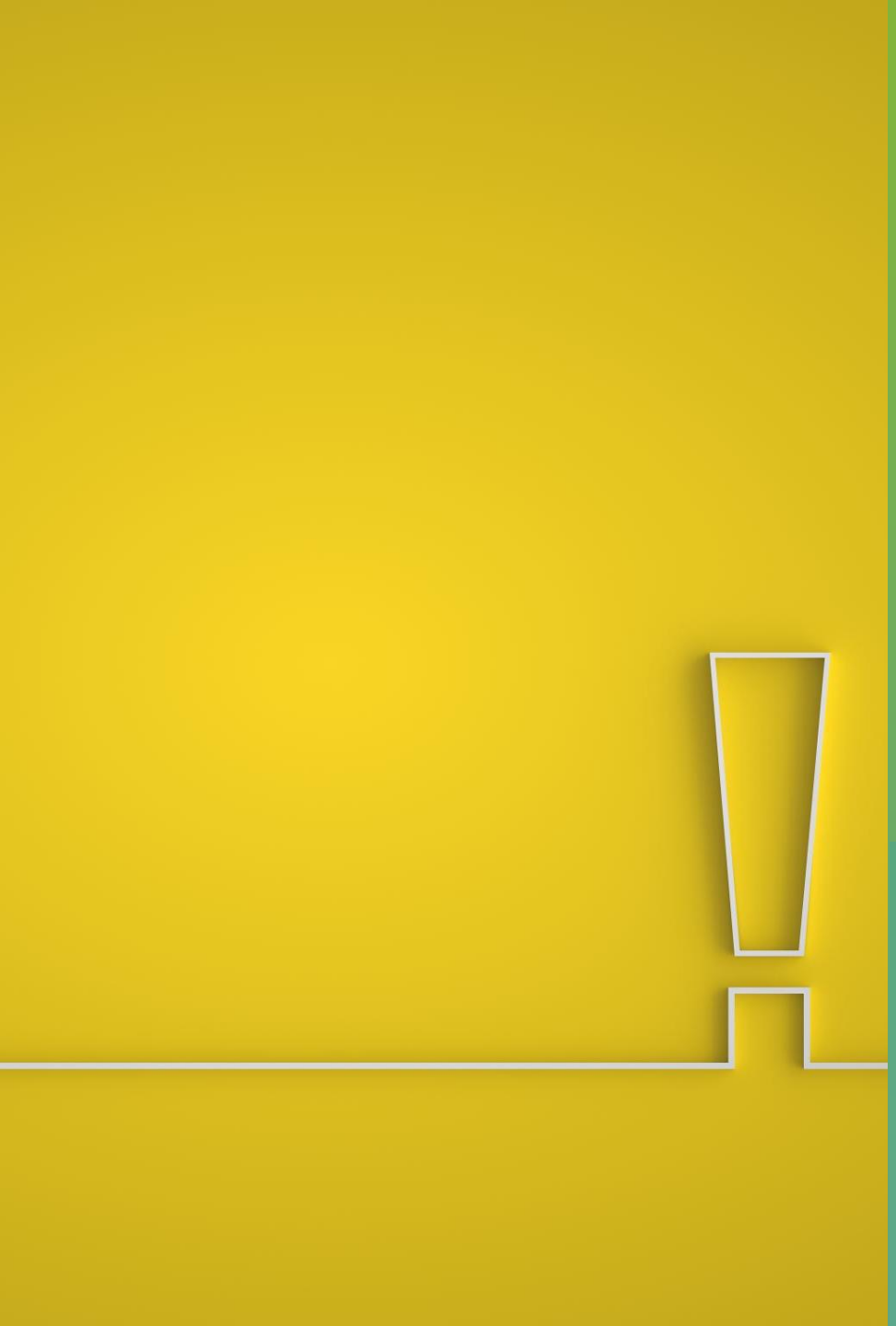


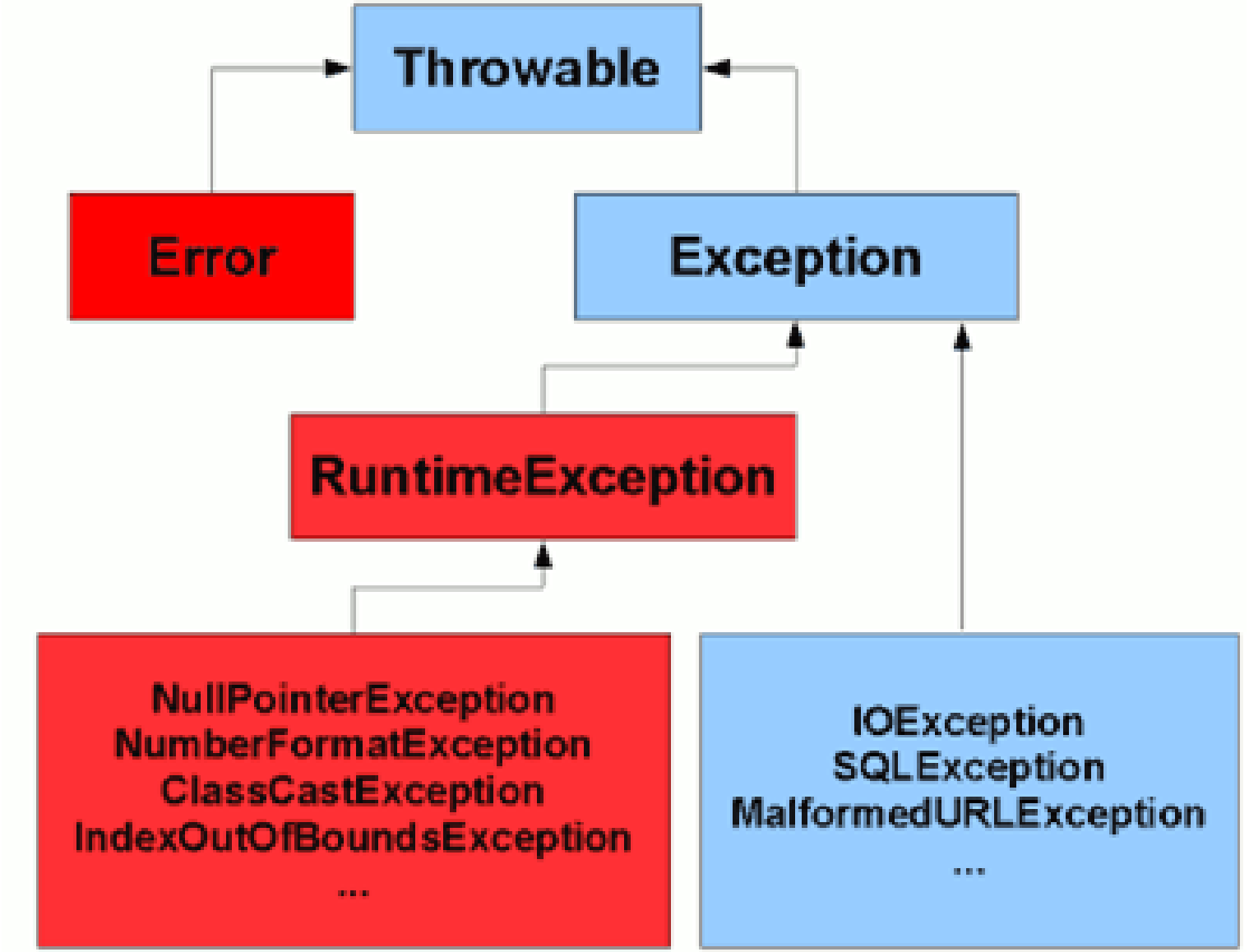
EXCEPTION HANDLING

EXPECTING THE UNEXPECTED

EXCEPTIONS

- An *exception* is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
- Could be a bad (if you're demoing something to important people) or good (if you're developing and find out a use case you haven't handled)
- Note: Exceptions are different from errors. Errors indicate serious problems that an application shouldn't try to catch. They're events that are fatal to the program runtime. They're not something that can be handled. Like a stack overflow, its something out of the developer's hands.

EXCEPTION HIERARCHY



CHECKED VS UNCHECKED

- *Checked* exceptions are exceptions that java forces you to handle.
- *Unchecked* exceptions are subclasses of Runtime Exception. A method is not forced by the compiler to declare the unchecked exceptions thrown by its implementation.

THROWING EXCEPTIONS

- Obviously, exceptions tell you when something is wrong
  - You'd want to know why your program isn't working
- Exceptions are also good for enforcing logic/rules in your program
  - In fact, you can make custom exceptions for this purpose!
- You throw exceptions using the throw keyword

HANDLING EXCEPTIONS

- Exceptions are handled using a try-catch block
- You wrap the risky (might throw an exception) code in a try block
- Handle any exceptions that have occurred in a catch block
  - Note that order matters! Catch the more specific exceptions first then the more general ones
- Clean up any open resources using a finally block
  - Or run any code that should always run with or without exception

QUESTIONS?

COMMENTS? CONCERNS? VIOLENT REACTIONS?

DISCUSSION QS

- What are exceptions? How are they different from errors?
- Why handle exceptions?
- How do you handle exceptions?
- Why throw exceptions?
- What is the exception hierarchy?