159 XTENDED

A WEEKLY REVIEW

UML CLASS DIAGRAMS

Pull up **Socrative.com**

and join room **XTEND159**

WHAT IS CONFUSING TO YOU ABOUT UML CLASS DIAGRAMS?

Pull up **Socrative.com**

and join room **XTEND159**

UML CLASS DIAGRAM REVIEW

- Classes
- Objects
- Interfaces
- Abstract Classes
- "Extends"
- "Implements"

WE KNOW WHAT THESE THINGS ARE…

BUT HOW CAN WE MODEL THEM?

AND THE RELATIONSHIPS BETWEEN THEM?

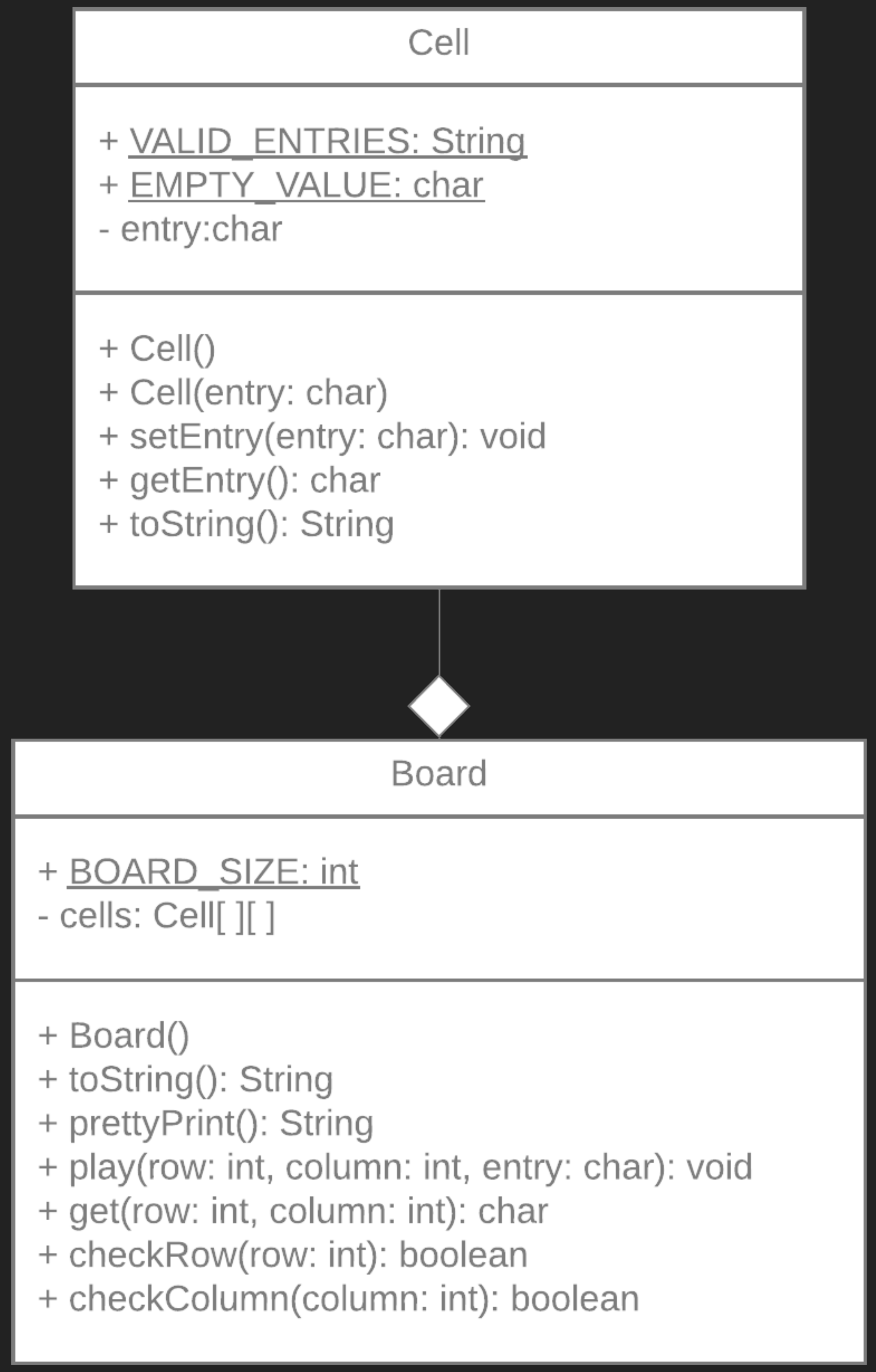
UML???? HUH?

- UML stands for Unified Modeling Language.
- From the Unified Modeling Language User Guide…

  "[UML] is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system."
- Less jargon: It's an industry standard language for modeling software systems.

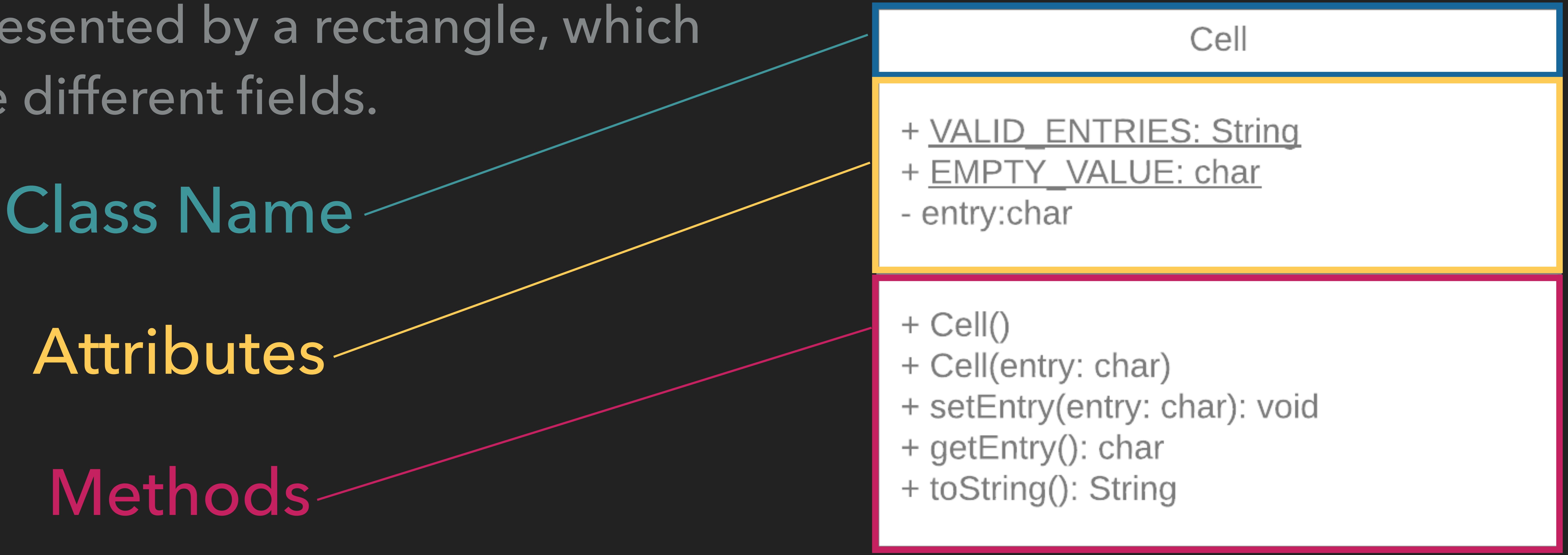
UML???? HUH?

- UML Class Diagrams give us a graphical representation of classes, interfaces, relationships between these things, and more.
- These diagrams are used to specify the requirements for a system.



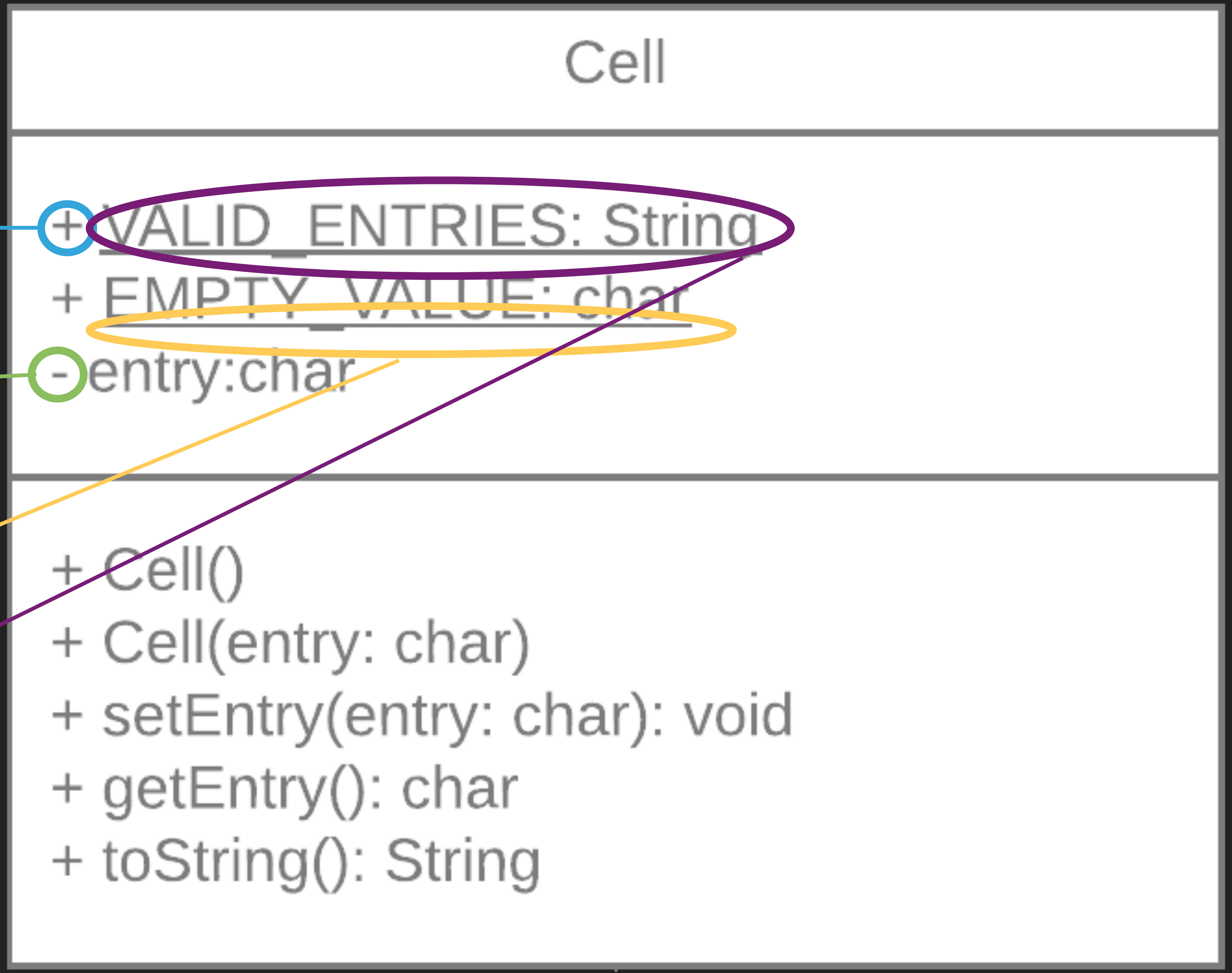
MODELING A CLASS

- A class is represented by a rectangle, which contains three different fields.



MODELING A CLASS

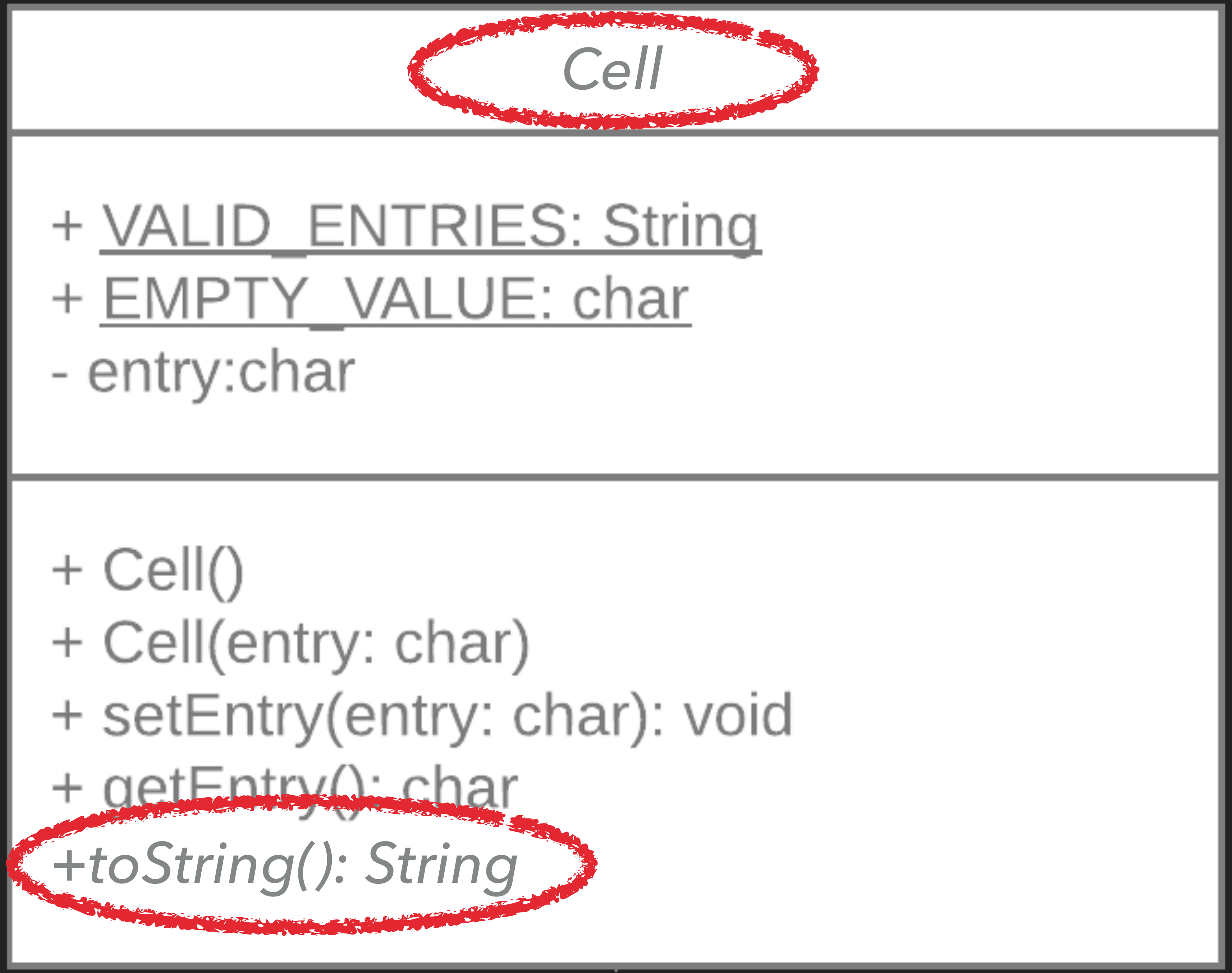
- Key:
  - + denotes public visibility
  - - denotes private visibility
  - # denotes protected visibility
  - Underlined attributes are static
  - CAPITAL attributes are final



```
Cell
------------------------------
+ VALID_ENTRIES: String
+ EMPTY_VALUE: char
- entry:char
------------------------------
+ Cell()
+ Cell(entry: char)
+ setEntry(entry: char): void
+ getEntry(): char
+ toString(): String
```

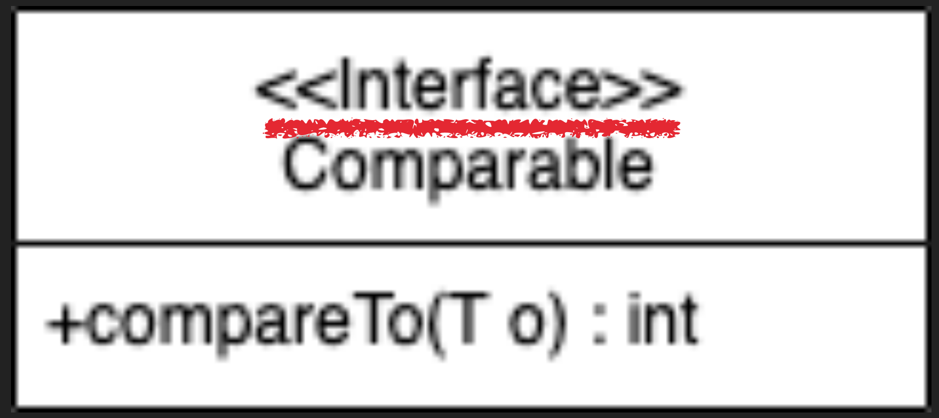
MODELING AN ABSTRACT CLASS

- An abstract class is denoted by the class name being italicized.
- Similarly, an abstract method is denoted by the method signature being italicized.



MODELING AN INTERFACE

- An interface is denoted by `<<Interface>>` above the Interface name.
- Methods and attributes are represented the same way as they are with a class.



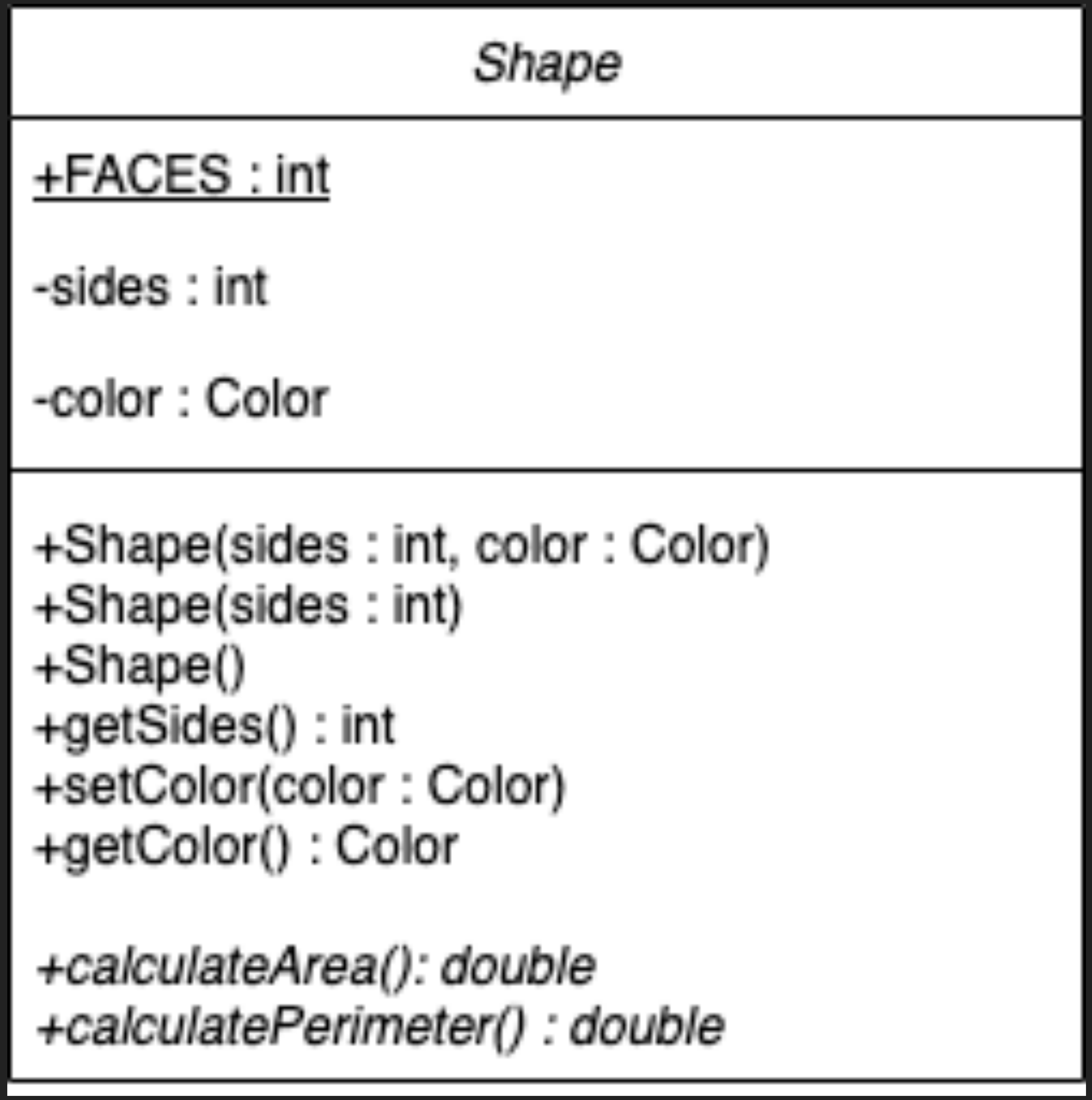
ACTIVITY: BUILDING A UML CLASS DIAGRAM

```
public abstract class Shape
{
    // Instance Variables
    public static final int FACES = 1;
    private int sides;
    private Color color;

    public Shape(int sides, Color color)
    {
        this.sides = sides;
        this.color = color;
    }

    public Shape(int sides)
    {
        this(sides, Color.black);
    }

    public Shape()
    {
        this(4, Color.black);
    }

. . .
```

```
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public abstract double calculateArea();

    public abstract double calculatePerimeter();
}
```

- On a sheet of paper, create a UML class diagram representation of this class.

ACTIVITY: BUILDING A UML CLASS DIAGRAM

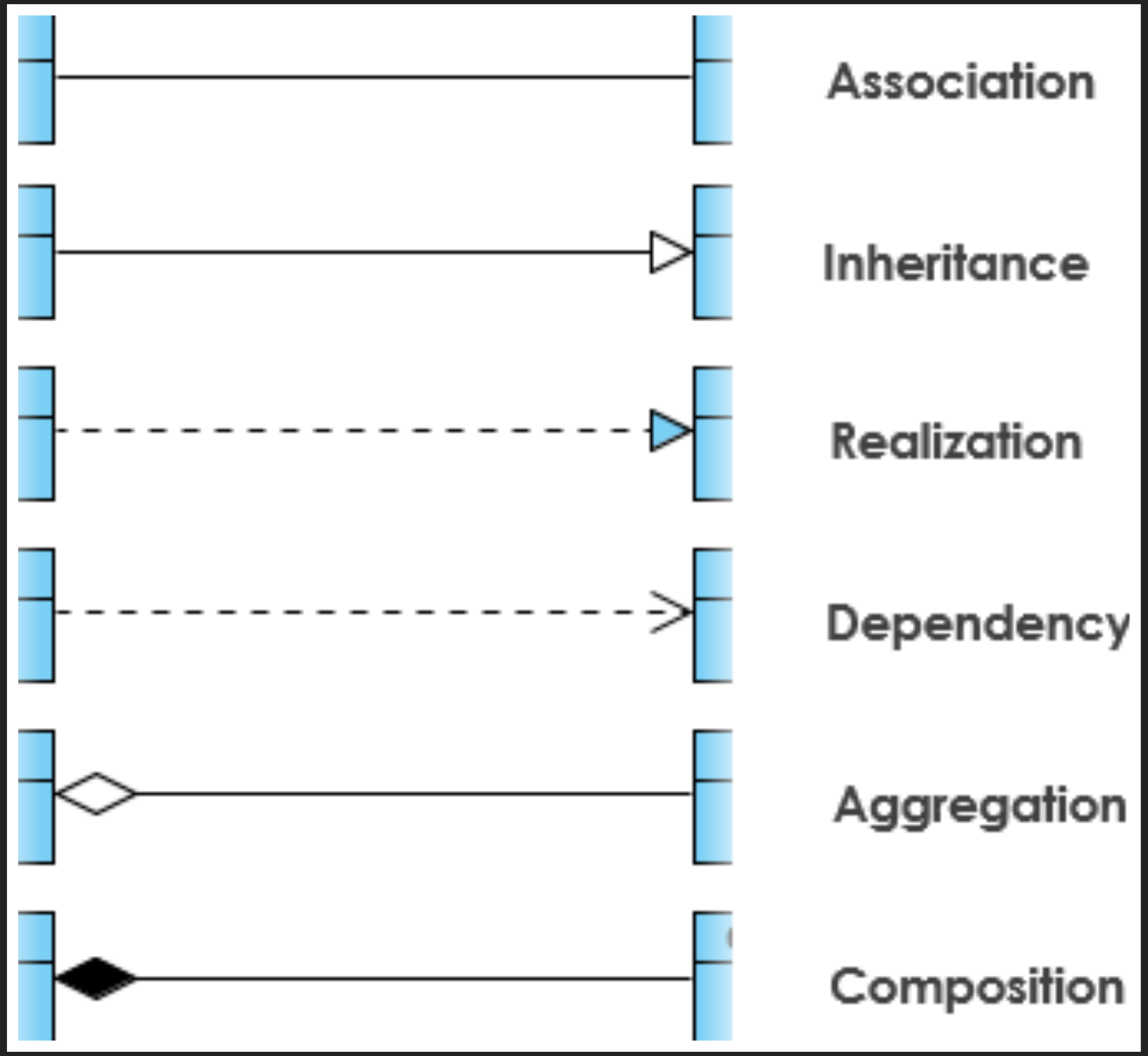


```
Shape
---------------------------------------------
+FACES : int

-sides : int

-color : Color
---------------------------------------------
+Shape(sides : int, color : Color)
+Shape(sides : int)
+Shape()
+getSides() : int
+setColor(color : Color)
+getColor() : Color

+calculateArea(): double
+calculatePerimeter() : double
```

- The class name is italicized because Shape is an abstract class.
- sides and color are prefaced with "-" because they are private.
- FACES is capitalized and underlined because it is final and static.
- calculateArea and calculatePerimeter are italicized because they are abstract methods.

MODELING RELATIONSHIPS

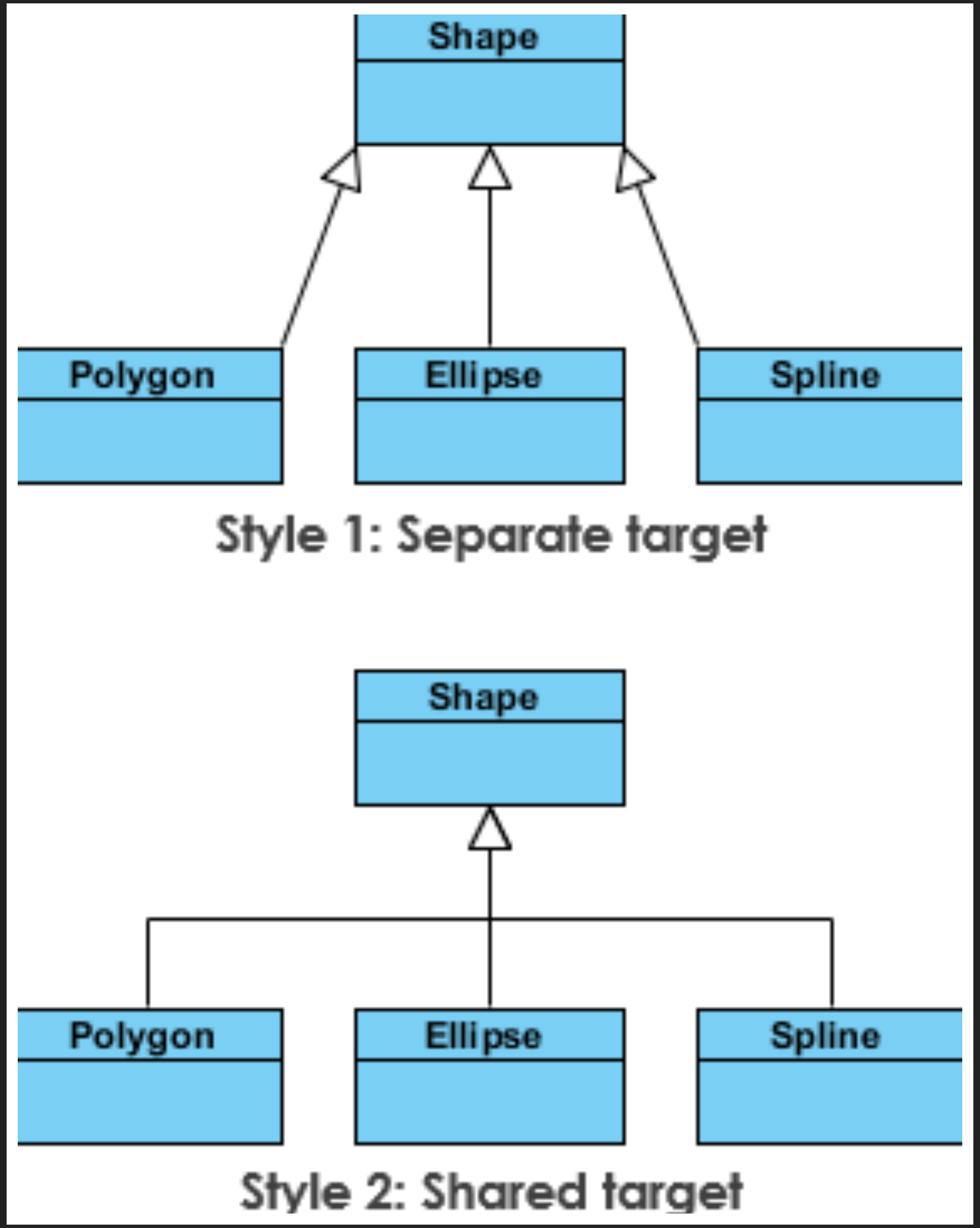
- Items in a class diagram can be related in many different ways.
- There are various relationship-denoting lines we can use to connect classes and/or interfaces.



From Visual Paradigm's UML Class Diagram Tutorial

REVIEW: CLASS INHERITANCE

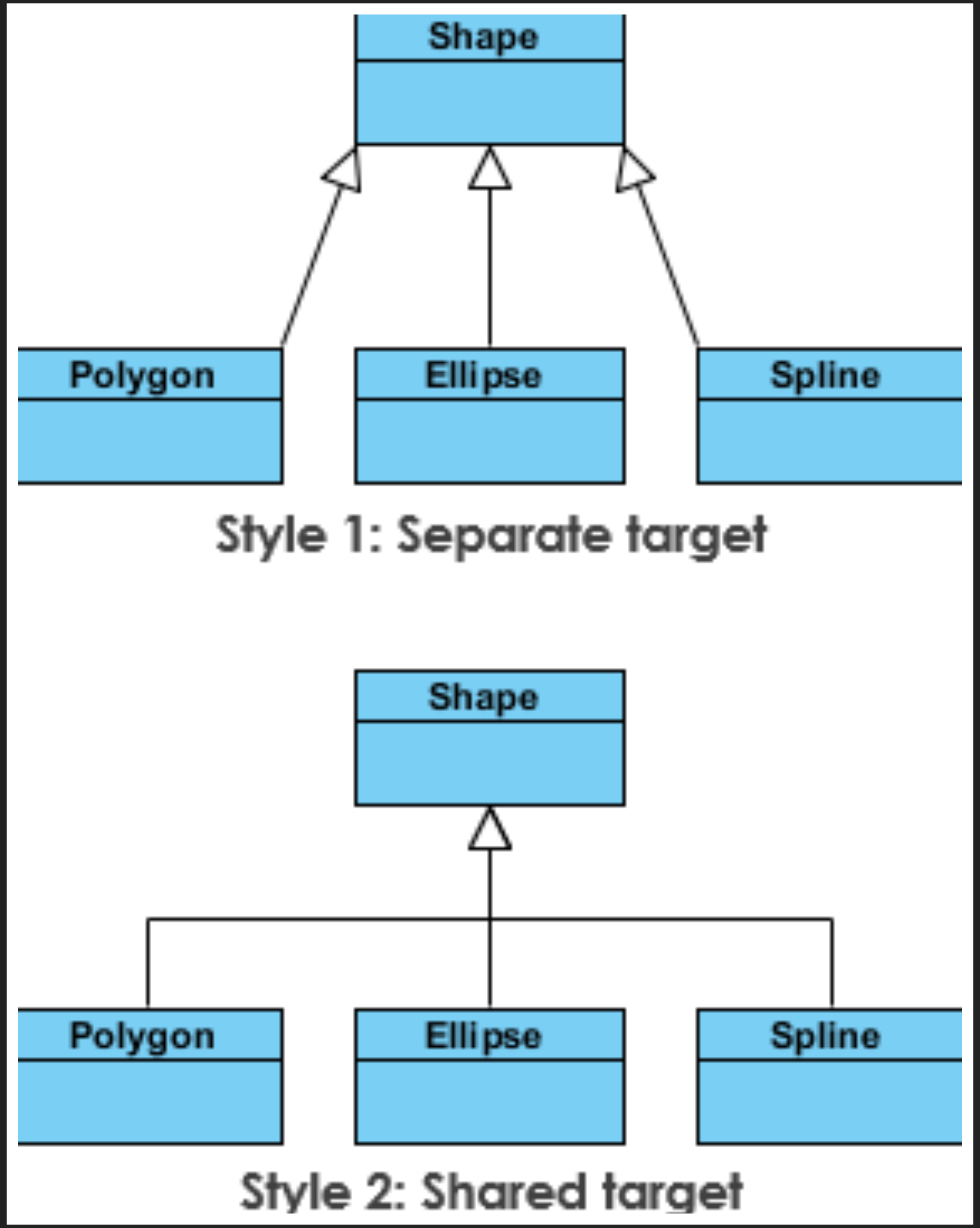
- Inheritance allows us to re-use code by pulling methods and attributes used by multiple classes up in to a generic "Superclass".
- "Subclasses" then extend our superclass, and <u>inherit</u> everything contained within the superclass.



From Visual Paradigm's UML Class Diagram Tutorial

MODELING INHERITANCE

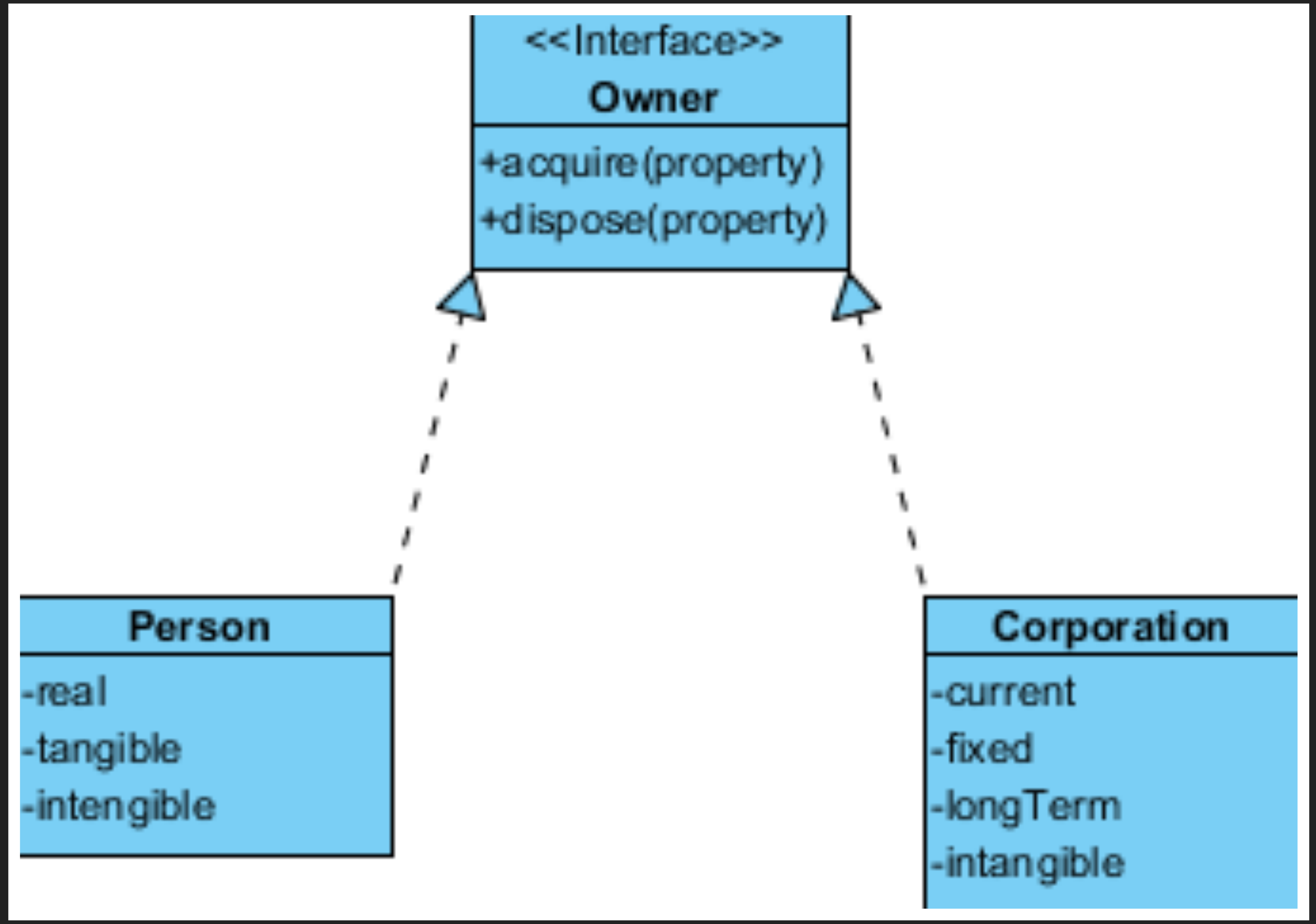
- An inheritance relationship is represented by a solid black line with a white arrow on the end.
- The class that is touching the white arrow is the superclass.
- The class that is touching the opposite side is the subclass.



From Visual Paradigm's UML Class Diagram Tutorial

REVIEW: INTERFACES

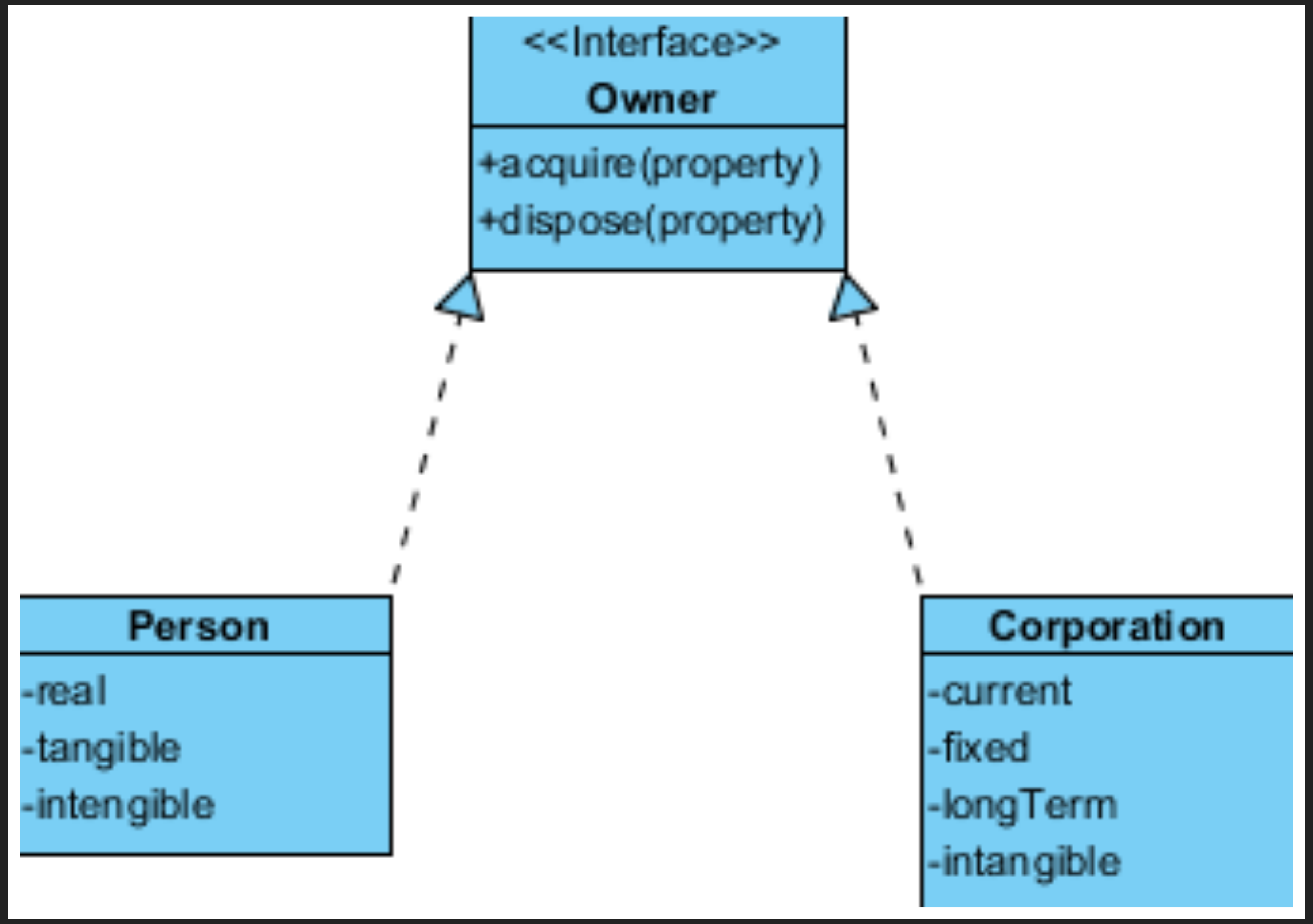
- Interfaces describe the behavior of classes.
- Using interfaces, you could describe that classes are iterable, comparable, or modifiable.
- Interfaces contain no method implementations.
- Everything specified in an interface must be included in classes that implement it.



From Visual Paradigm's UML Class Diagram Tutorial

MODELING INTERFACES

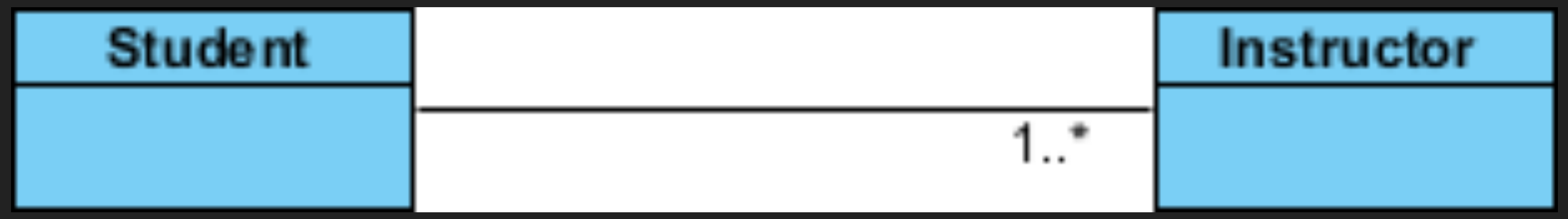
- A realization relationship is represented by a dashed line with a solid arrow on the end.
- The item touching the solid arrow is the interface.
- The item touching the opposite side is the class.



From Visual Paradigm's UML Class Diagram Tutorial

MODELING ASSOCIATION

- Sometimes, two classes in a system may need to communicate with each other.
- This means one class uses the functionalities provided by another class.
- We can say these classes are linked, connected, or associated.
- We represent this with a solid black line connecting the two classes.

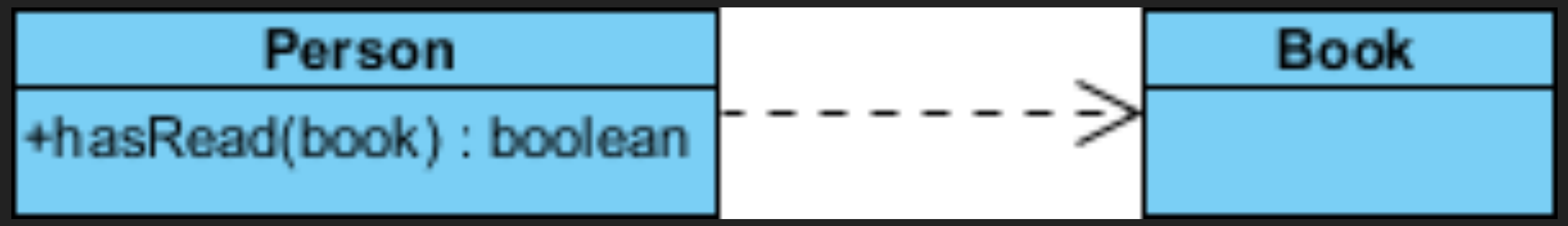


From Visual Paradigm's UML Class Diagram Tutorial

- Here, a student is associated with an instructor.
- The "1..*" Suggests that a student can be associated with one or more instructors.

MODELING CLASS DEPENDENCY

- Sometimes, a method in one class (class A) uses an instance of a different class (class B).
- We say that class A depends on class B.
- This is represented by a dashed line with an open arrow on the end.
- The class touching the open arrow is the depended upon class.

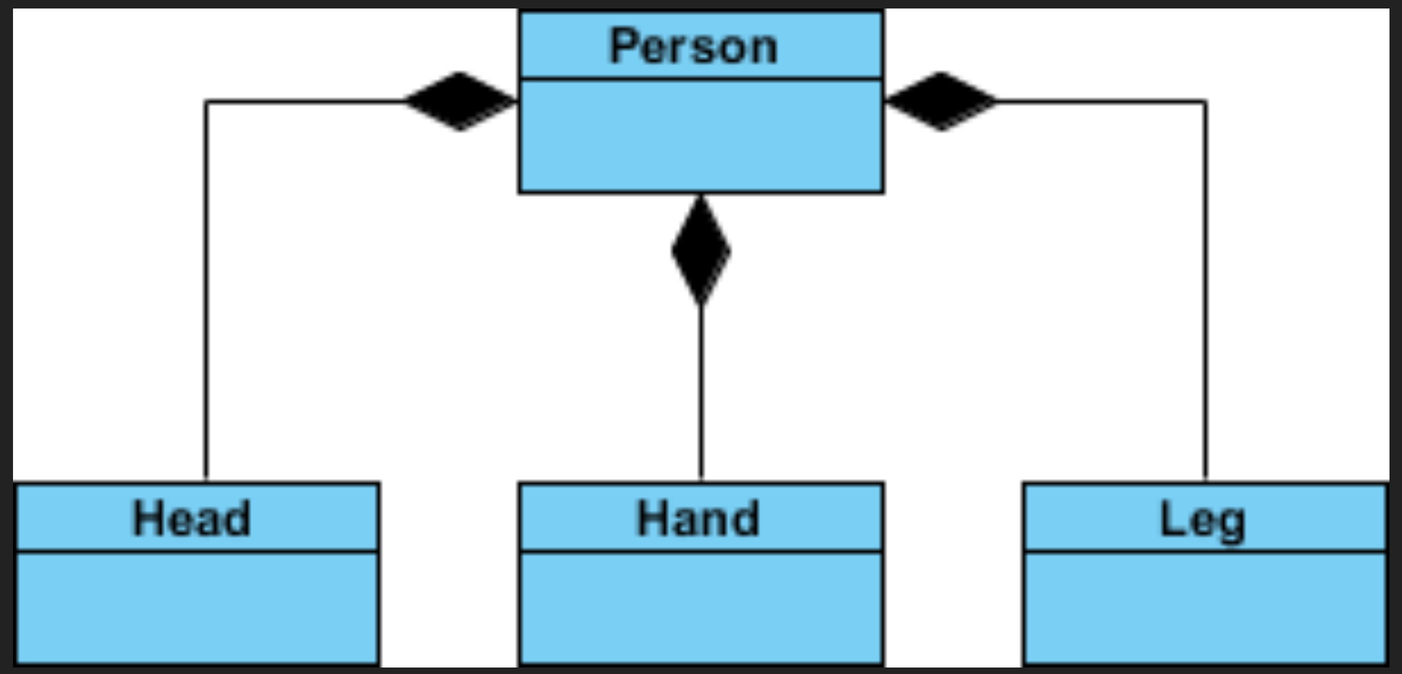


From Visual Paradigm's UML Class Diagram Tutorial

- The Person class has a hasRead method with a Book parameter that returns true if the person has read the book.

MODELING COMPOSITION

- A composition is a specific case of association.
- An instance of one class can "own" an instance of another class.
- The child class in a composition relationship cannot exist independently of the parent class.
- This is represented by a solid line with a solid diamond on the end of it.

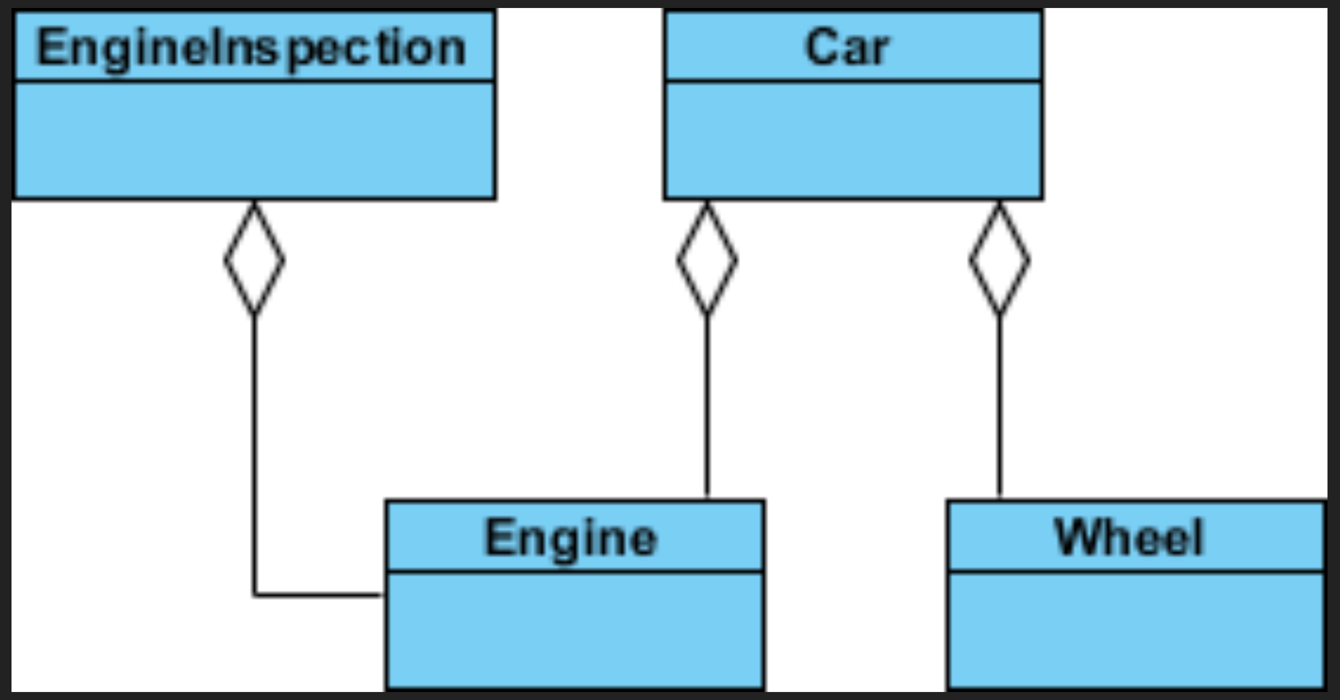


From Visual Paradigm's UML Class Diagram Tutorial

- Hand, Head, and Leg belong to Person.
- If a Person is deleted, then the others are also deleted.

MODELING AGGREGATION

- An association is a specific case of association.
- An instance of one class can "own" an instance of another class.
- The child class in a composition relationship **is able to** exist independently of the parent class.
- This is represented by a solid line with a white diamond on the end of it.



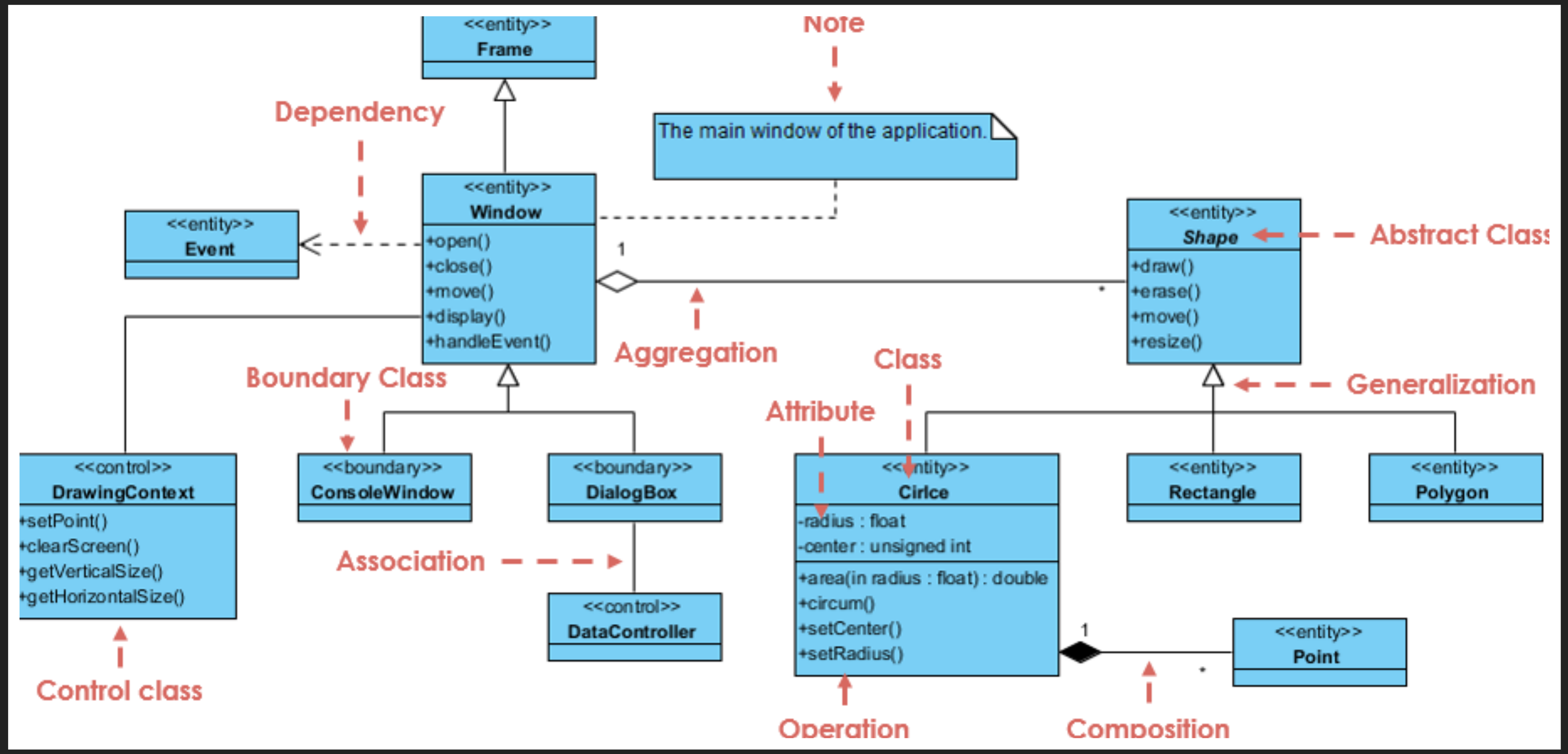
From Visual Paradigm's UML Class Diagram Tutorial

- Engine and Wheel are a part of a Car.
- An Engine can exist independently of a Car.

LET'S SUM IT UP.
WHAT DOES THIS ALL MEAN?

WHY DO WE DO THIS?

- When a UML class diagram is implemented, the resulting code should reflect the intention of the system designer.
- Someone can give us a UML class diagram, and we can implement their system.
- We can make a UML class diagram that helps us explain how our system works.



From Visual Paradigm's UML Class Diagram Tutorial

WHAT IS A UML CLASS DIAGRAM?

Pull up **Socrative.com**

and join room **XTEND159**

WHAT TYPES OF RELATIONSHIPS CAN WE REPRESENT IN A UML CLASS DIAGRAM?

Pull up **Socrative.com**

and join room **XTEND159**

WHY WOULD YOU EVER MAKE A UML CLASS DIAGRAM?

Pull up **Socrative.com**

and join room **XTEND159**