Object Oriented Methodology

Week – *3*, Lecture – *1*

Inheritance and Data Hiding

SAURABH SRIVASTAVA

VISITING FACULTY

IIIT LUCKNOW

The idea of Inheritance

Can you cook a pizza at home?

- Apparently, you can !!
- You can buy a pizza base from the market with along other ingredients like pizza sauce and cheese

You can then cook a simple pizza at home (given that you have all ingredients) as

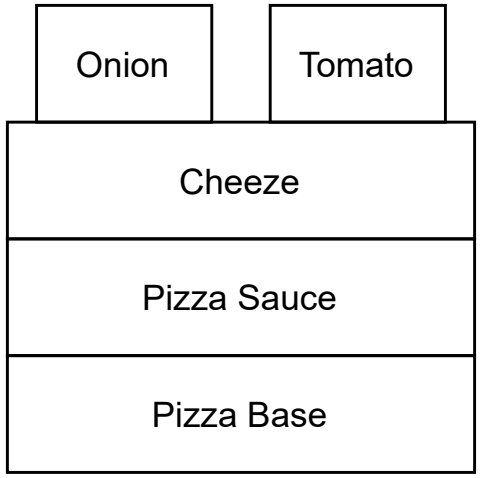
- Take a pizza base and apply pizza sauce over the base
- Grate some cheese over it and add the toppings of your choice over the same
- Cook the pizza base in an oven (I am not putting the temperature and duration here :P)

Assume that you like onions and tomato as toppings and your sibling likes paneer and capsicum

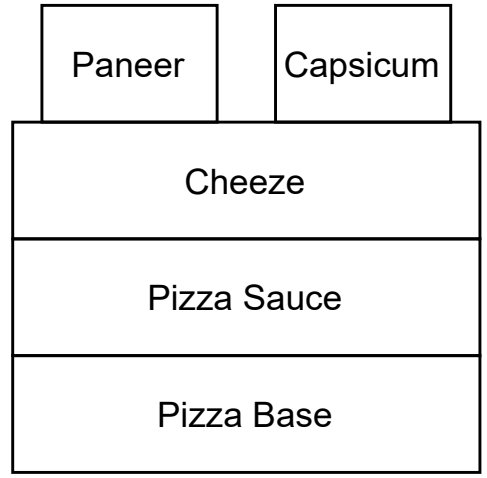
- Can the two of you do some of the stuff together?
- Yes.. you can both put the sauce and cheese together on the base, and then change the toppings

The idea of Inheritance

Onion and Tomato Pizza



Paneer and Capsicum Pizza



The idea of Inheritance

Can you cook a pizza at home?

- Apparently, you can !!
- You can buy a pizza base from the market with along other ingredients like pizza sauce and cheese

You can then cook a simple pizza at home (given that you have all ingredients) as

- Take a pizza base and apply pizza sauce over the base
- Grate some cheese over it and add the toppings of your choice over the same
- Cook the pizza base in an oven (I am not putting the temperature and duration here :P)

Assume that you like onions and tomato as toppings and your sibling likes paneer and capsicum

- Can the two of you do some of the stuff together?
- Yes.. you can both put the sauce and cheese together on the base, and then change the toppings
- **In other words, the process has some common elements, which can then be bifurcated**

Inheritance – Building stuff hierarchically

The idea of starting from common elements and then diversifying later is actually very helpful

- We use this idea often while building system

We often start with a generic entity, and keep on "extending" it as and when required

This entity, is often a *class*

- We start with a generic class, and keep on creating more classes by extending the generic one

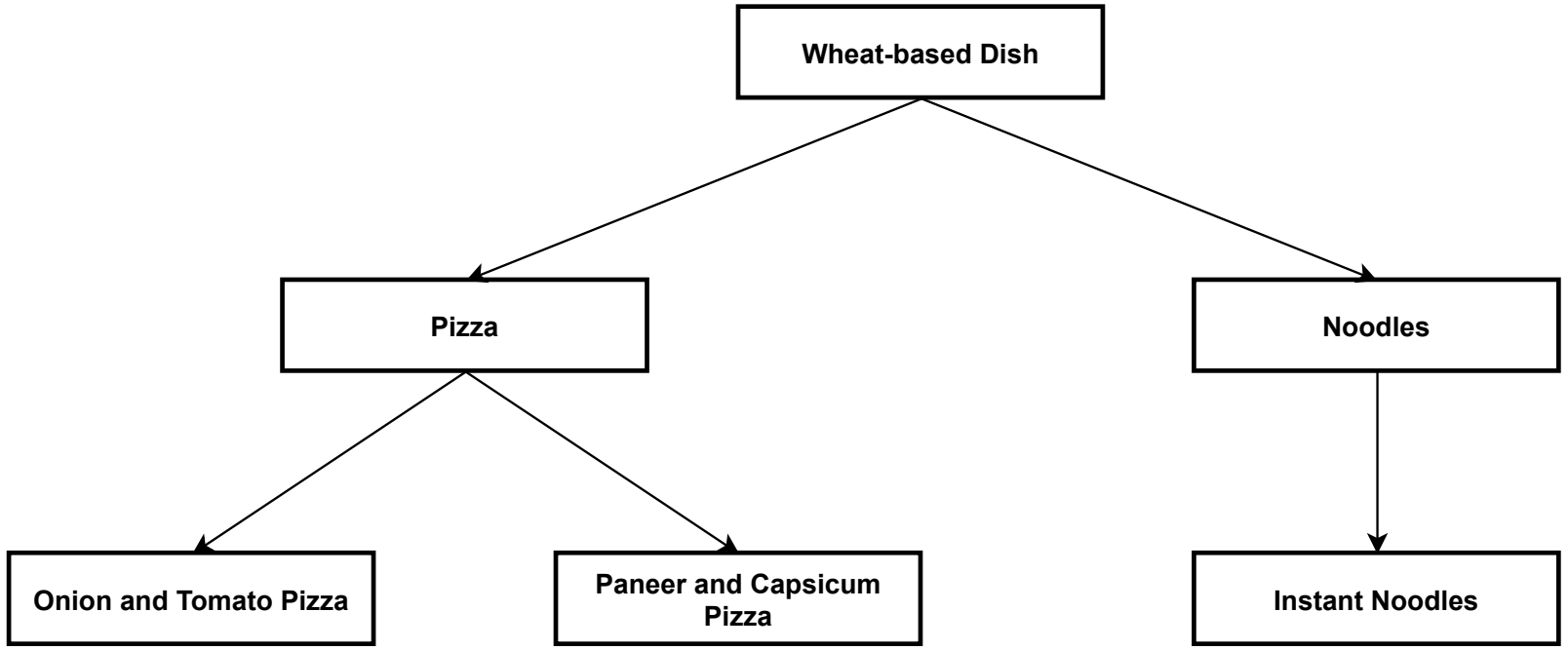
In this process, we add more attributes to entities in a hierarchical fashion

- We start with basic common attributes, and at each level, add some specific attributes to the hierarchy
- Overall, an entity at a lower level "inherits" attributes of the entities above it in the hierarchy

This phenomenon is known as *Inheritance*

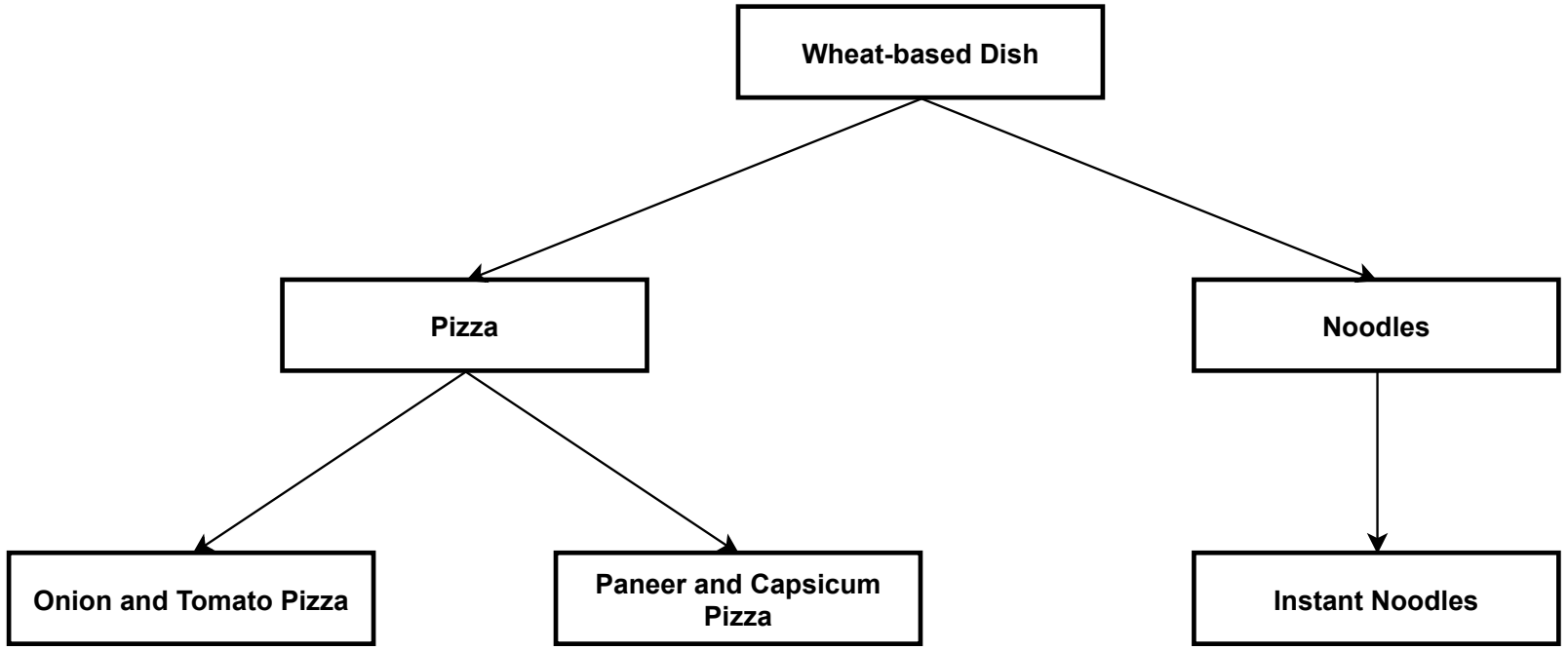
The class higher in hierarchy is called the *base class*, and those inheriting it are called *derived classes*

Inheritance – Building stuff hierarchically



An example of Inheritance Hierarchy

Inheritance – Building stuff hierarchically



An example of Inheritance Hierarchy

These are all classes – with those on lower levels, “inheriting” properties of their ancestors in the tree

Inheritance – Building stuff hierarchically

The idea of starting from common elements and then diversifying later is actually very helpful

- We use this idea often while building system

We often start with a generic entity, and keep on "extending" it as and when required

This entity, is often a *class*

- We start with a generic class, and keep on creating more classes by extending the generic one

In this process, we add more attributes to entities in a hierarchical fashion

- We start with basic common attributes, and at each level, add some specific attributes to the hierarchy
- Overall, an entity at a lower level "inherits" attributes of the entities above it in the hierarchy

This phenomenon is known as *Inheritance*

The class higher in hierarchy is called the *base class*, and those inheriting it are called *derived classes*

We can view the derived classes as specialisations of their base classes

Hiding and Exposing Features

When we create a class, it is important to figure out what should be “visible” to outside world?

- For example, you really don’t need to know the Recipe of the Pizza base – you just need to use it !!

Also, should a basic behaviour of a class be open for modification?

- For example, can the Pizza base be used to create noodles? (I seriously don’t know :P)

These decisions can seriously impact the use of Inheritance

- If a class is too restrictive, i.e. it has very few visible or extensible traits, it may not be useful
- On the other hand, if it is too permissive, there is a risk of too much exposure or change of behaviour

The Open-Closed Principle

"Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."

- Beyer Meyer,
  Object-oriented software construction. Vol. 2. Englewood Cliffs: Prentice hall, 1997

So what is the difference between "modification" and "extension"?

- Assume that it is possible to turn the Pizza base "into" noodles – this is the example of modification
- You can apply sauce and toppings "on top of" the Pizza base – this is the example of extension

In essence, inheritance should ideally be used to turn general entities into specific entities …

- … but the general entities should not lose their identity, i.e., their basic behaviour

Inheritance Example – New Cricket format

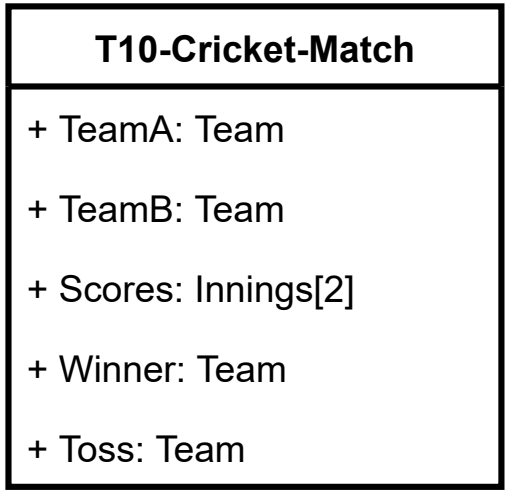
Assume that you are planning to build a class to represent a match in a new Cricket format …

- … i.e., in addition to Tests, ODIs and T20s
- Let us assume we are representing a format called T10 (10 overs each side, single innings)

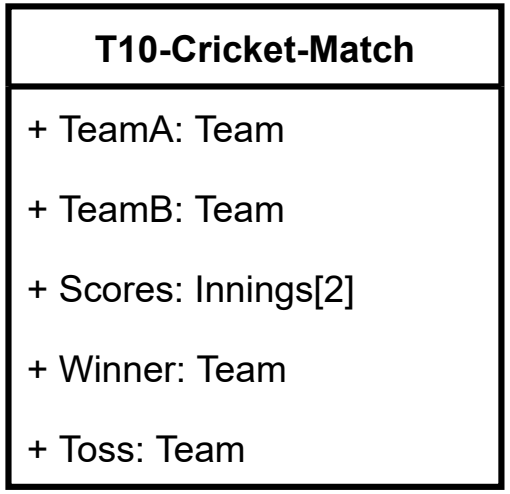
What information should be stored in the fields of this class?

- We need to store the result of the toss
- We need to store the names of the playing eleven
- We need to store the scorecard for the first innings
- We need to store the scorecard for the second innings
- We need to store the name of the winning team, etc.

Inheritance Example – New Cricket format



Inheritance Example – New Cricket format



Here, `Team` and `Innings` are two additional classes that capture required details

Inheritance Example – New Cricket format

Assume that you are planning to build a class to represent a match in a new Cricket format …

- … i.e., in addition to Tests, ODIs and T20s
- Let us assume we are representing a format called T10 (10 overs each side, single innings)

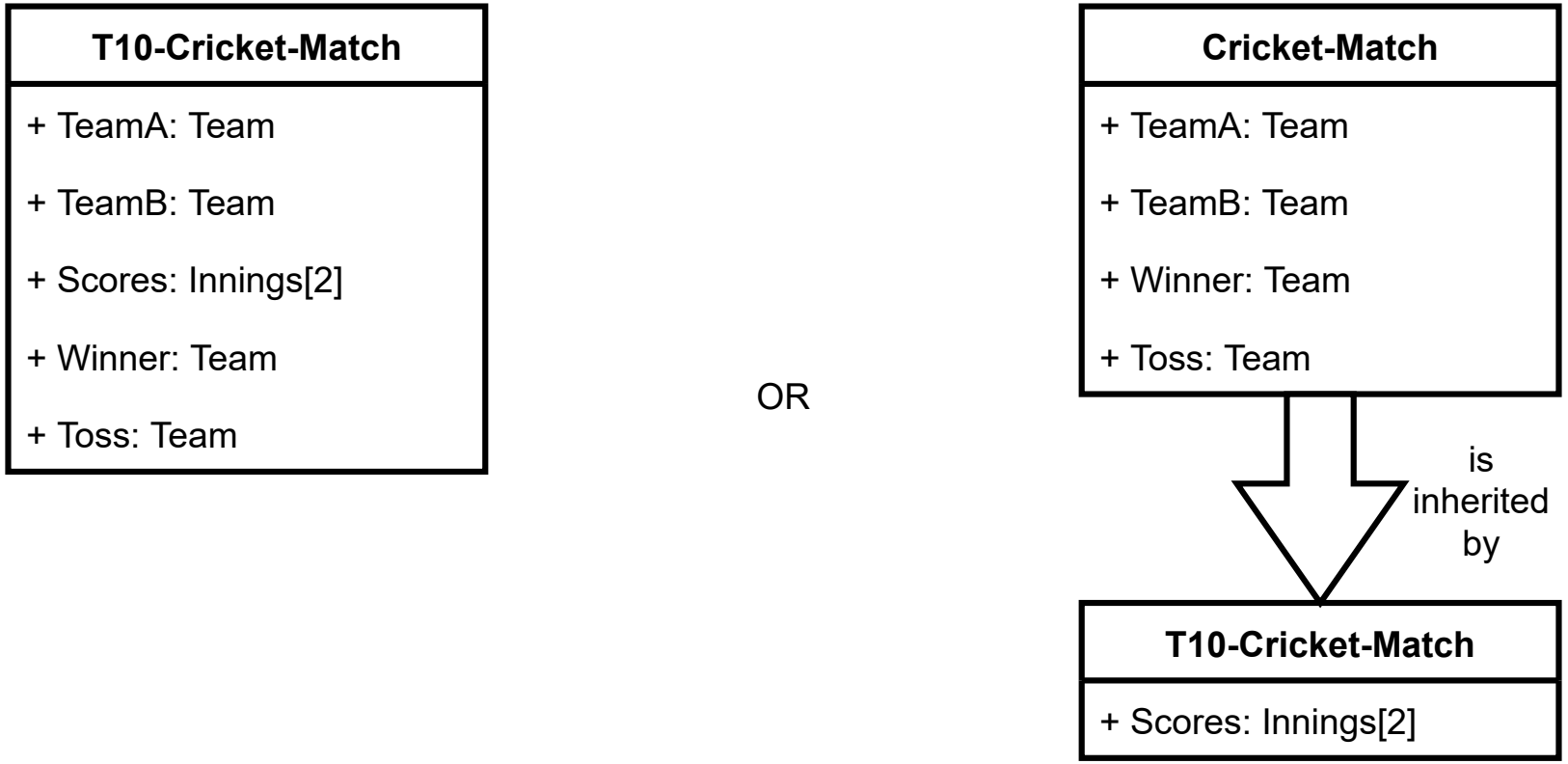
What information should be stored in the fields of this class?

- We need to store the result of the toss
- We need to store the names of the playing eleven
- We need to store the scorecard for the first innings
- We need to store the scorecard for the second innings
- We need to store the name of the winning team, etc.

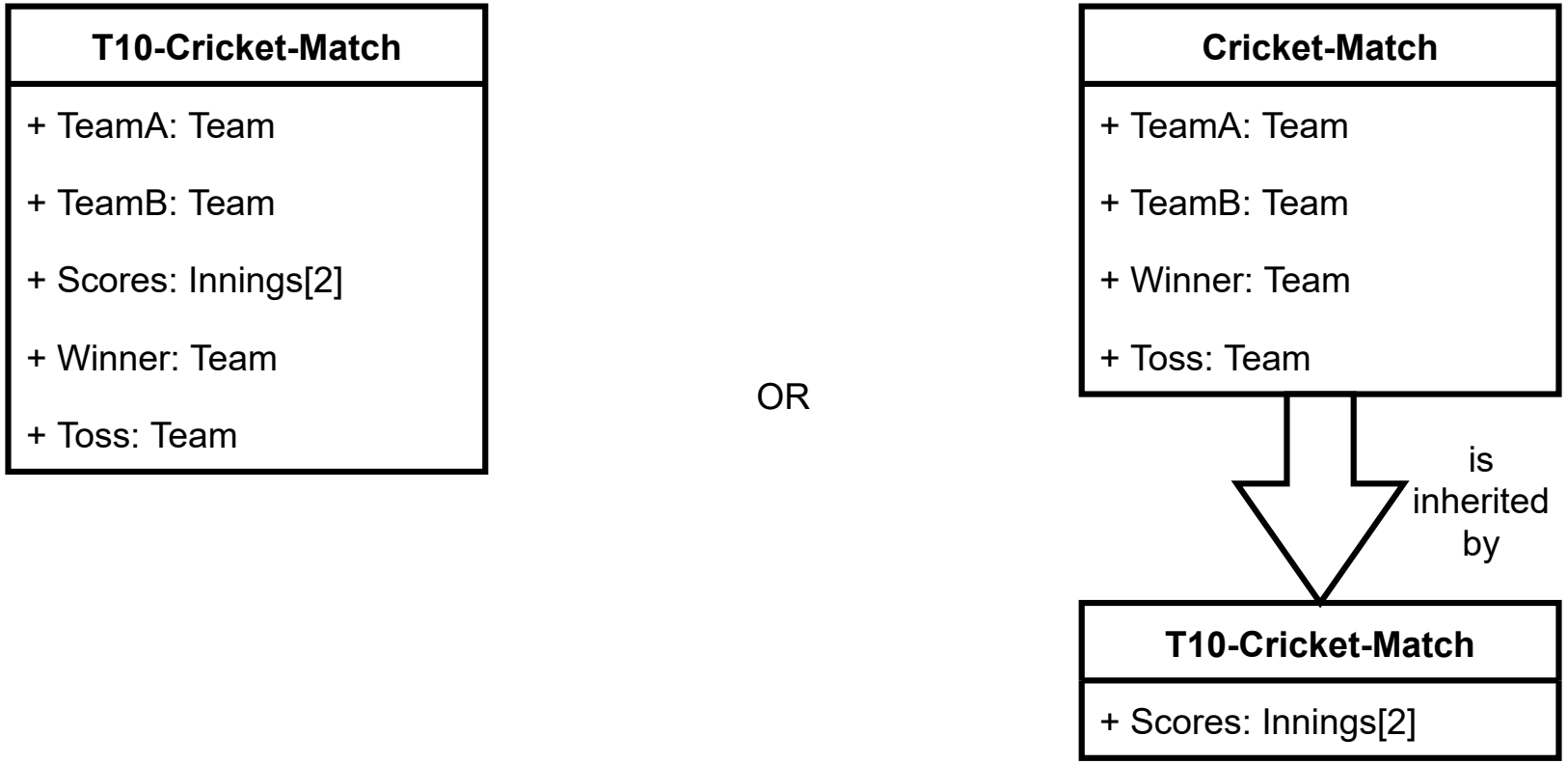
Here, except for the scorecards (which will be for 10 overs), everything else is common …

- … for all Cricket matches – including Tests, ODIs and T20s

Inheritance Example – New Cricket format



Inheritance Example – New Cricket format



Provided that all fields of `Cricket-Match` are ”inheritable”, this is another way to create the `T10-Cricket-Match` class

Inheritance and Methods

While inheriting fields seem a straightforward idea, inheriting methods is slightly more complex

Methods perform some operations – usually over the fields – changing states of Objects

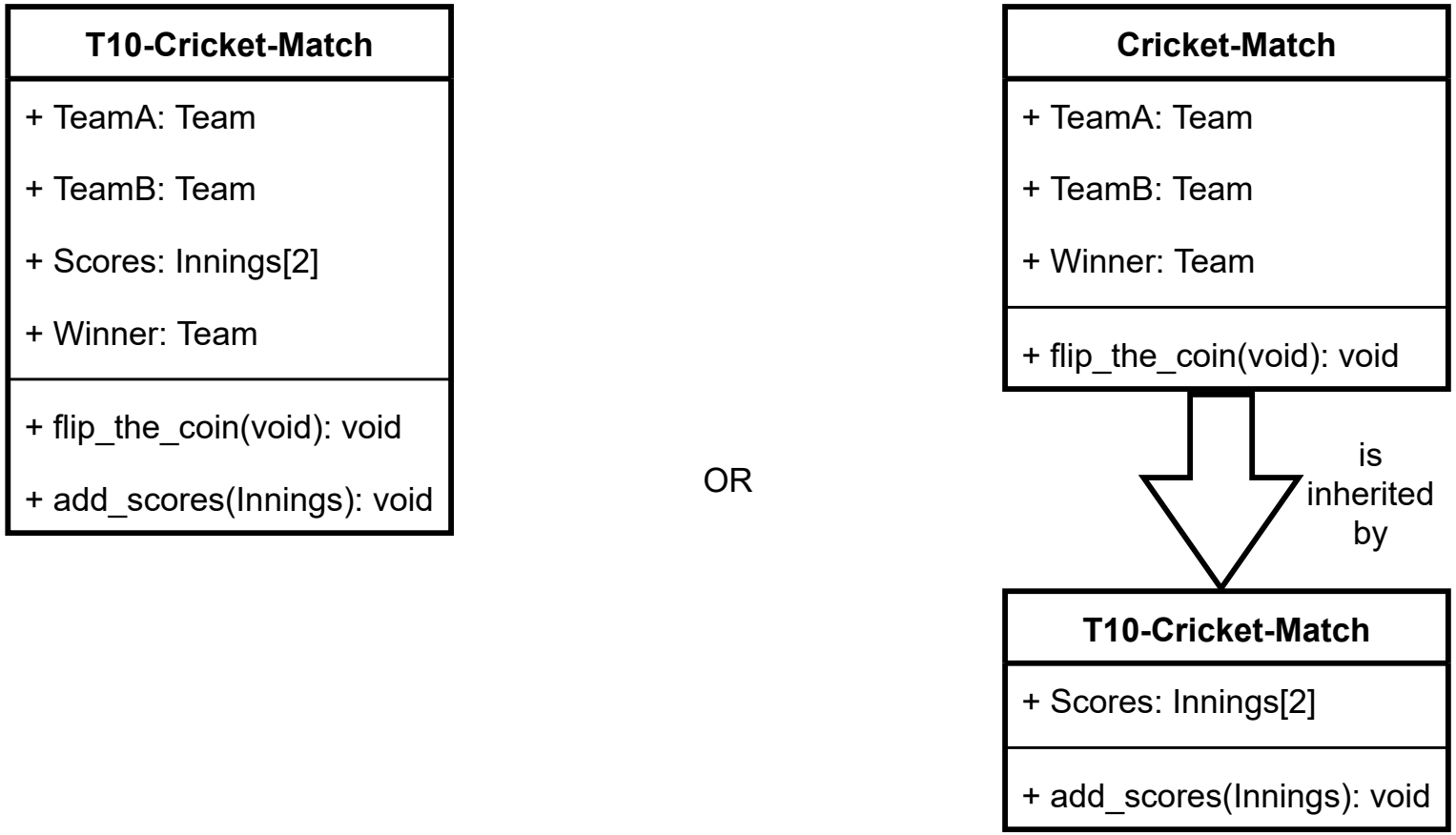
Example: We need to have a method called *flip_the_coin()* in T10-Cricket-Match

- The method imitates the toss at the start of a match
- The value of the field Toss is set to NULL at the start …
- … and it is set to one of the two Teams after it is called
- Keep in mind that tosses are required at the beginning of all Cricket matches

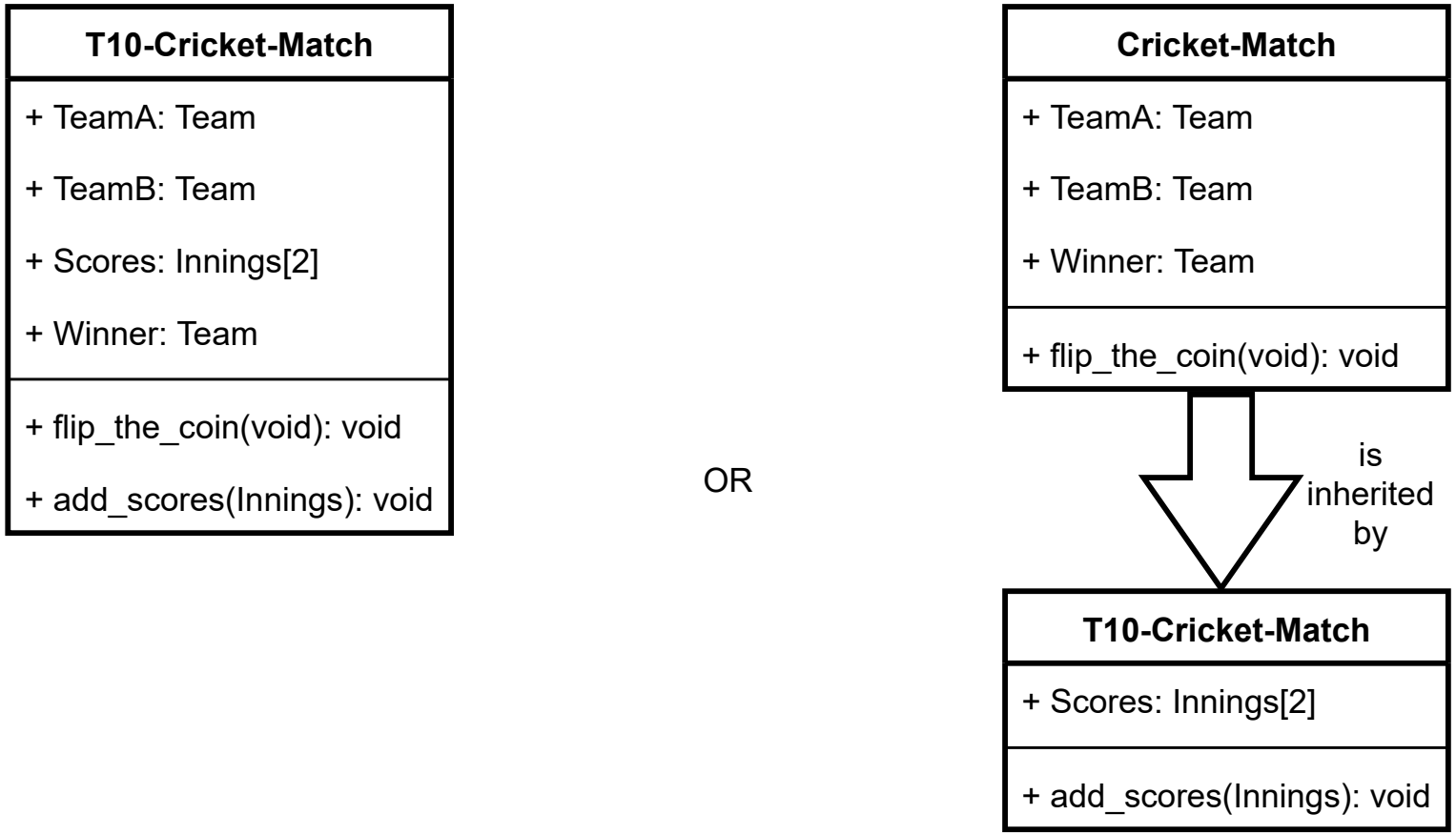
Another example: Method *add_scores()* in T10-Cricket-Match

- The method adds the scorecards to the match – one innings at a time
- The value of the field Scores point to two NULL elements
- On the first call, the first NULL element becomes non-NULL
- On the second call, the second NULL element also becomes non-NULL

Inheritance Example – New Cricket format



Inheritance Example – New Cricket format



The added methods, with and without inheritance

Visible vs Non Visible Members

Should all fields of a base class be accessible to the derived classes?

- Maybe not, if the data is essential for the internal working of the base class …
- … and it does not concern the derived classes

Should all methods of a base class be accessible to the derived classes?

- Maybe not, if the method implements a core behaviour of the base class …
- … and it does not concern the derived classes

In general,

- We usually hide the fields of the base class from the derived classes, and …
- … let the derived classes access the methods

It is “not a rule”, but if you choose to do otherwise, convince yourself that it’s the right decision

OO Programming languages usually provide ways to decide visibility of each class member individually

Overriding – Modifying inherited behaviour

In the OO world, a very powerful feature is extension or modification of “inherited” behaviour`

This phenomenon is called Overriding

We will study more about overriding later, but to summarise …

- … it involves “redefining” a method of the base class in a derived class

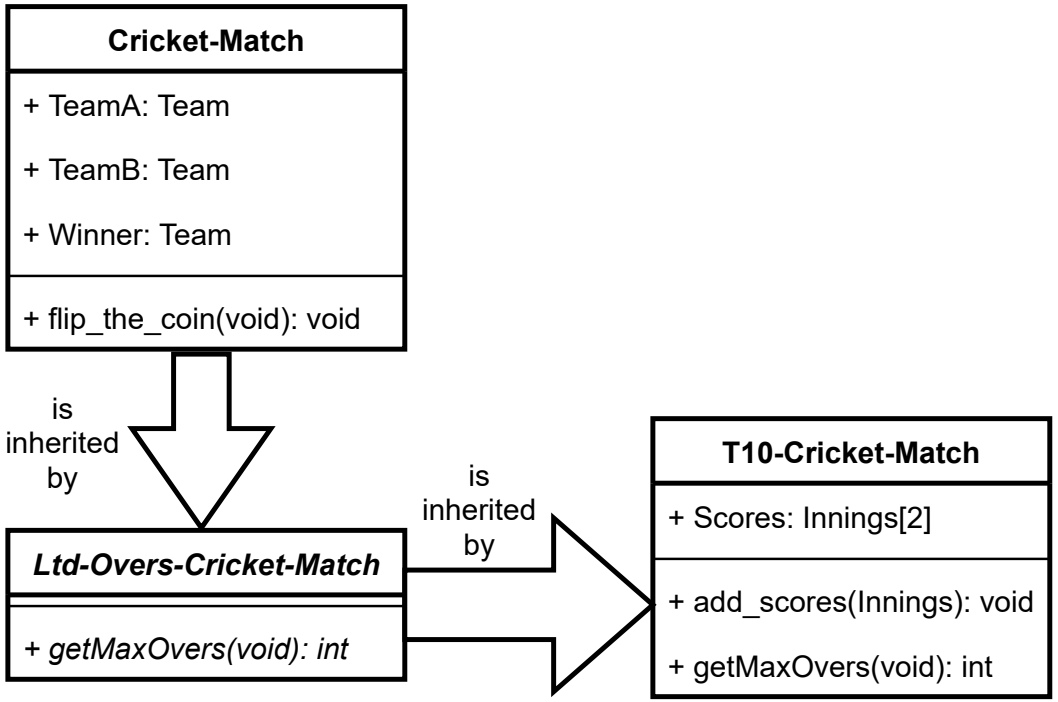
The method may have had a definition in the base class, or, it may just have a declaration

- In the latter case, the method is considered to be “abstract”
- Another term for such a method is “pure virtual function”
- We will get to know about them later in the course

If a base class has even one abstract method, the class too, becomes abstract

- This means that such no objects of such a class can be created …
- Abstract classes are, thus, meant for only inheritance (e.g. a Pizza with no toppings :P)

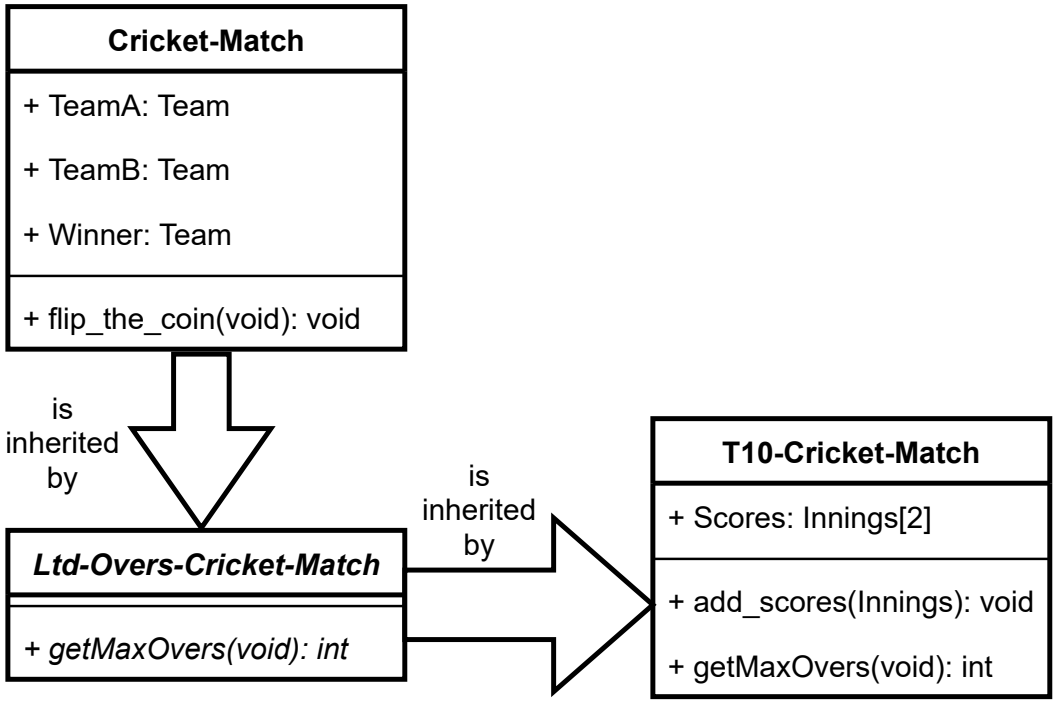
Inheritance Example – New Cricket format



Ltd-Overs-Cricket-Match is an Abstract class

getMaxOvers() is an Abstract or Pure Virtual method

Inheritance Example – New Cricket format



Ltd-Overs-Cricket-Match is an Abstract class

getMaxOvers() is an Abstract or Pure Virtual method

For instance, we can add an abstract class to this hierarchy, which has an abstract method (check your homework)

Homework !!

For the T10 example, think about other types of class hierarchies you can come up with

- **Hint**: If there is a class called `Ltd-Overs-Cricket-Match`, what could be its peer?

For the example in the last slide, why do you think the method `getMaxOvers()` is abstract?

- **Hint**: Can we really provide a definition for it in `Ltd-Overs-Cricket-Match`? If not, why?