1. Visitor design pattern:
   1. Personally I think this is the most tricky one out of the 23 design patterns
   2. Say we have: Parent arr[] = {new Child1(), new Child2()}
   3. When we traverse `arr`, if we see Child1, we want to do something; if we see Child2, we want to do something else.
   4. How to achieve that without using instanceof?
   5. A naive thought: use overloading functions, so define
      1. DoSomething (Child1 c) {...} and DoSomething(Child2 c) {..}
      2. Then call DoSomething (arr[i]) – does this actually work? Why not?
   6. okay, we have to use polymorphism
      1. Polymorphism works when you call an overridden method on the target object. In our case, ideally we wish we could do something like arr[0].DoSomething()
      2. But class Child does not have the DoSomething() method defined (for obvious reason that the author of the Child class has no way to know the “something” you want to do)
   7. Visitor design pattern: a combination of bullet e and f.
      1. See the lecture code. Hard to explain without code..
      2. The technique of this combination has a name: double dispatch: <https://en.wikipedia.org/wiki/Double_dispatch>
         1. Remember dynamic dispatch?
2. Less interesting design patterns
   1. Lazy initialization: defer the initialization of individual objects in a container
      1. Example: visit birds in a (static) conservatory
   2. Observer design patterns:
      1. Register a list of callback functions to event sources
      2. Typically used for event-driven programming
   3. Facade design patterns
3. A quick word about other design patterns
4. Concurrency design patterns
   1. Why multithreading?
      1. Multiple CPUs/cores
         1. E.g., speeding up mergesort a huge array
      2. Useful even on a single core – blocking I/O
         1. E.g., how to use console input to terminate a program
   2. Threads
      1. Starting a thread to something else concurrently
         1. <https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html>
      2. A thread is a unit of concurrency
      3. Let’s refine our singleton design pattern example: I have two pages and 10 clicks on each page happening at the same time
      4. Does the singleton counter get incremented to 20? Why not?
   3. Synchronization
      1. Lock design pattern
         1. <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html>
         2. Usually lock the data
      2. Sometimes we want to “coordination” rather than mutual exclusion
         1. wait/notify – real time analogy
         2. Java Condition
      3. Monitor design pattern
         1. A combination of locking and condition
         2. <https://en.wikipedia.org/wiki/Monitor_(synchronization)>
   4. Alternative way to implement a monitor in Java
      1. Synchronized keyword
      2. Every object in Java is inherently a Monitor – back to the Object class
         1. Wait() and notify()
   5. Implement a bounded blocking queue
      1. <https://leetcode.com/problems/design-bounded-blocking-queue/>
      2. The built-in data structures like linked list, queue, etc are not thread-safe
      3. Let’s beef up our hand-rolled linked list to make it a bounded blocking queue
      4. Java built-in thread-safe data structures: LinkedBlockingQueue