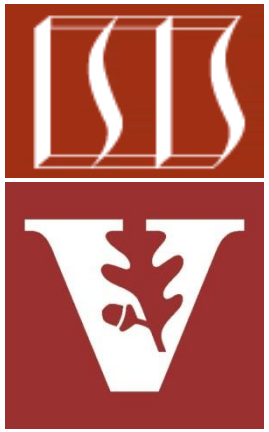
Introduction to Java
Monitor Objects

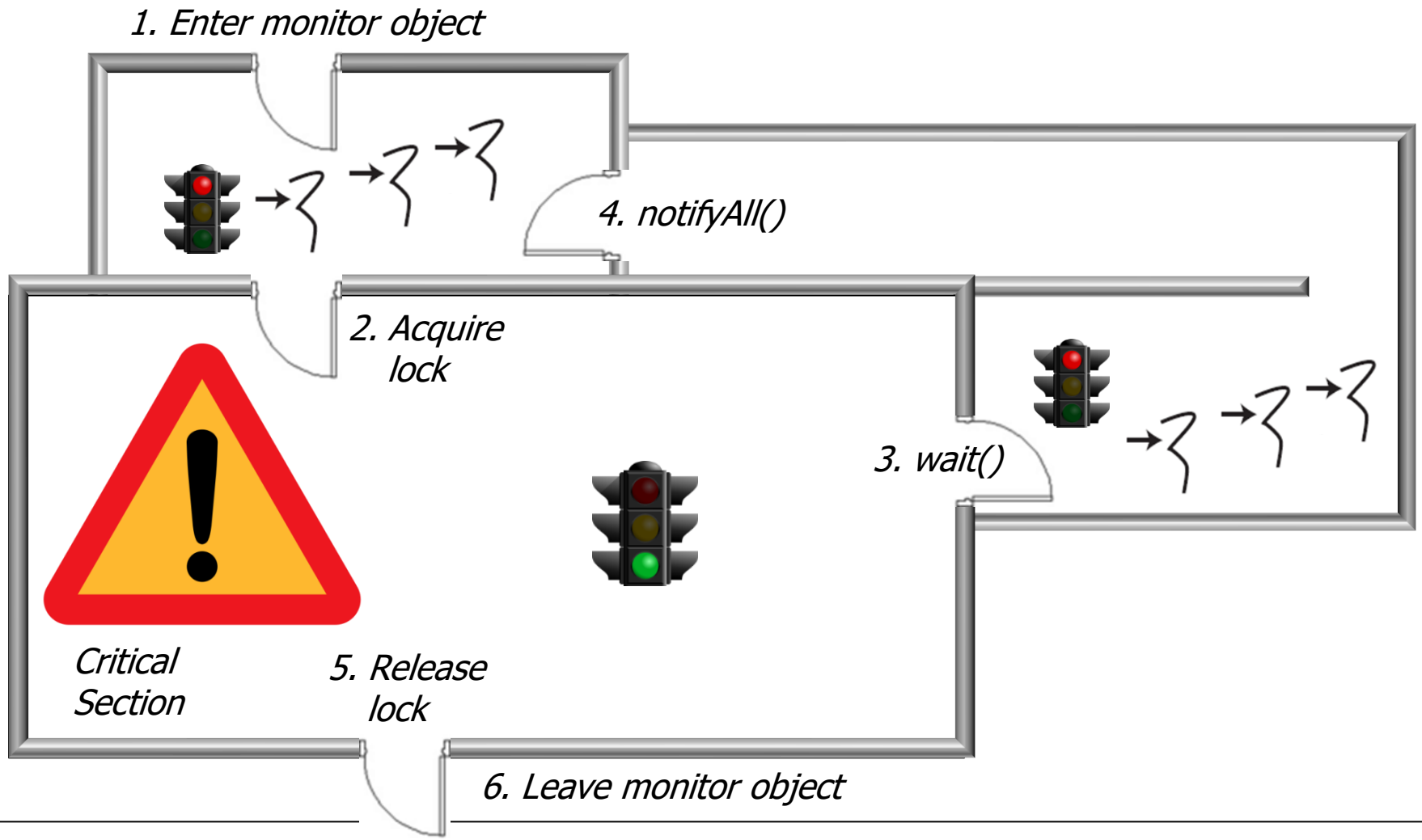


Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

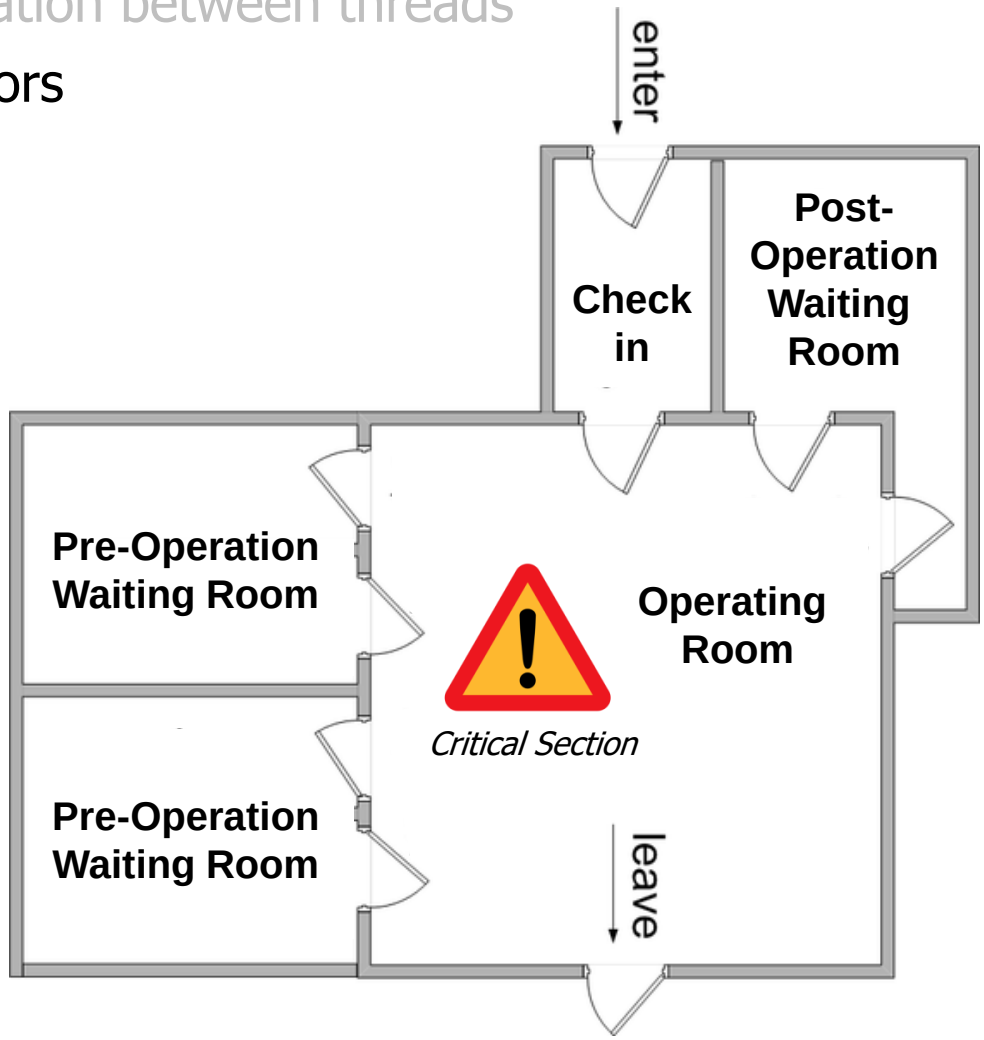
- Understand what monitors are & know how Java built-in monitor objects can ensure mutual exclusion & coordination between threads



See www.artima.com/insidejvm/ed2/threadsynch.html

Learning Objectives in this Part of the Lesson

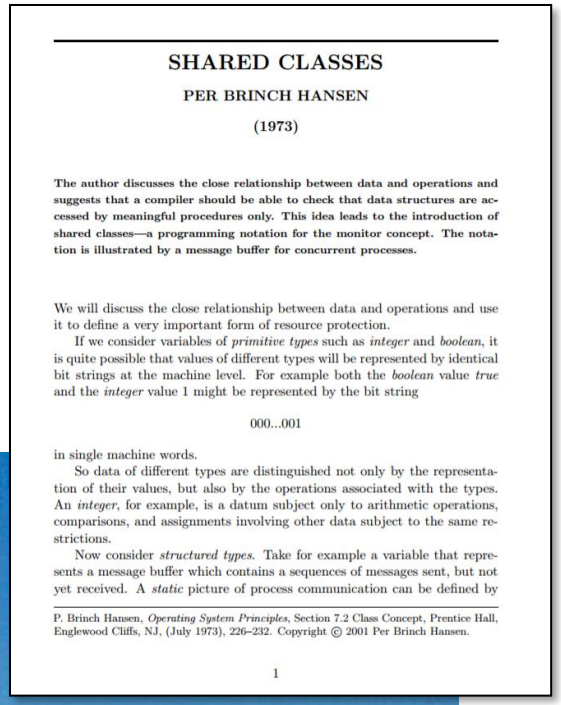
- Understand what monitors are & know how Java built-in monitor objects can ensure mutual exclusion & coordination between threads
- Note a human known use of monitors



Overview of Monitors

Overview of Monitors

- A monitor is a synchronization mechanism designed in the early 1970s



SHARED CLASSES

PER BRINCH HANSEN

(1973)

The author discusses the close relationship between data and operations and suggests that a compiler should be able to check that data structures are accessed by meaningful procedures only. This idea leads to the introduction of shared classes—a programming notation for the monitor concept. The notation is illustrated by a message buffer for concurrent processes.

We will discuss the close relationship between data and operations and use it to define a very important form of resource protection.

If we consider variables of *primitive types* such as *integer* and *boolean*, it is quite possible that values of different types will be represented by identical bit strings at the machine level. For example both the *boolean* value *true* and the *integer* value 1 might be represented by the bit string

000...001

in single machine words.

So data of different types are distinguished not only by the representation of their values, but also by the operations associated with the types. An *integer*, for example, is a datum subject only to arithmetic operations, comparisons, and assignments involving other data subject to the same restrictions.

Now consider *structured types*. Take for example a variable that represents a message buffer which contains a sequences of messages sent, but not yet received. A *static* picture of process communication can be defined by

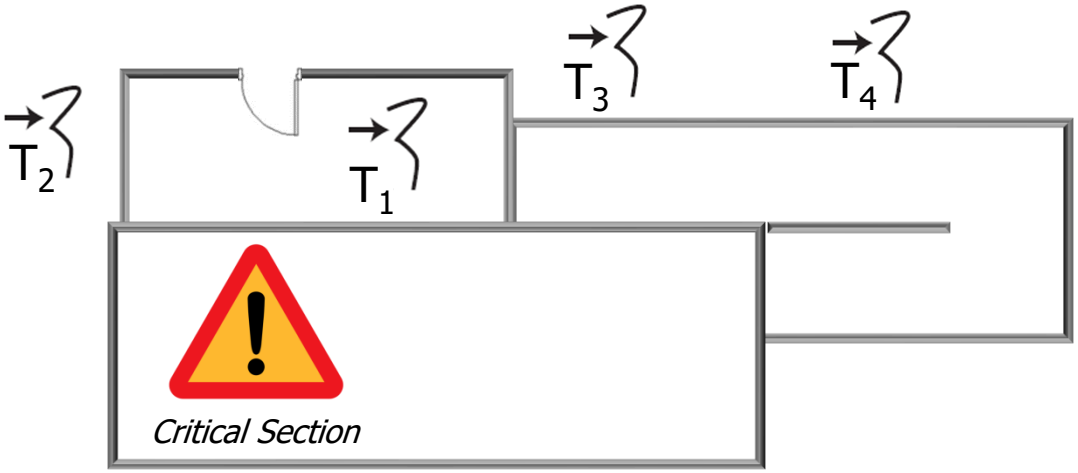
P. Brinch Hansen, *Operating System Principles*, Section 7.2 Class Concept, Prentice Hall, Englewood Cliffs, NJ, (July 1973), 226–232.

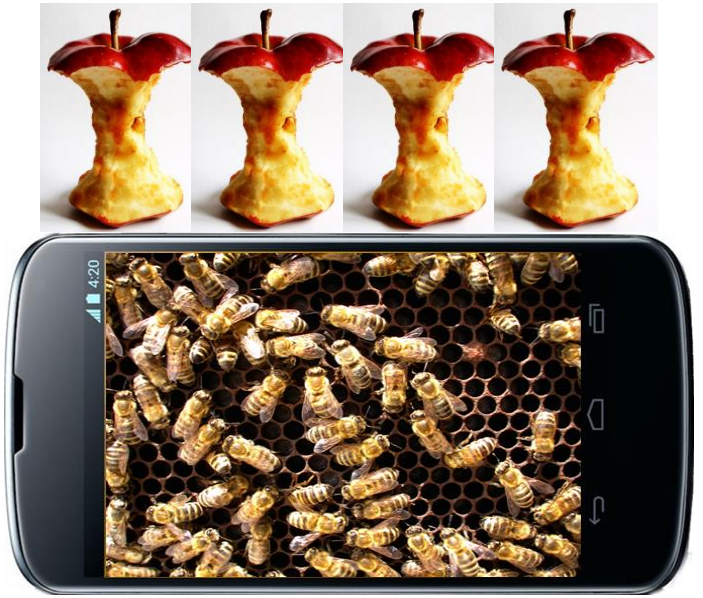
1

See en.wikipedia.org/wiki/Monitor_(synchronization)

Overview of Monitors

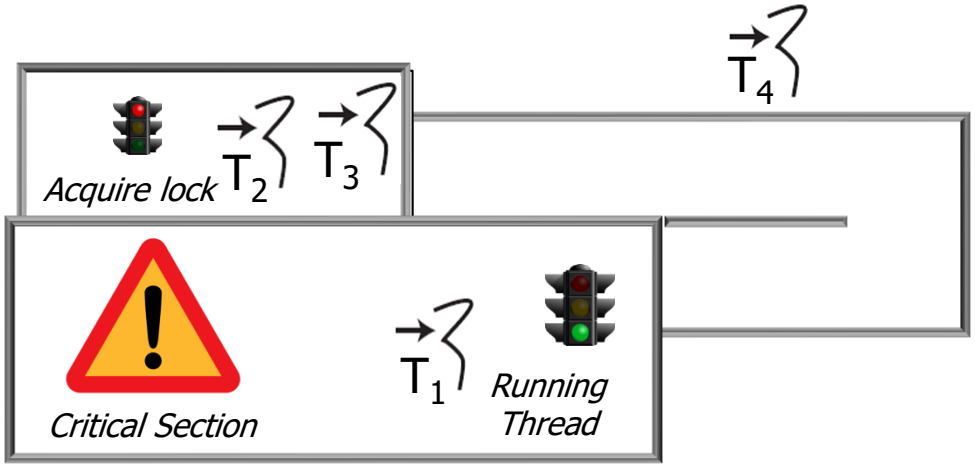
- A monitor provides three capabilities to concurrent programs

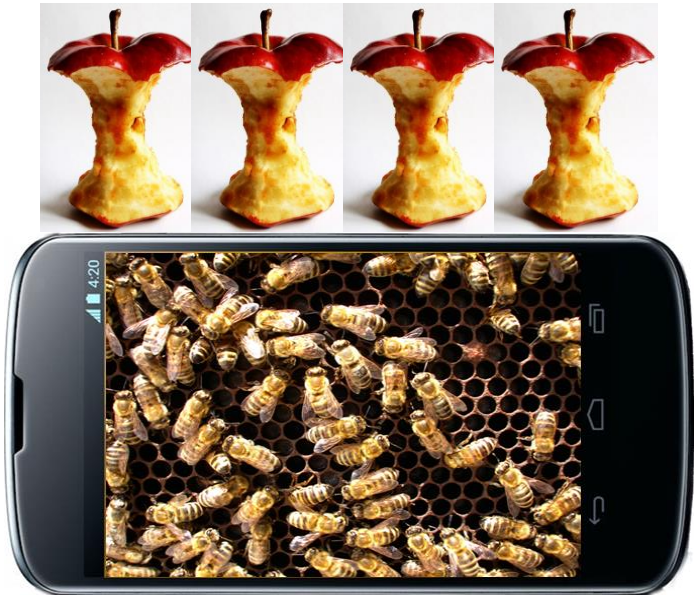




Overview of Monitors

- A monitor provides three capabilities to concurrent programs
  1. Only one thread at a time has mutually exclusive access to a critical section

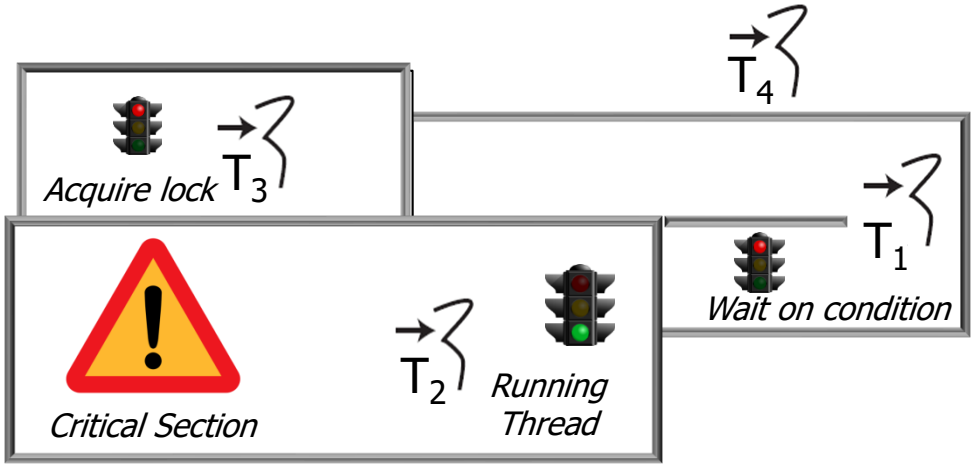


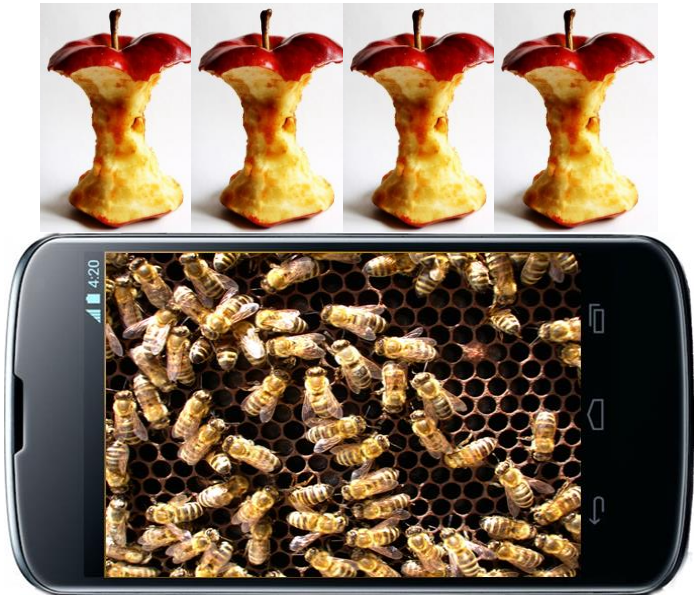


See en.wikipedia.org/wiki/Critical_section

Overview of Monitors

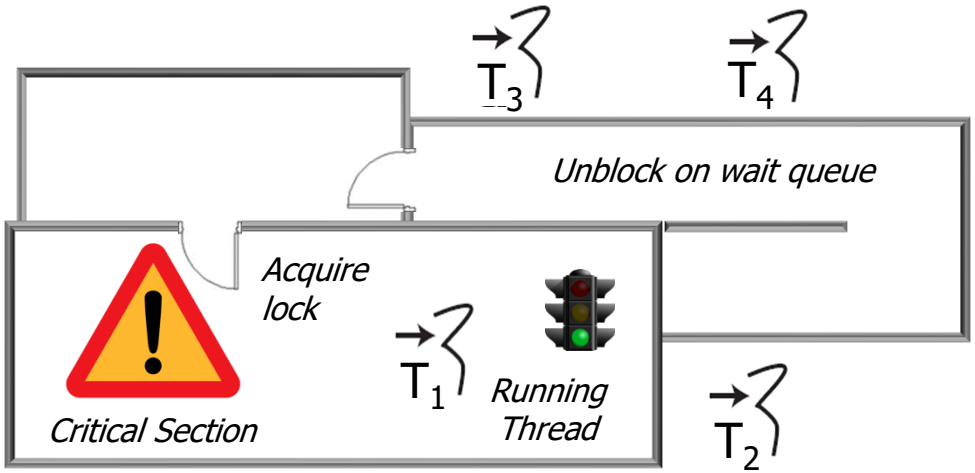
- A monitor provides three capabilities to concurrent programs
  1. Only one thread at a time has mutually exclusive access to a critical section
  2. Threads running in a monitor can block awaiting certain conditions to become true

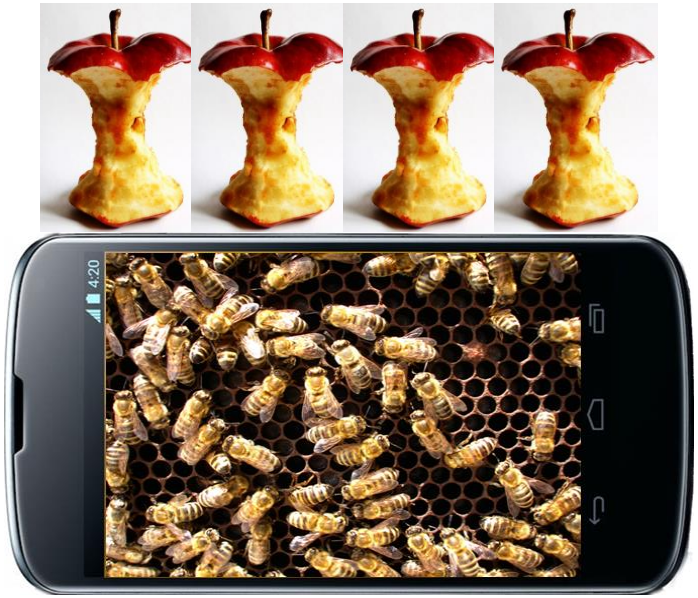




Overview of Monitors

- A monitor provides three capabilities to concurrent programs
  1. Only one thread at a time has mutually exclusive access to a critical section
  2. Threads running in a monitor can block awaiting certain conditions to become true
  3. A thread can notify one or more threads that conditions they're waiting on have been met

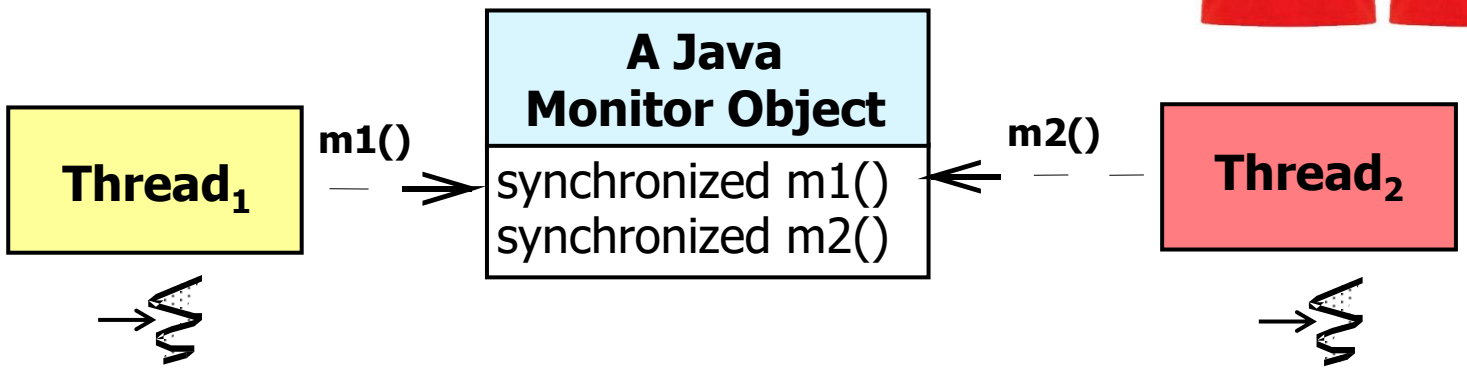




Overview of Built-in Java Monitor Objects

Overview of Java Built-in Monitor Objects

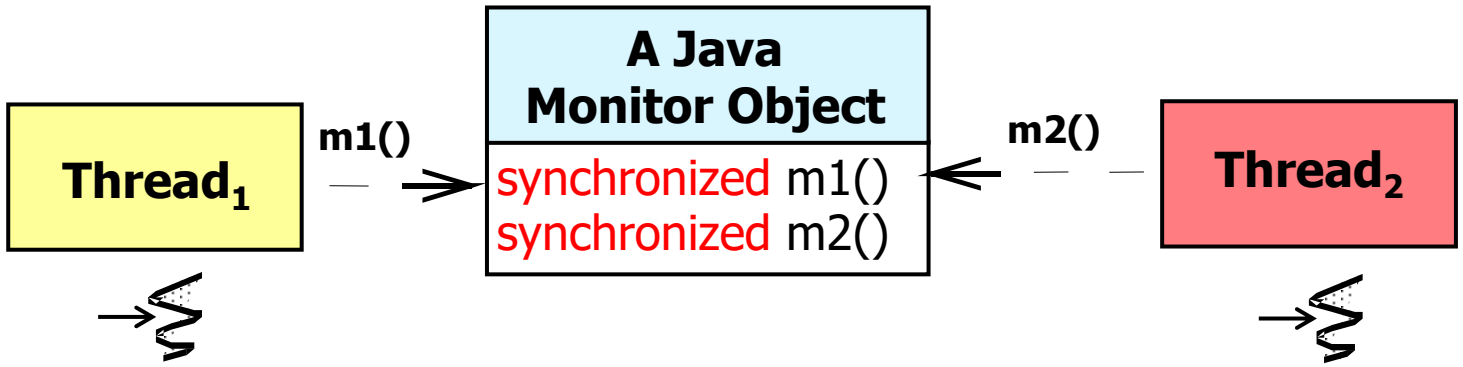
- All objects in Java can be used as built-in monitor objects, which support two types of thread synchronization



See en.wikipedia.org/wiki/Monitor_(synchronization)#Implicit_condition_variable_monitors

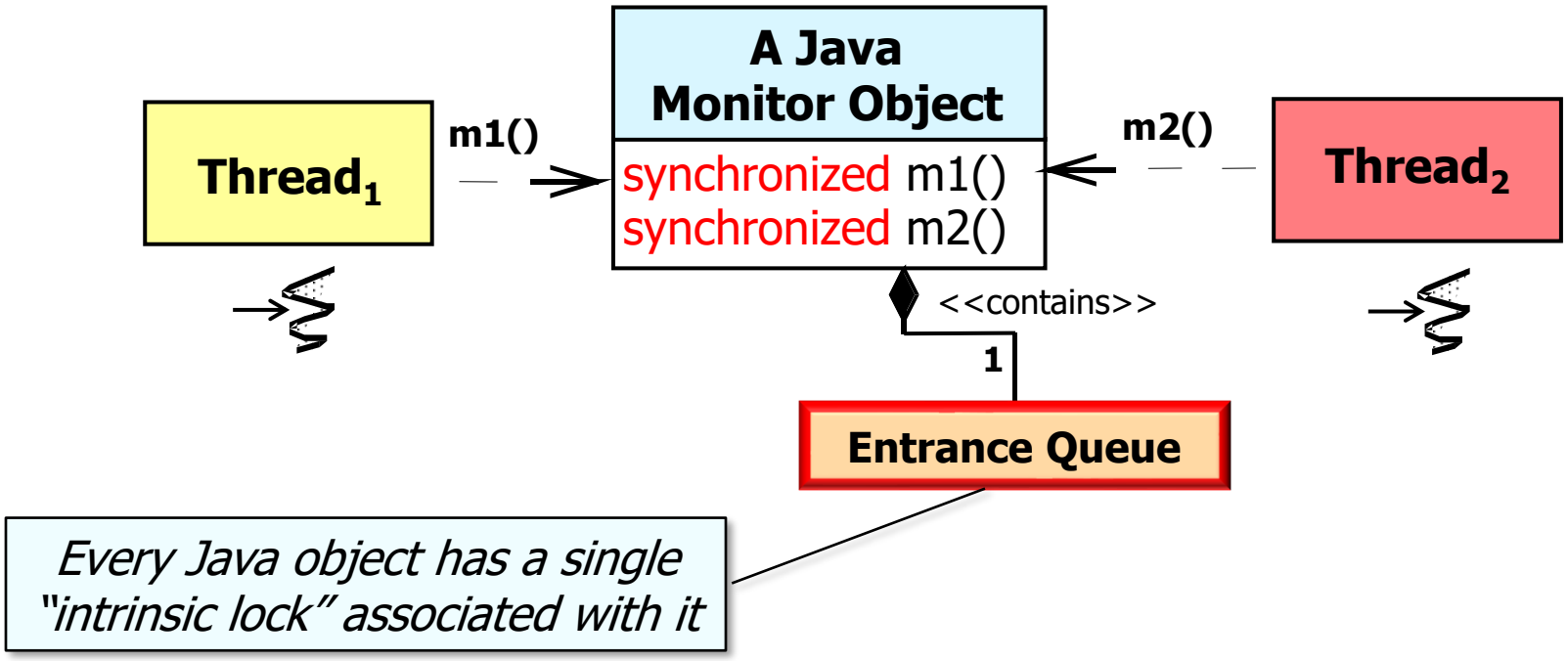
Overview of Java Built-in Monitor Objects

- All objects in Java can be used as built-in monitor objects, which support two types of thread synchronization
  - **Mutual exclusion** – allows concurrent access & updates to shared data without race conditions



Overview of Java Built-in Monitor Objects

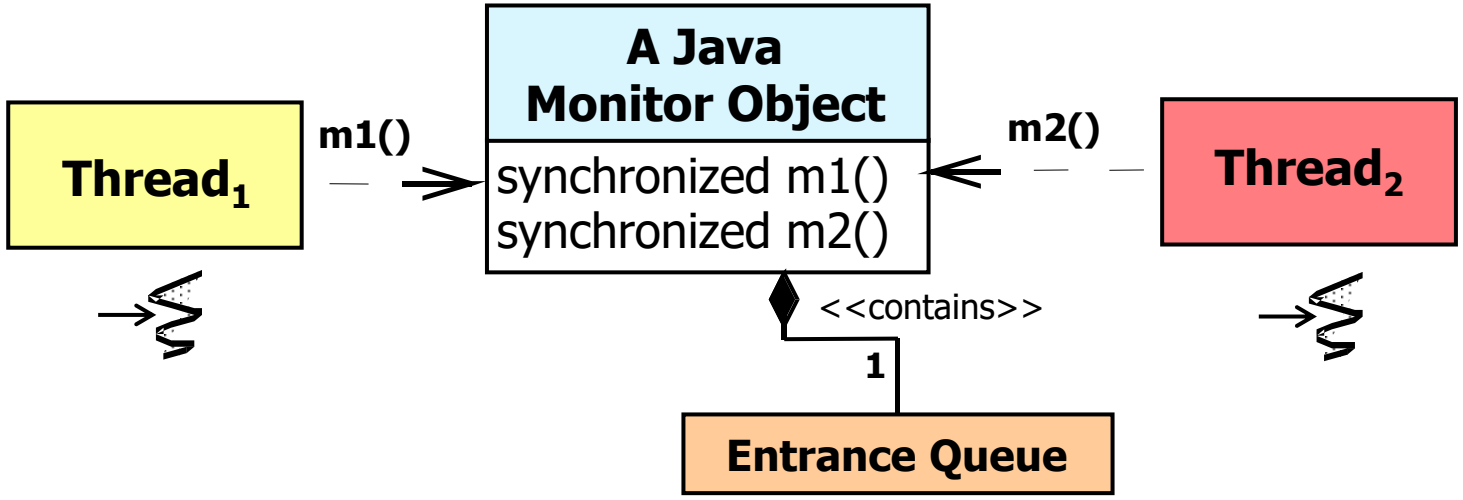
- All objects in Java can be used as built-in monitor objects, which support two types of thread synchronization
  - **Mutual exclusion** – allows concurrent access & updates to shared data without race conditions



Java's execution environment supports mutual exclusion via an entrance queue & synchronized methods/statements

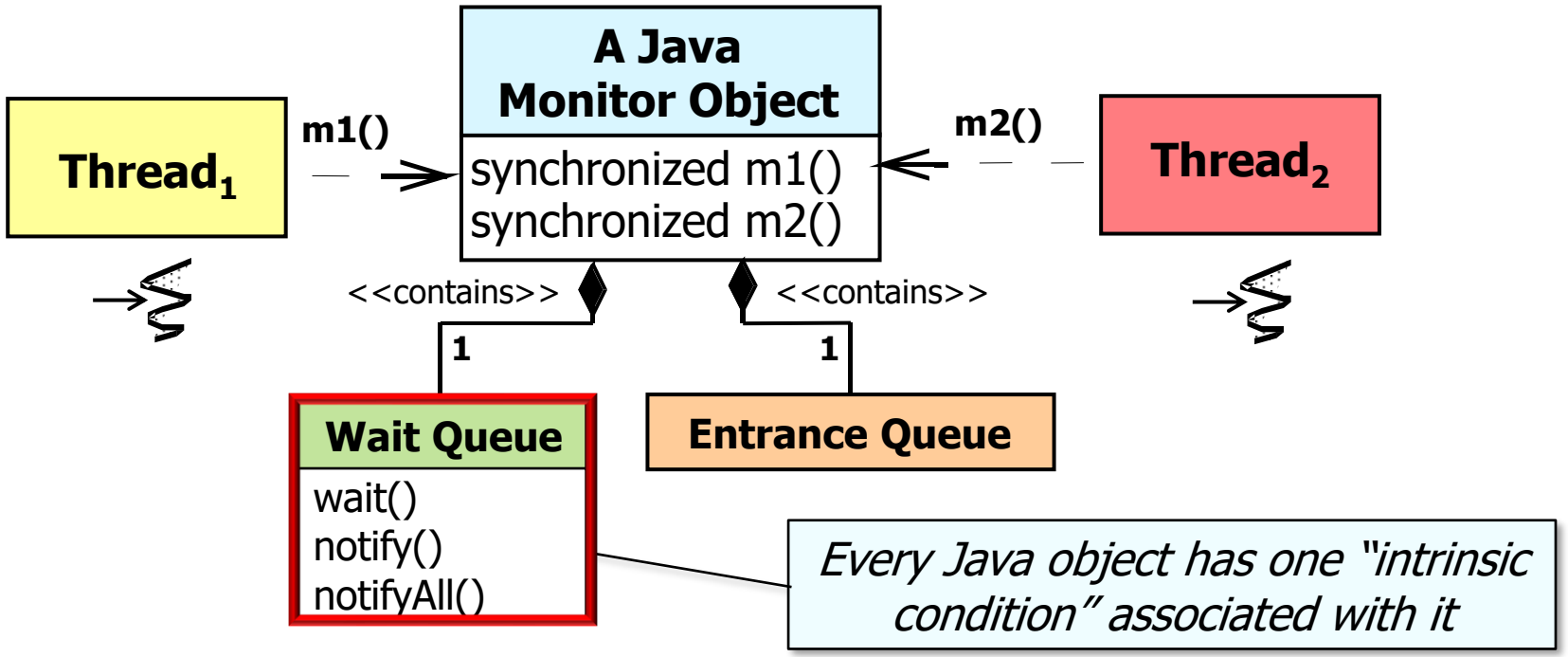
Overview of Java Built-in Monitor Objects

- All objects in Java can be used as built-in monitor objects, which support two types of thread synchronization
  - **Mutual exclusion** – allows concurrent access & updates to shared data without race conditions
  - **Coordination** – Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.



Overview of Java Built-in Monitor Objects

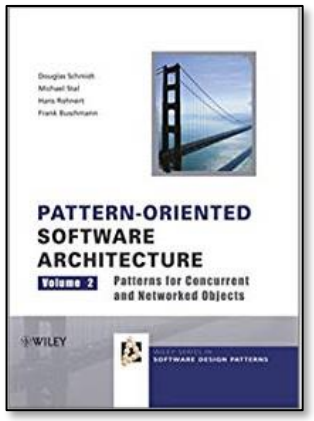
- All objects in Java can be used as built-in monitor objects, which support two types of thread synchronization
  - **Mutual exclusion** – allows concurrent access & updates to shared data without race conditions
  - **Coordination** – Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.

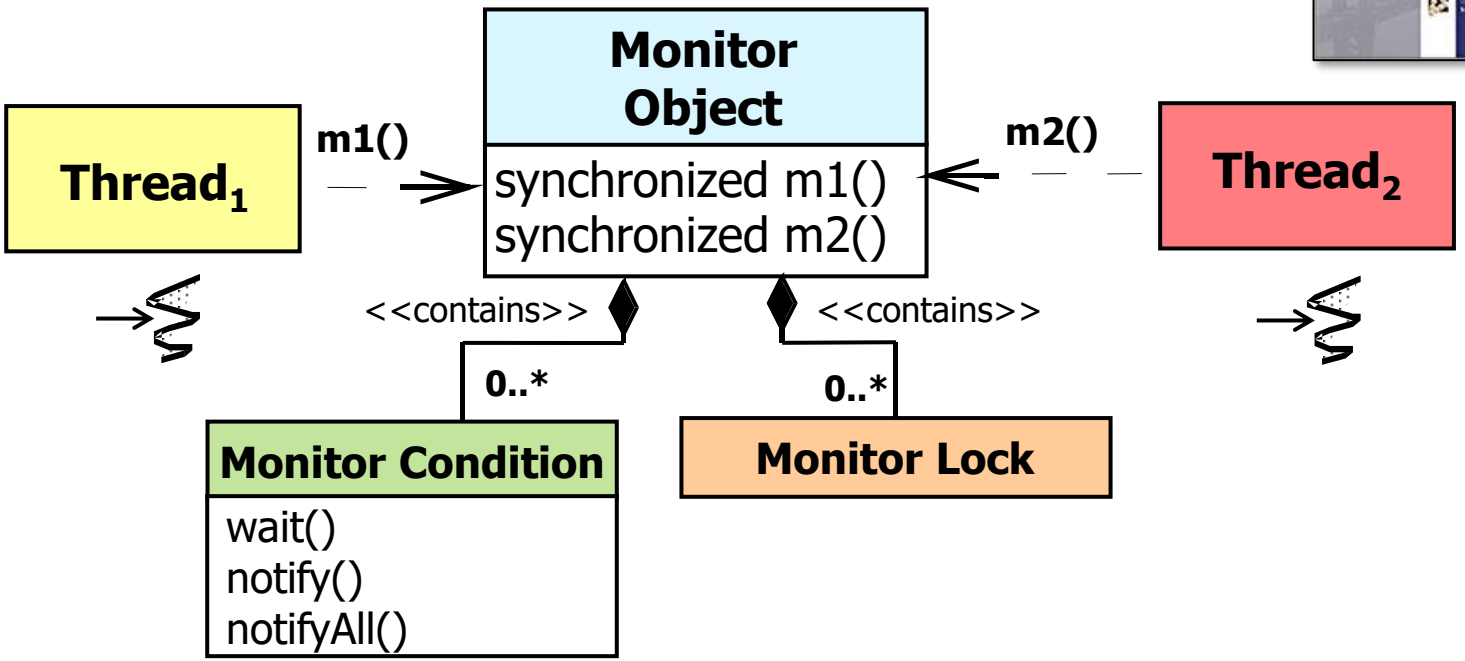


Java's execution environment supports coordination via a wait queue & notification mechanisms

Overview of Java Built-in Monitor Objects

- These mechanisms implement a variant of the *Monitor Object* pattern

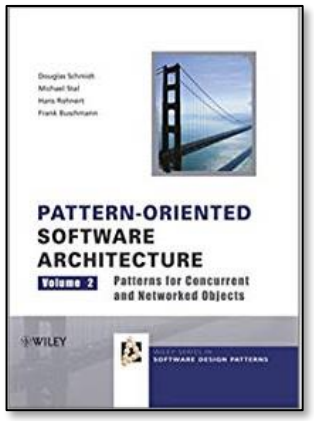


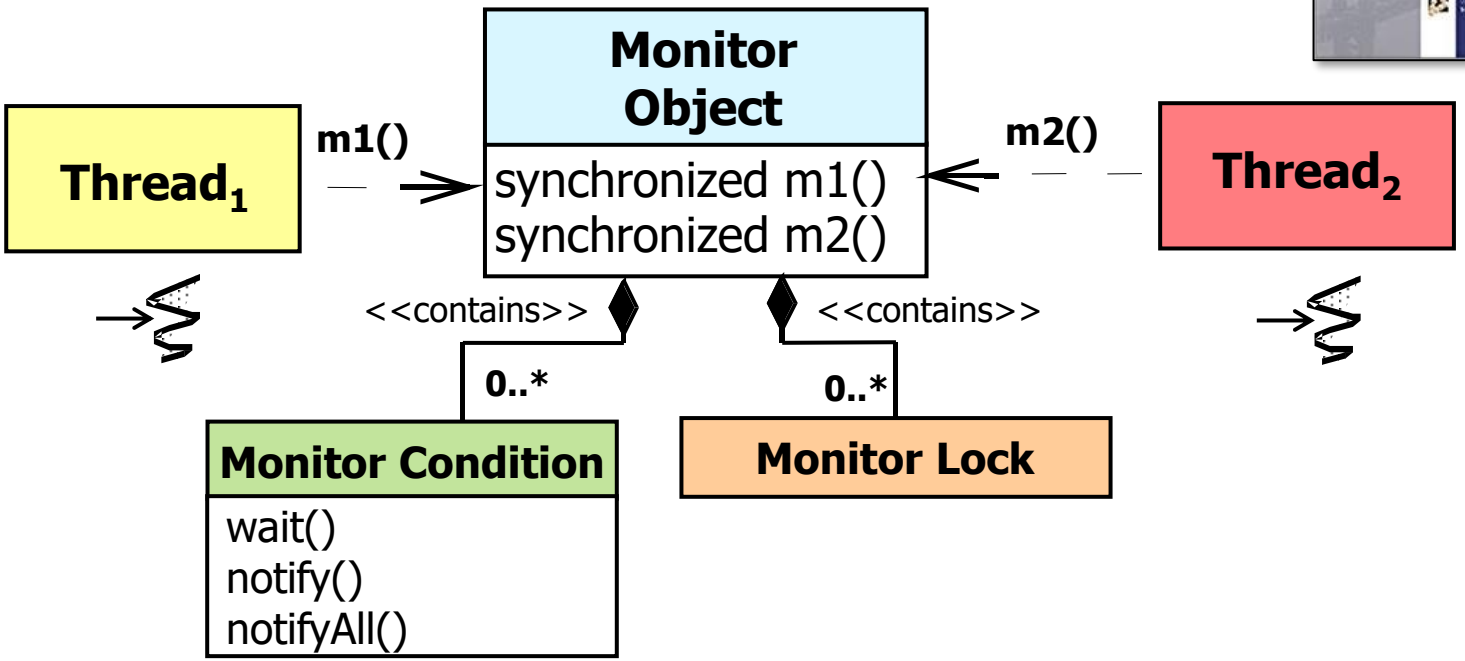


See www.dre.vanderbilt.edu/~schmidt/PDF/monitor.pdf

Overview of Java Built-in Monitor Objects

- These mechanisms implement a variant of the *Monitor Object* pattern
  - **Intent** – Ensure that only one method runs within an object & allow an object's methods to cooperatively schedule their execution sequences

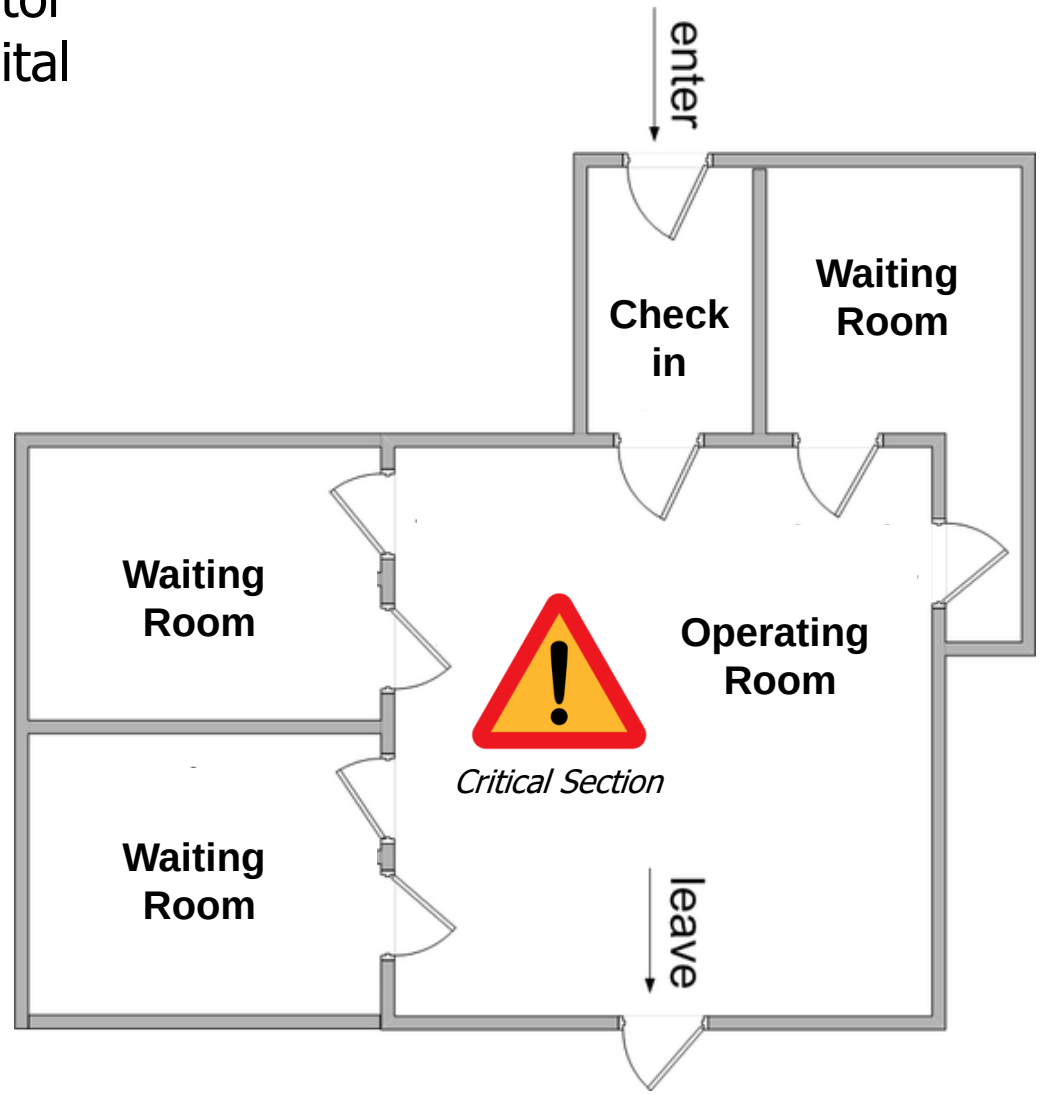




Human Known Use of Monitors

Human Know Use of Monitors

- A human known use of a monitor is an operating room in a hospital



End of Introduction to Java Monitor Objects