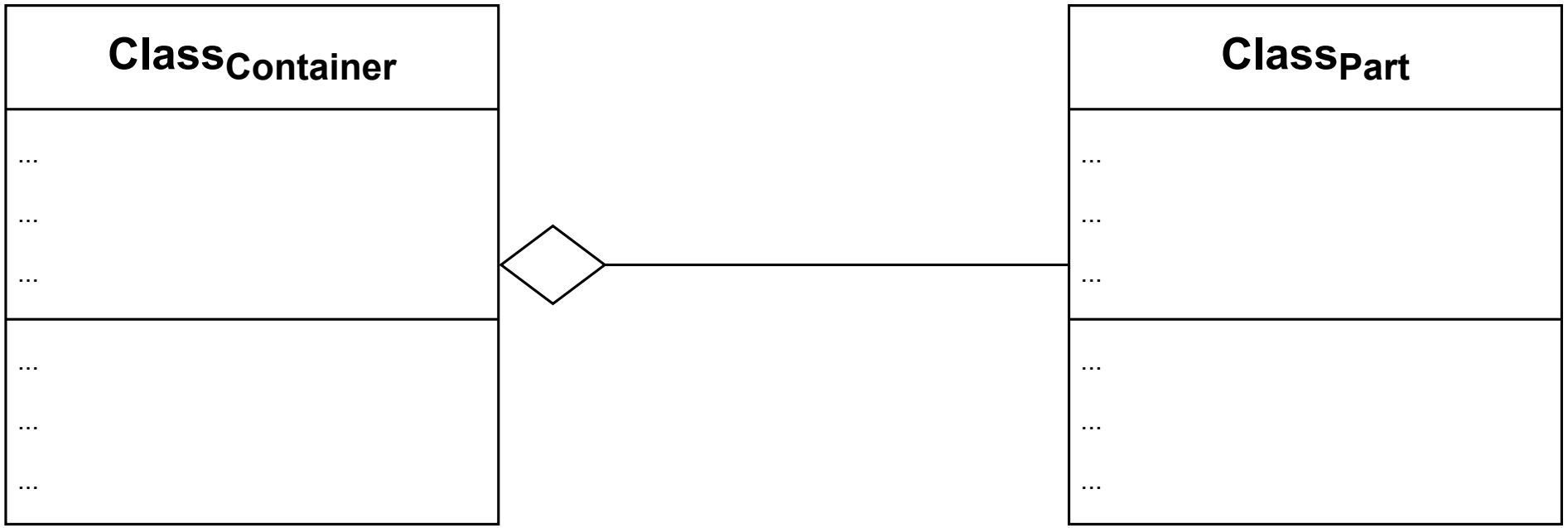
A specific form of Association is known as *Aggregation*

Aggregation represents the “part-of” relationship between two classes …

- … meaning that objects of one class are “part of” the object of another class

An Aggregation is shown by adding a diamond to the association line towards the containing class

Showing Aggregations



Aggregation between two classes - A $\text{Class}_{\text{Containter}}$ object contains $\text{Class}_{\text{Part}}$ object(s)

Representing Relationships (2/3)

A specific form of Association is known as *Aggregation*

- Aggregation represents the “part-of” relationship between two classes …
- … meaning that objects of one class are “part of” the object of another class
- An Aggregation is shown by adding a diamond to the association line towards the containing class

Aggregation Example – *Battery-operated Toy* and *Battery* …

- … a Battery-operated Toy (when operating normally) “contains” one or more instances of Battery

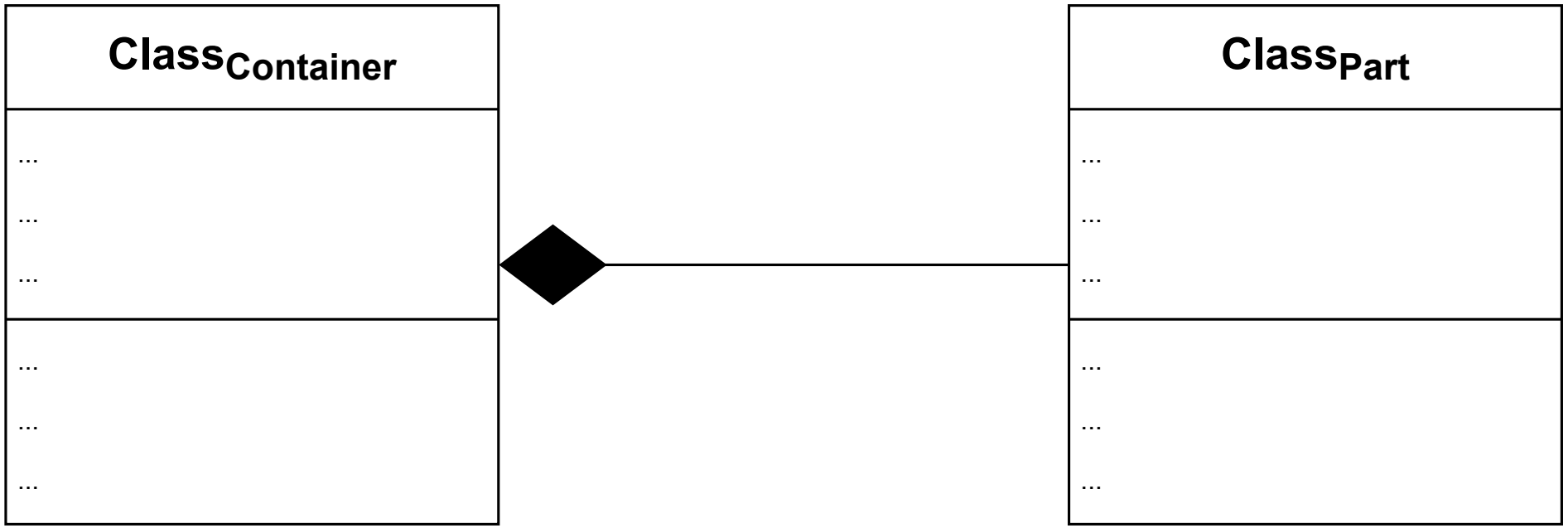
A specific form of Aggregation is *Composition* …

- … where the objects of the part class(es) have no significance independently
- Essentially, they live and (more importantly) die with their container object
- To represent a composition relationship, we make the diamond solid (instead of hollow)

Composition Example - *Battery-operated Remote-controlled Car* and *Car’s Remote* …

- The Remote is basically useless, if the car goes bad ☹

Showing Aggregations



Composition between two classes - A $Class_{Containter}$ object contains $Class_{Part}$ object(s)

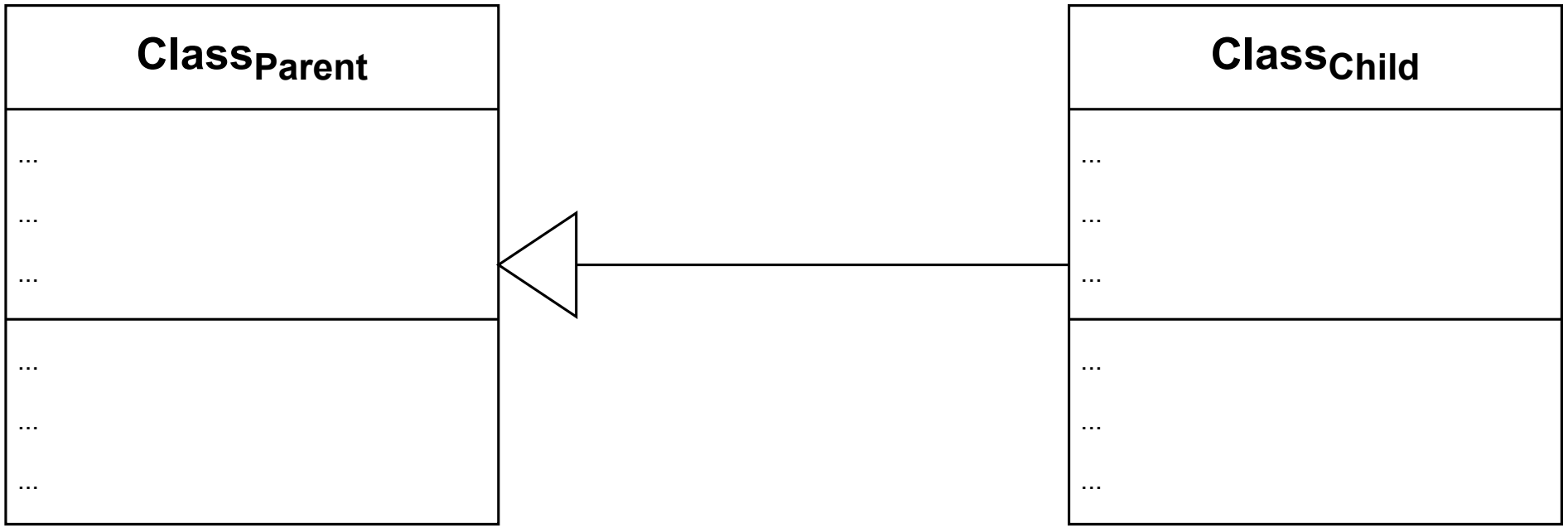
Representing Relationships (3/3)

The core relationship in OOM is Inheritance or the “type of” relationship

In a class diagram this relationship can be shown using a generalisation-specialisation connector

- It looks similar to an aggregation, but instead of a diamond, a hollow triangle is used
- The triangle is added towards the side of the Parent class – the other class being the Child

Showing Generalisations



An example of *Generalisation* & *Specialisation* - $Class_{Child}$ is a specific type of $Class_{Parent}$

Representing Relationships (3/3)

The core relationship in OOM is Inheritance or the “type of” relationship

In a class diagram this relationship can be shown using a generalisation-specialisation connector

- It looks similar to an aggregation, but instead of a diamond, a hollow triangle is used
- The triangle is added towards the side of the Parent class – the other class being the Child

It is common to see multiplicities with Aggregations – not so with Generalisations

- Although UML designer tools may not restrict you from doing so

Usually Generalisations do not require a relationship name as well – it is often obvious

- Still, UML designer tools may allow you to add relationship names to Generalisations as well

If you are adding a relationship name, it is a good idea to add an arrow to the association as well …

- … so that the reader can understand the relationship better

Homework !!

Go back to Lecture 2.1 (there are some corrections in it, so download the updated version)

- Try to understand the example class diagram in the slides

Create Class Diagrams for your projects

- This is a deliverable for your final project report

Watch the chapter on Class Diagrams starting at **9:39** in this tutorial (it's only about 8 mins :D)

- https://www.youtube.com/watch?v=WnMQ8HlmeXc

Additional Reading

We have only covered the bare minimum parts of Class Diagram

- The idea is to only learn essential UML for your project work
- You may read these links for a concise explanation of the other elements you may see in a Class Diagram
  https://sparxsystems.com/resources/tutorials/uml2/class-diagram.html
  https://en.wikipedia.org/wiki/Class_diagram