CSE 219 MIDTERM REVIEW

with Yvonne The TA
(and some other cool guy)

DISCLAIMER:

I know nothing about your exam. I do not have a copy of it. Professor McKenna has not told me anything. It is against the rules for me to even know about your exam. Even if I did have relevant knowledge, it is even more against the rules for me to tell you. The contents of these slides are based off the public materials given on the website, and my knowledge from TA'ing and taking this course myself. I am not responsible for any potential discrepancies in content, and/or your failure.

But isn't 219 just a #$%@ing huge semester long project?

- No
- Hell no
- Oh god hell no
- NOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOPE
- Yeah, the lectures a kinda really important.

How to ace the test:

- Do your homework. Preferably correctly.
- Know what you are actually doing when writin code.
- Know how to do more of it.

How to get a job:

- Do some projects. Preferably ones that work.
- Know what you are actually doing when writin code.
- Know how to do more of it.
- (Also algorithms and data structures. And schmoozing galore.)

Doing your homework.

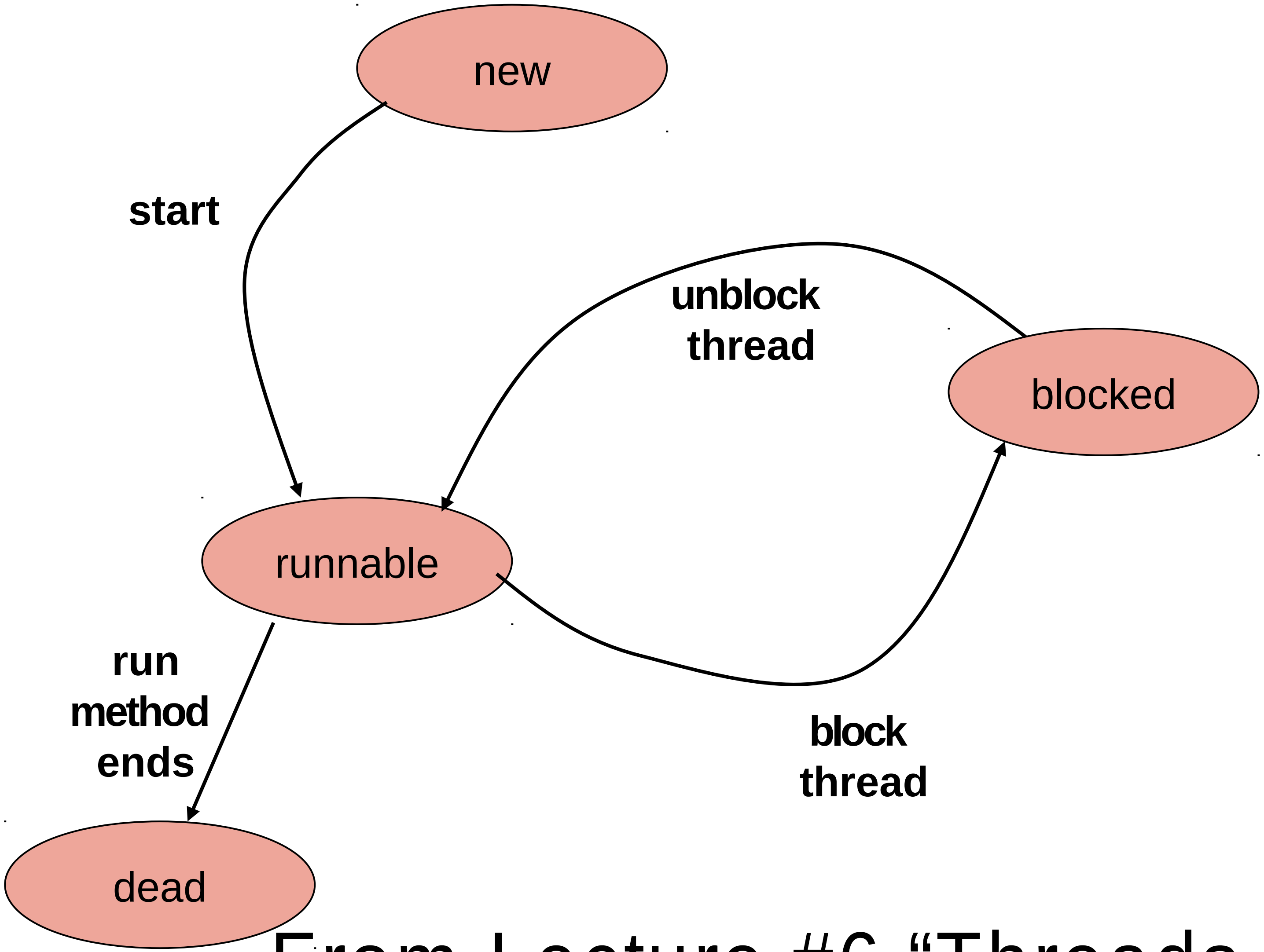
- How does HTML and XML work? What do they stand for?
- How did we read in data?
- What data structures did we use?
- How does MiniGameFramework work?
- Probably two questions about little non-technical details. I'm going to assume it's how the sorting algorithms work.

Knowing what you are actually doing when writing code

- Threads
- You should do some real ***object oriented*** studying to make things a bit less ***abstract***! (haha)
- OOP+ is your #1 study buddy
- A bit of Swing, mostly HW related things!

I know, my sense of humor is really chee

Threads!



From Lecture #6 “Threads & Timer

HW#2 vs HW#3: The threads unravel!

- HW #2 relies on event-based programming. Program updates itself when you press a button through listeners.
- HW #3 uses threading to check for clicks, and subsequent events when mandatory, all the time. (Or at least based on how fast it's running!)
- Learn to love threads! You can't escape them!

Starting Those Threads!

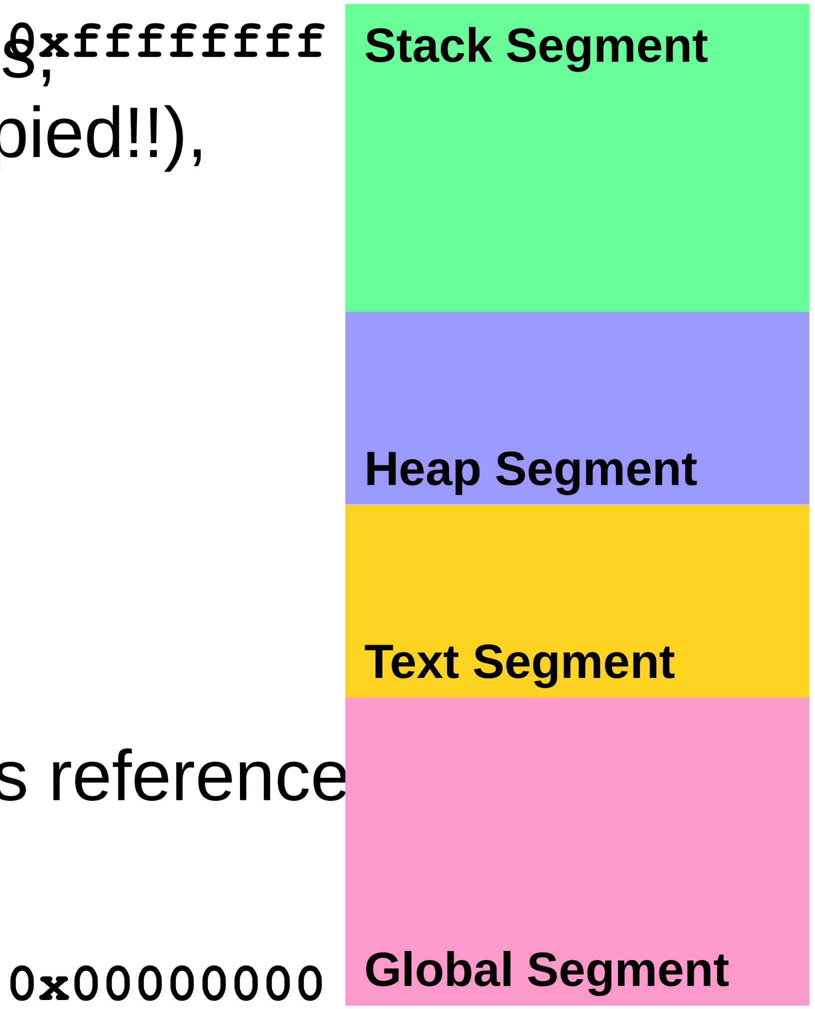
- start() - makes thread runnable, automatically calling run() if using Thread class, you don't ever need to define it yourself
- run() - where stuff actually happens, automatically called if using Thread class but i does NOTHING, if using a class that extends Thread**you must define it**

When good threads go bad.

- Atomicity! (Remember this if you plan on taking 305!)
- Transactions run until completed, or not at all.
- If you are using an object for multiple simultaneous transactions, you are looking for trouble.
- Use locks! Specifically ReentrantLock in a try, finally block. (Unlock in the finally!!)

Memory Management

- Savor the simplicity, you get into the nitty gritty in 220.
- **Stack**: methods! Method variables, method arguments (which are copied!!), removed when returned
- **Heap**: reserved at compile time (static data!)
- **Text**: program instructions
- **Global**: dynamic and persistent data for objects. Created when a new object is made, lasts as it is reference



Inheritance

- Person p = new Student(); // okay!
- Student s = new Person(); // not okay!
- **Label** l = new **Box**();
- Everything must fit into the box!
- Objects can only be cast to an ancestor!

Collections

- Really cool lists that use abstraction to make our lives easier
- A list of specific objects! Catches compiler errors over run-time! No need to do so much object casting!
- Comparable vs Comparator: Comparable sorts with natural ordering, comparator with custom

Apparent vs. Actual Types

- **Apparent**:
  the type an object is **declared** as
  only the **compiler** cares
  determines what methods can be called
  the “**label**”
- **Actual**
  the type an object is **constructed** as
  only the **JVM** cares
  determines where the implementation of a method is defined
  the “**box**”

Static vs Non-Static

- Static methods/variables:
  scoped to a class
- Non-static methods/variables:
  scoped to a single object
- Like socks and towels in a hot dryer, static methods/variables are really sticky
- Data is stored on the **heap**, not copied to the **stack**, like non-static

Abstract & Interface Classes

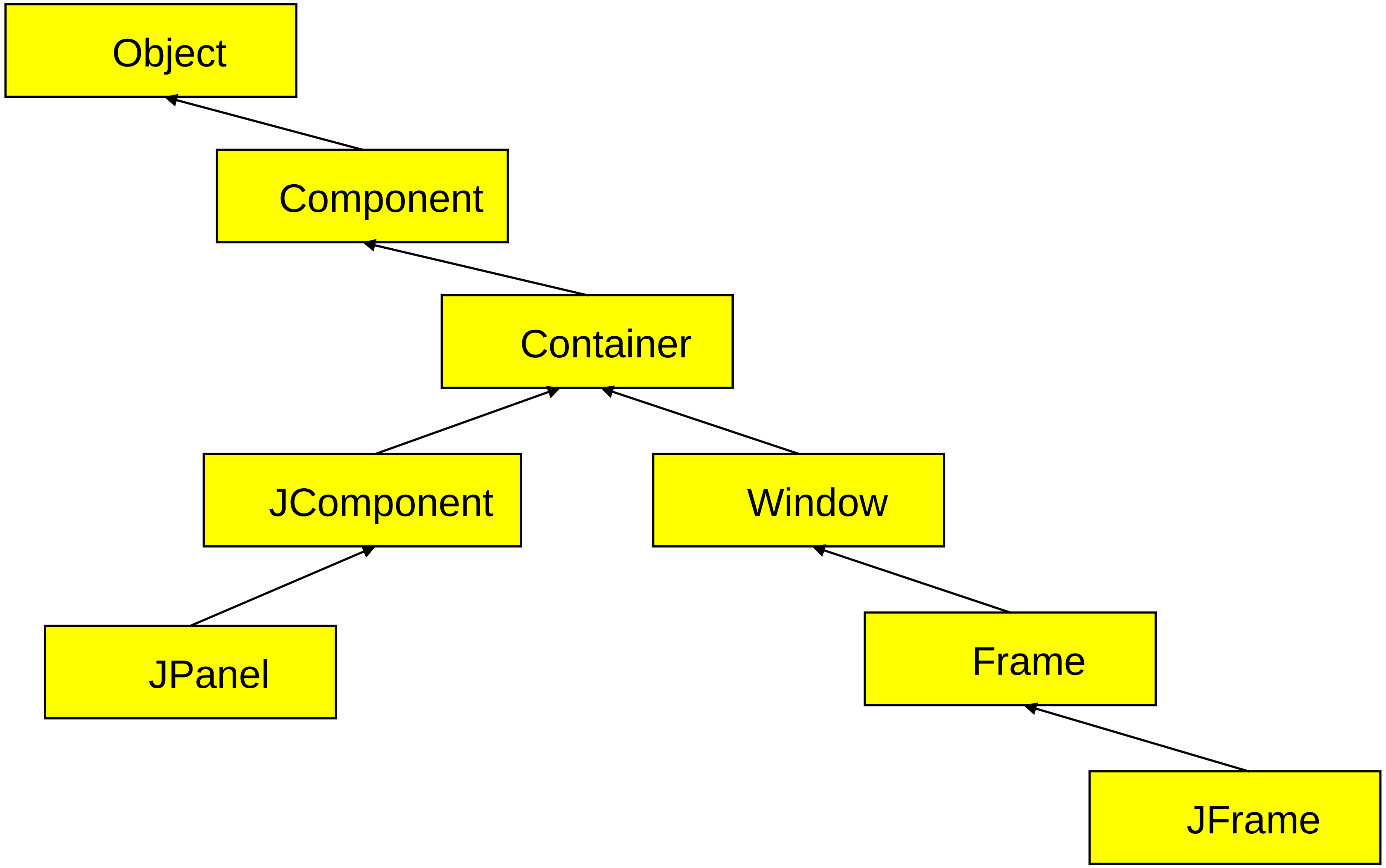
- Abstract Classes:
  can specify abstract and concrete methods
- Interfaces:
  specifies abstract methods

  method headers with no bodies!
  e.g. ActionListener
- Objects **can have apparent type** of concrete/interface/abstract, **can never have actual type** of interface or abstract

Swing! Swing! Swing!

- Event-based GUI interface
- Already made pretty by Java automatically
- Forms to fill out! Buttons to click! Events to register! Data to send! Update it all, and then start again!
- Stick to focusing on things used in HW/lecture Also, layouts!

Hierarchies and Inheritance



JPanel – Where the action's at!

- Look at your homeworks, there's a class that extends Jpanel!
- Do not call paintComponent(Graphics g). Java does that for you automatically.
- Instead, **repaint()**
- Graphics define shape and texts to render on this panel
- Adjust with **mouse-based events**

Knowing how to design more software.

- Version control! What is it, and why do we use it?
- Debugging! (You'll get the hang of it eventuall
- Software development lifecycle
- UML and design organization

IMPORTANT

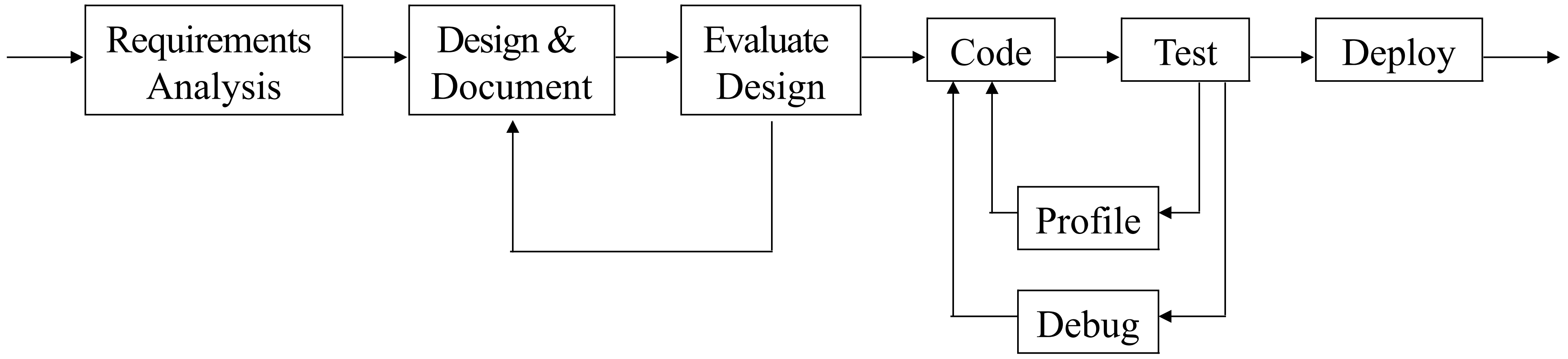
First, define the problem

Design, then code

VERY IMPORTANT

First, define the problem	**First, define the problem**	**First, define the problem**
Design, then code	**Design, then code**	**Design, then code**
First, define the problem	First, define the problem	First, define the problem
Design, then code	Design, then code	Design, then code
First, define the problem	**First, define the problem**	**First, define the problem**
Design, then code	**Design, then code**	**Design, then code**
First, define the problem	First, define the problem	First, define the problem
Design, then code	Design, then code	Design, then code
First, define the problem	**First, define the problem**	**First, define the problem**
Design, then code	**Design, then code**	**Design, then code**
First, define the problem	First, define the problem	First, define the problem
Design, then code	Design, then code	Design, then code
First, define the problem	**First, define the problem**	**First, define the problem**
Design, then code	**Design, then code**	**Design, then code**

Software Development Lifecycle



From Lecture #1 Software Development Lifecy

Waterfall Model

Requirements Analysis

Design

Evaluate Design

Code

Test, Debug, & Profile Components

Integrate

Test, Debug, & Profile Whole

Deploy

Maintain

I wish we could learn Agile though. It's pretty popular and sounds cool.. ;_

Properties of High Quality Software

- Correctness
- Efficiency
- Ease of use
  - for the user
  - for other programmers using your framework
- Reliability/robustness
- Reusability
- Extensibility
- Scalability
- Maintainability, Readability, Modifiability, Testability, etc

Software Design!

- Data-driven design, make sentences!: nouns (objects, attributes of objects), verbs (methods)
- Top-down design: Divide your problems, make a skeleton outline, and conquer!
- K.I.S.S. Keep It Simple, Stupid
- D.R.Y. Don't Repeat Yourself
- Modularity!
- In practice: GameStateManager (data manipulation) and UI (the things you see and interact with) are separate! Why?

UML

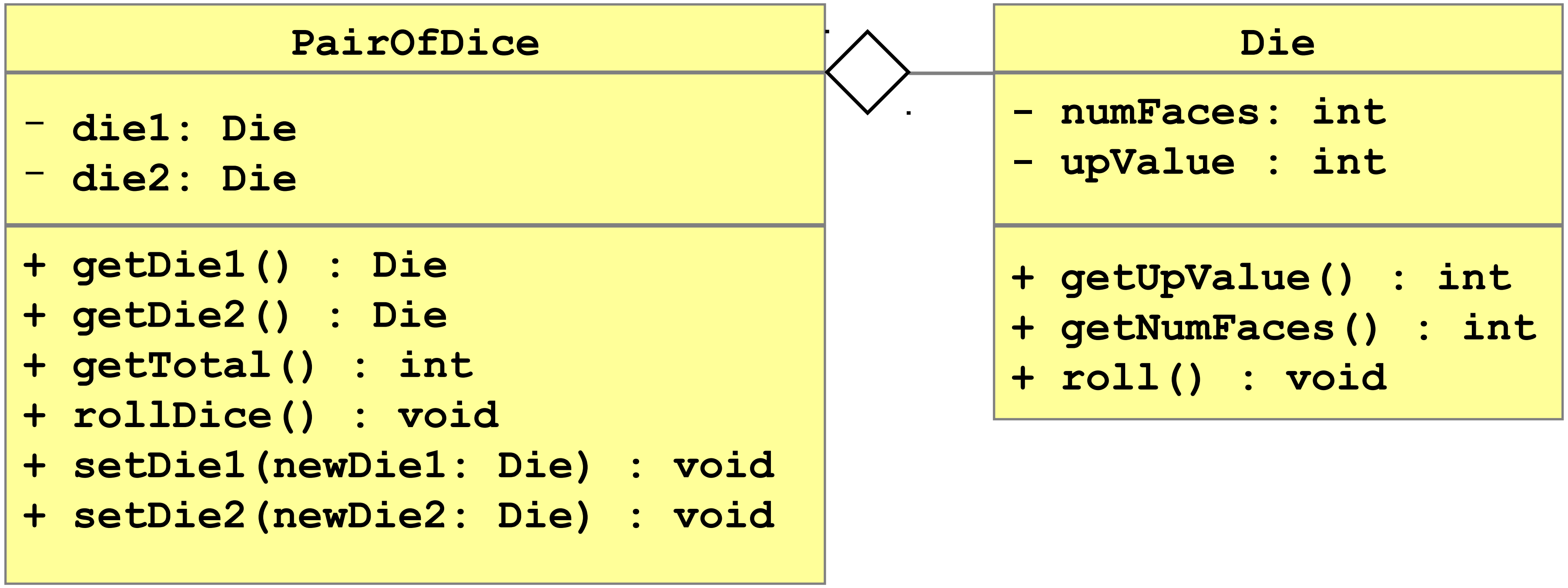
- Unified Modeling Language
- Organize your plans!
- The more you plan now, the more you can sleep later. The better you plan now, the better you'll sleep later.
- Use case diagram: describe all the ways users will interact with the program
- Class diagram: describe all of our classes for our app
- Sequence diagram: describe all event handling

Don't worry, I find it tedious too...

Class diagram

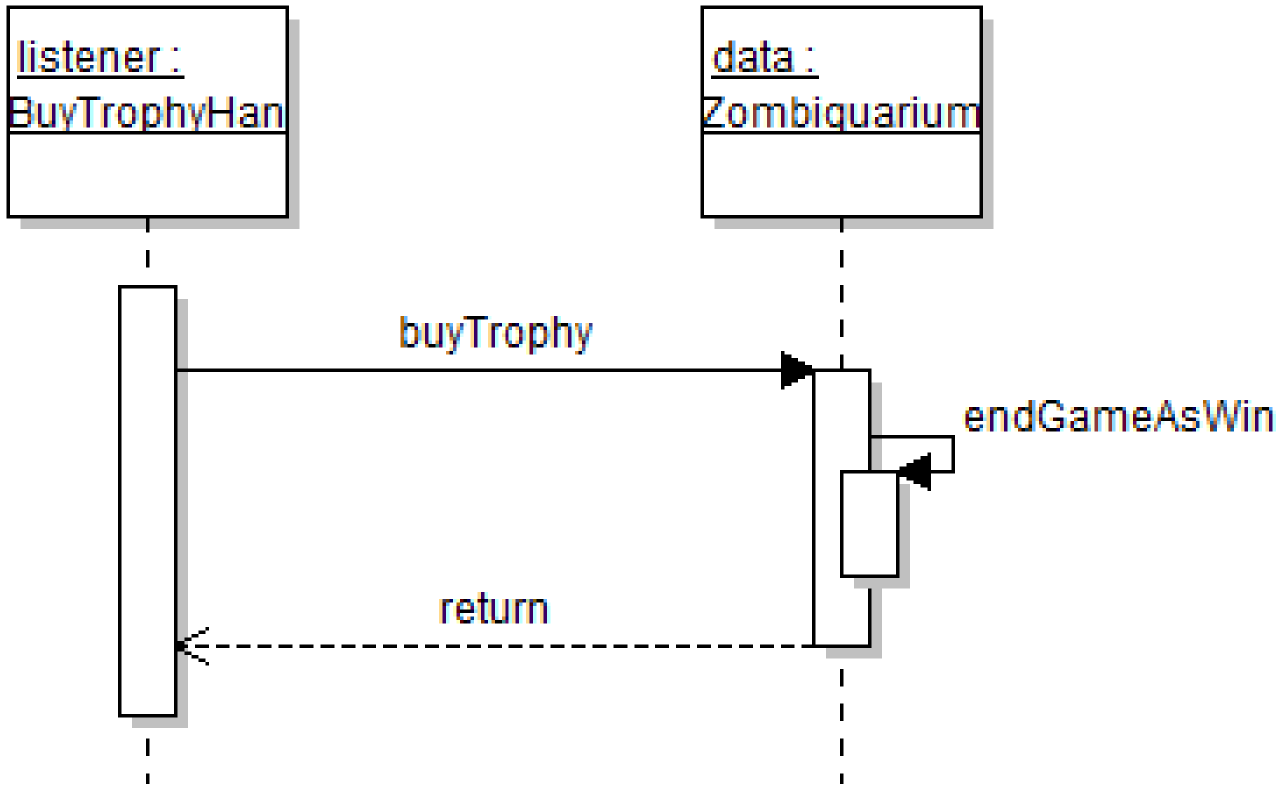
- Makes outlines of classes.
- Memorize the formatting: public vs private variables, HAS-A (diamond) vs IS-A (arrow).
- Remember symbol position. (DicePair <>--Die Person<--Student)
- <<interfaces>>, or <<I>>
- *AbstractClass*, or {abstract}

Students have swag. Professionals have class diagrams.



Sequence Diagrams

- Demonstrate the behavior of objects in program
  - describe the objects and the messages they pass
  - diagrams are read left to right and descending



Version Control

- IS A LOT MORE AWESOME AND AMAZING THAN YOU THINK.
- Case in point: GitHub
- (employers love it)
- (you might even find cool things on it too)

https://github.com/yvds/219s14-ReviewSessi

You're welcome, and good luck!! :).