How's your spring break on Wed?

Mentimeter

good

prepare for midterm

a lot of debugging

3

Today's plan

- → Review Ex 12-1
- → Recap question
- → Final project

Ex 12-1: Casting and inheritance

- Object slicing: when treating the derived class object as its based class
- e.g. A* a = &b; where b is of class type B inherited from A
- It's safe to cast derived class objects to their base class. (Upcasting)
- B* b = &a; // compilation error, where a is of class type A
- Downcasting is dangerous, but we may need it

Example of downcasting (**dynamic_cast**)

```
class A {};
class B : public A {};
A* a = new B(); // object slicing
B* b = a; // compilation error
B* b = dynamic_cast<B*>(a);
```

dynamic_cast will help you do the type check
and let you know if you can perform the
downcasting safely at runtime.

Other ways to do downcasting

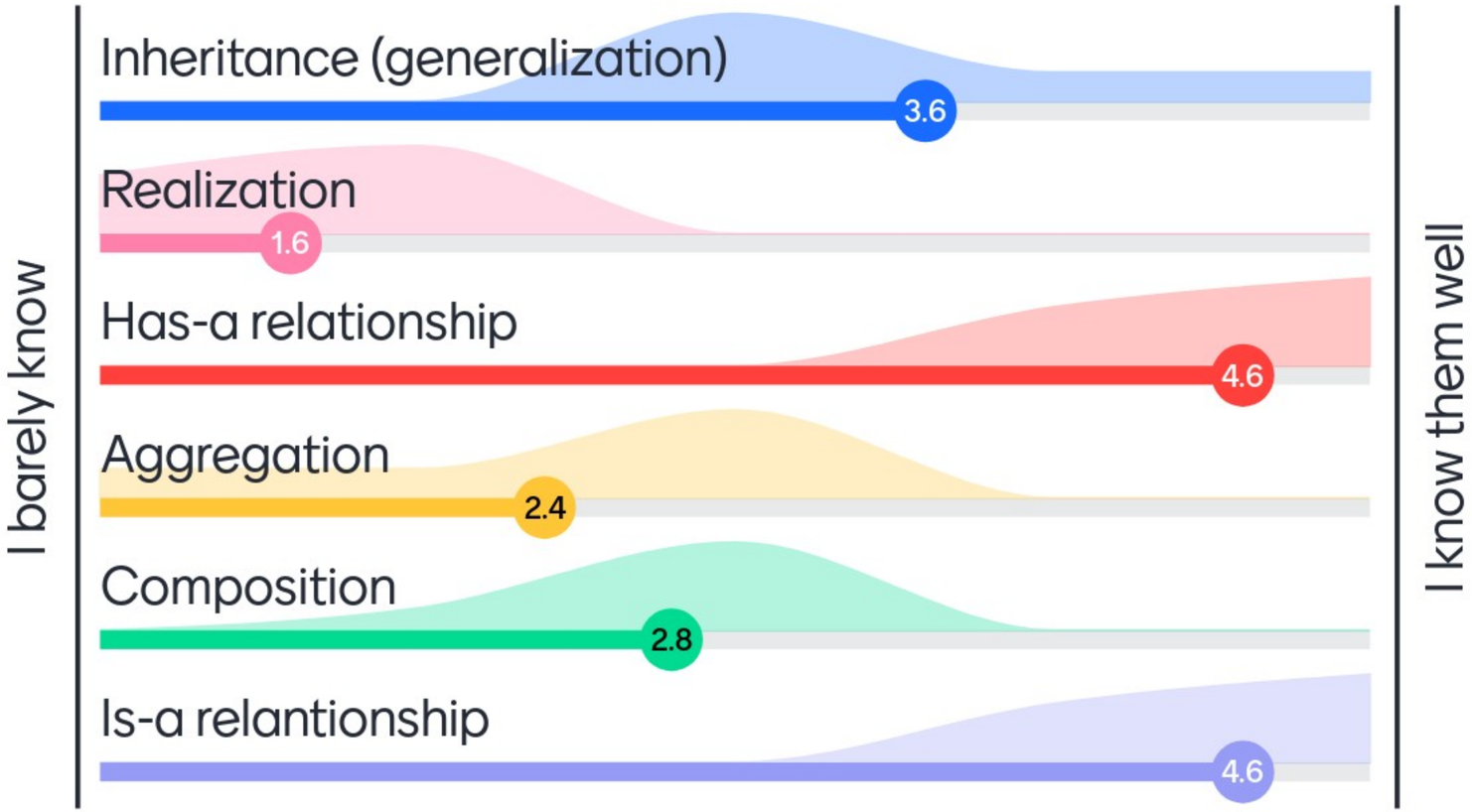
- static_cast
- e.g. B* b = static_cast<B*>(a);
- It will be casted at compile time, no type check is performed
- i.e. it is powerful, but use it at your own risk
- reinterpret_cast
- As the name suggested, you reinterpret the memory. It's the most powerful one, but also the most dangerous one. Not recommended to use
- Only use reinterpret_cast if you know what you are doing

Ex 12-1: **virtual** and *pure* **virtual** functions

- virtual functions (i.e. dynamic dispatch) enables the polymorphism in OO
- To write an interface / abstract class, we define *pure* virtual functions
- e.g. virtual func() = 0;
- The derived classes should provide the implementations; otherwise, it cannot be instantiated.

How much do you know about these UML terms for classes

Mentimeter



5

What is UML?

Mentimeter

way to visualize design of your program

Unified Modeling Language

2x

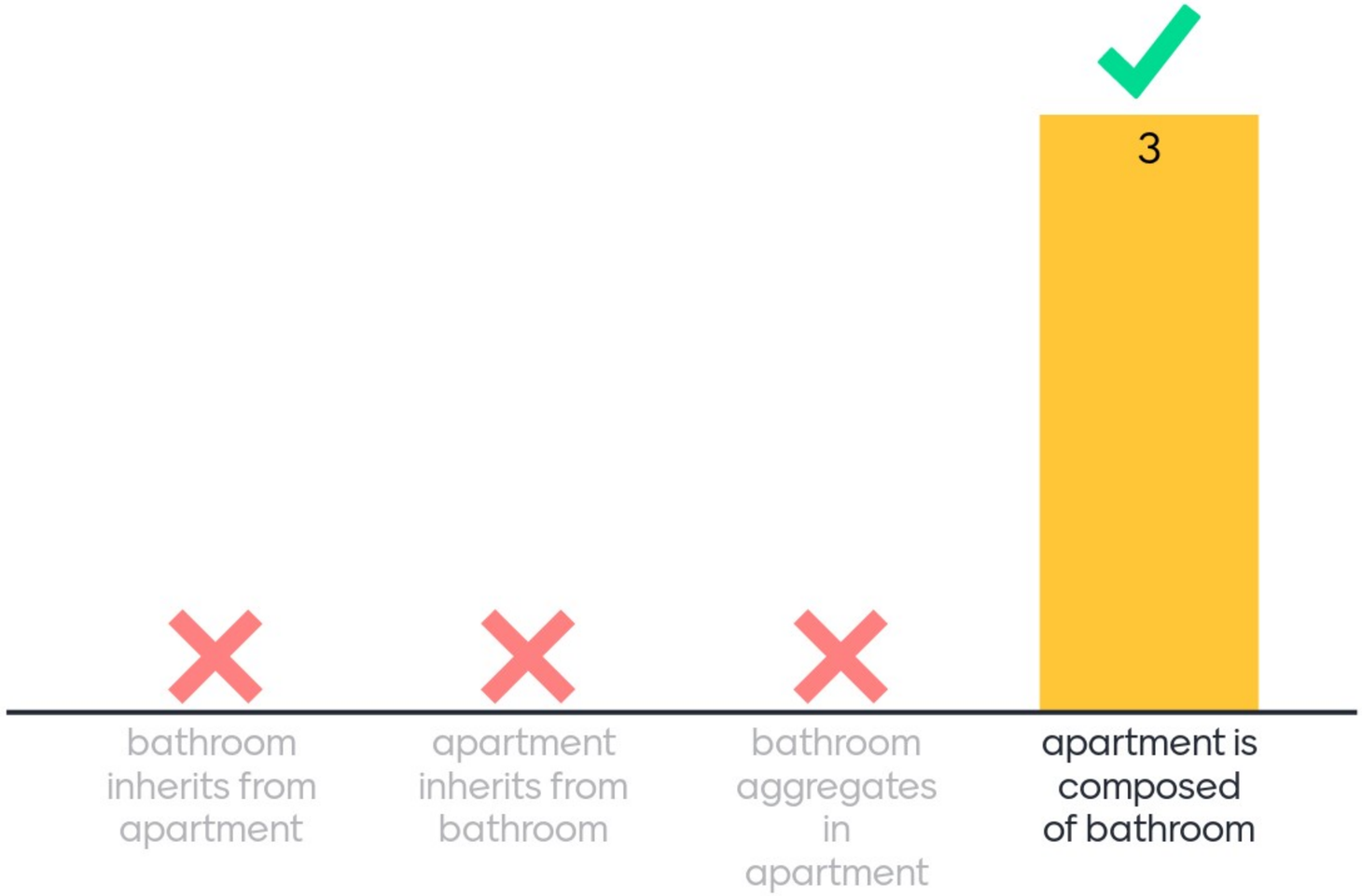
Unified Modeling Language to help you visualize relationships

The correct answer is: Unified Modeling Language - a way to visually represent class diagrams and other software engineering components

4

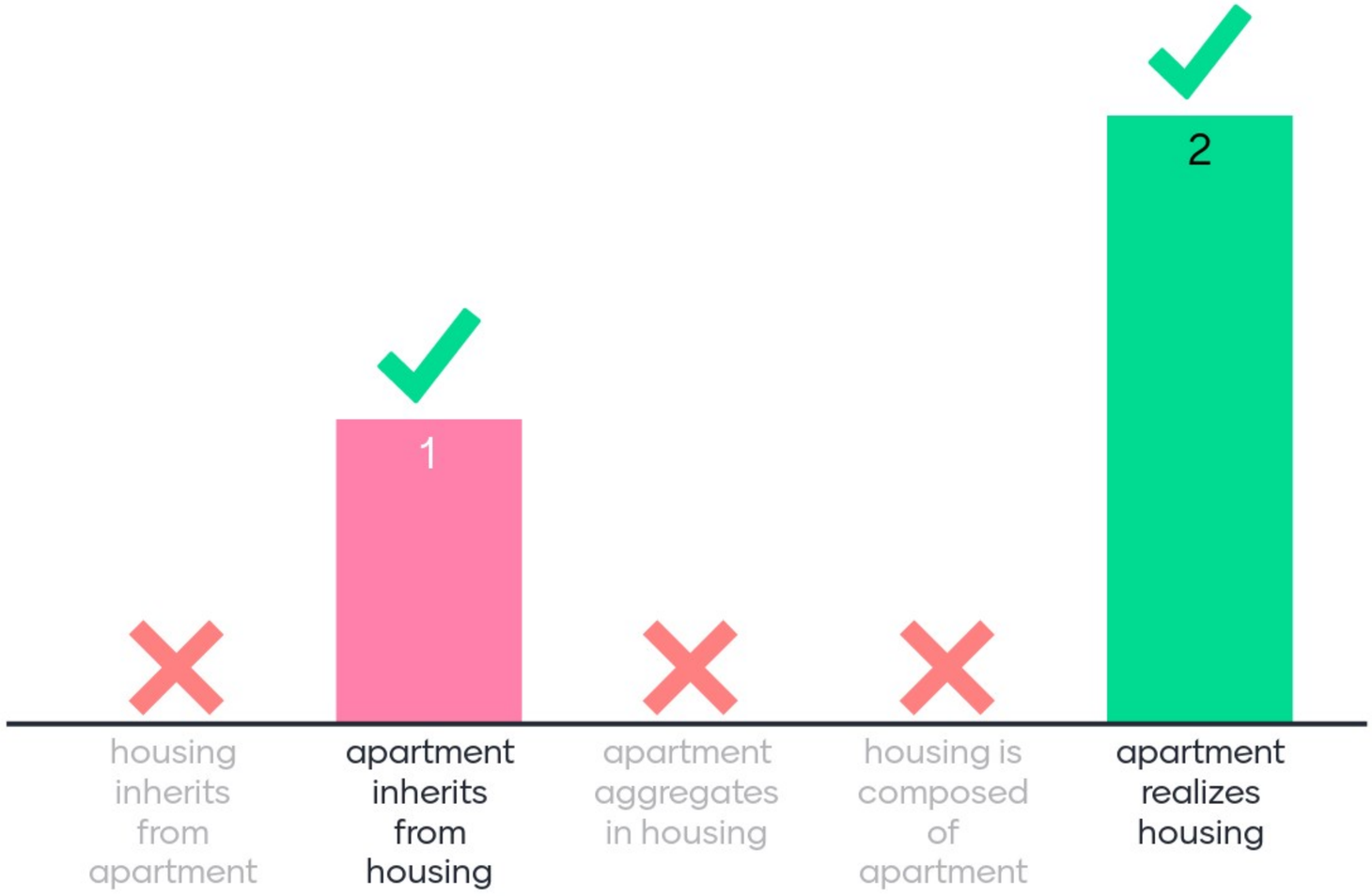
How should you model the relationship between an "apartment" object and a "bathroom object?

Mentimeter



3

How should you model the relationship between an "apartment" object and a "housing" object?



Final Project - Logistic

- Github repo has been assigned if you have a team registered
- Otherwise, you should receive a private Piazza post about the team up
- Deadline: **April 30th at 11pm EDT**
- No late days are allowed!
- Individual contributions submission: **May 2nd at 11pm EDT**

Final Project - Chess (OO design)

- If you don't know how to play chess, check out the chess rules
- Recommend to try it out on https://www.chess.com
- 6 different pieces on the chess board: K/k, Q/q, B/b, N/n, R/r, P/p
- Piece.h is provided as an abstract class
- Your tasks are essentially implementing these 6 pieces and the game rules
- + save/load, and helper functions etc
- You need to use exception handling
- You also need to draw a UML diagram and write a README

Final Project - How to start?

- Clone the starter codes
- Read through the instructions carefully
- Check out the starter codes (in particular the **comments**) and see what you have and what you need to implement (do check out the comments)
- Begin with the UML diagram (as a team)
- You should identify clearly what classes you are going to implement and what are their relationships
- You should also identify what logics/functions should be implemented
- Then, you can use the UML diagram to discuss the workload distribution
- Save your UML diagram for submission later

Final Project - Implementing the 6 pieces

- For each pieces, you must create a cpp file for that
- You may want to take out the [REPLACE THIS STUB] definition and put your implementation in the cpp file
- You need to implement those legal_move_shape pure virtual functions in the derived classes
- You may also need to override legal_capture_shape functions

Final Project - Implementing the Board and Game and Exceptions

- Check out the comments in the header to see what you need to implement for each function
- Board models the chess board that contains all the pieces (std::map< std::pair<char, char>, Piece* >)
- Game models all the game rules
- Each function implements a specific game logic
- Illegal moves should be handled by exceptions. You probably want to throw exceptions in the Game object
- Note: the move command is not the standard chess notation. The move command is only the start and end positions in 4 letters. e.g. E2E4.

Final Project - Using **Terminal.h** for coloring and Mystery Piece

- It contains useful functions for your board display
- We will test your program with our own mystery piece
- However, to test your program (the OO design), you can implement your own mystery piece
- If you do things right, your mystery piece should work fine without modifying the board and the game logic
- e.g. adding another piece with a different move rule and capture rule
- Some testing scenario are provided, e.g. mate_in_two.txt, promotion_to_mate.txt, stalemate.txt etc.

Ask me anything

Mentimeter

0 questions

0 upvotes