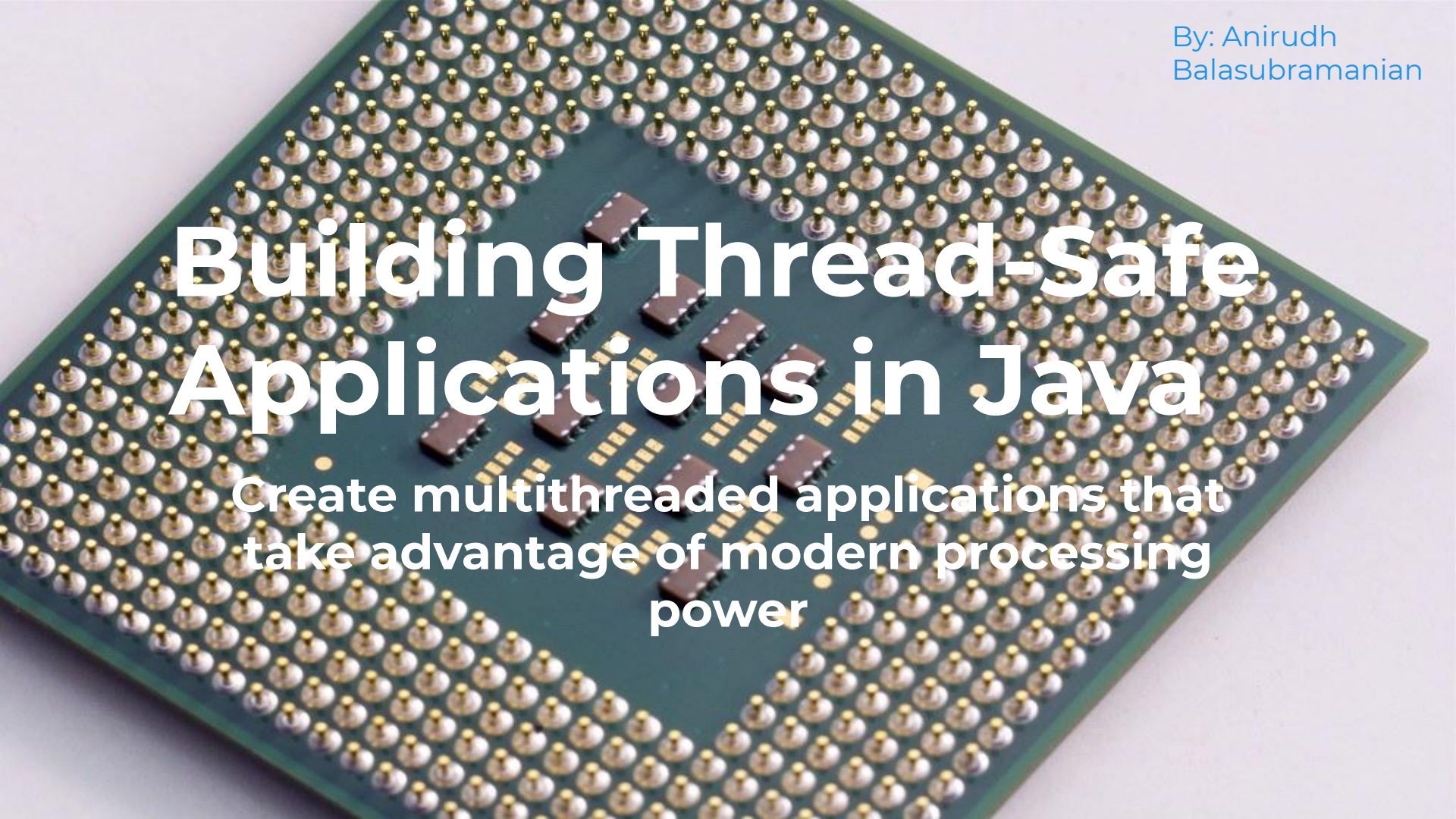
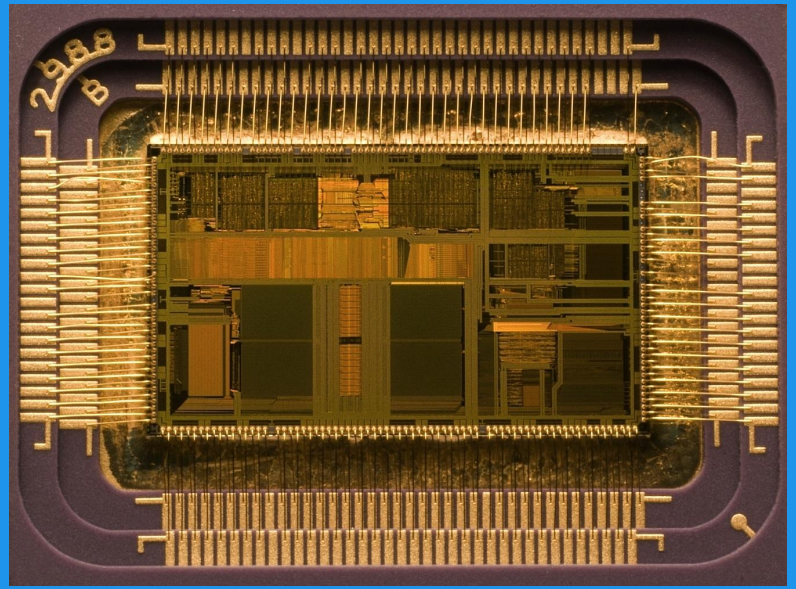


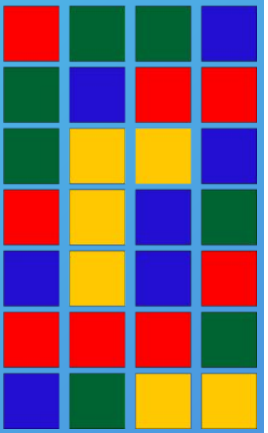
By: Anirudh Balasubramanian

Building Thread-Safe Applications in Java

Create multithreaded applications that take advantage of modern processing power

Introduction to Multithreading and Our Project



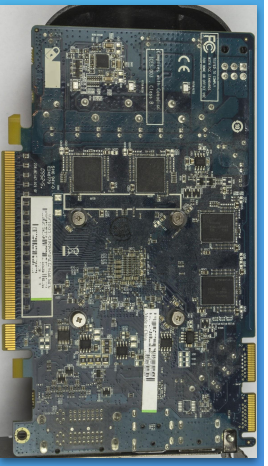


What are Threads and Why do we Care?

- **For any program you start on your computer, your operating system needs to allot it some system resources**
  - **These can be memory, disk space, CPU time, GPU time, VRAM, etc...**
- **Each program is called a process, and each process has its own system resources**
  - **Generally process can't share resources directly**
  - **Processes use messaging systems or the OS to communicate**
- **Each process has one or more threads**
  - **By default, a Java program only has one thread**
    - This is called the Main thread by convention
  - **Inside of each thread is a series of instructions**

Multithreading and Modern Computing

- In the old days, a CPU would only have one core
  - Thus, only one thread could be executed at a single time
  - Time-slicing is used to ensure that all the active threads get some CPU time
- Modern computers can have multiple CPUs, multiple cores, or even hyperthreading
  - All of these tools allow us to run multiple threads at once in a computer
  - With well designed programs we can have multiple threads that efficiently use system resources and execute in parallel
- The added complexity of multiple threads does require different design philosophies and techniques than single-thread programs
- Parallel execution drives much of modern computing from Machine Learning, GUI interfaces, or just faster general computing



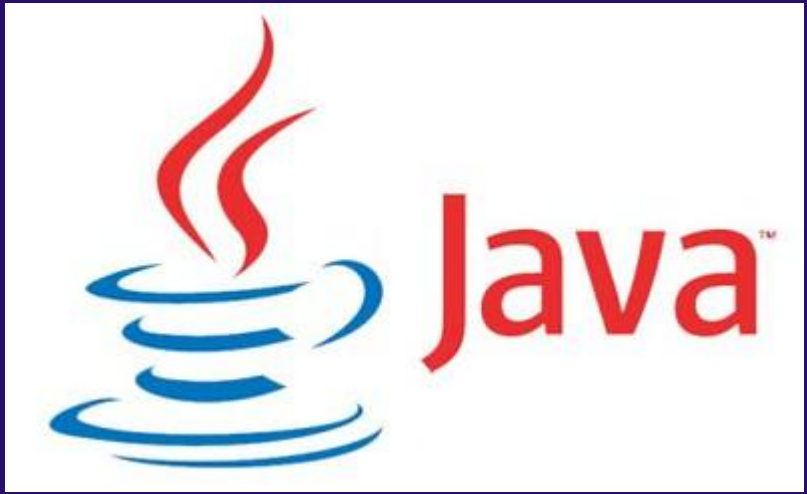
Multithreading in the Real World/Our Project

- Even when you're coding, multiple threads are executing in the background to allow the GUI to update at the same time as your program and background processes
- Video games depend on having many parallel tasks executing to ensure a user's input can be processed at the same time as the game's logic
- **Our program will be a thread-safe application that updates a user's score from various threads simultaneously**
    - **We will see that thread-safe applications are harder to develop but can be orders of magnitude faster than single threaded programs**

Our Project

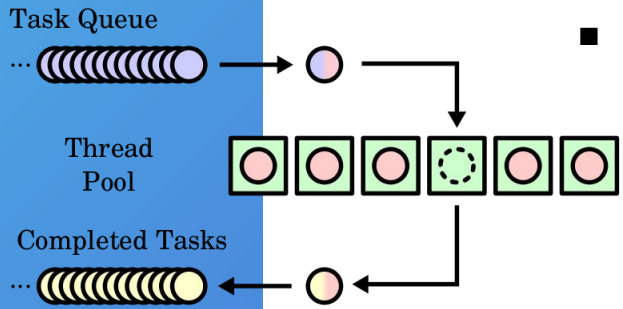
- We will learn how to make a threadsafe bank application that can be modified by multiple threads simultaneously
- We will learn how to use multiple threads, and how to keep them all separated from each other
- We will use Rhyme and Java to create our application
    - Your Rhyme VM should be pre-configured so you can get started once your desktop is connected

Introduction to Java Threads



Introduction to Java Threads

- **Two main ways to create a Thread**
  - **Remember that by default we have the main thread executing our program**
  - **With Inheritance**
    - Using inheritance, we can inherit the Thread class and then create instances of this class
    - Usually not used since Java doesn't support multiple inheritance
  - **With Interfaces**
    - We can implement the Runnable interface which tells Java that this task can take a full thread
    - We have to make sure we keep track of when we use the class vs when we want to perform thread functions



Methods of the Thread Class

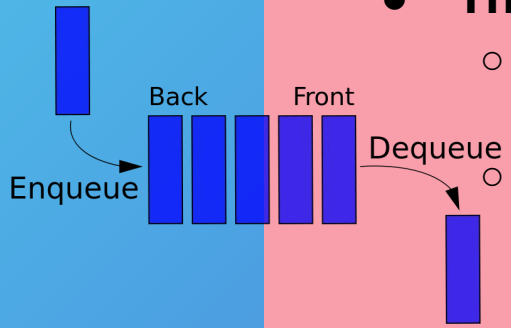
- **Constructors**
  - **Two constructors, default and one that creates a thread for a given Runnable object**
- **Static Methods**
  - **Sleep**
    - Sets thread to sleep for a given number of milliseconds
      - Can throw an InterruptedException which is checked
        - Most cases, we will put it inside of a try catch block
      - Exception only happens if a thread is interrupted, but this is extremely unlikely in modern code
  - **Yield**
    - Stops a thread temporarily and allows other threads system time

Methods of the Thread Class Cont.

- **Instance Methods**
  - **Start**
    - Tells JVM to call the given instance's run method
  - **isAlive**
    - Returns true if thread is currently running
  - **setPriority**
    - Sets priority of thread
      - Priorities range from 1-10 (10 is the max priority)
      - Constants such as Thread.MIN_PRIORITY are often used
  - **Join**
    - Waits for this thread to finish
  - **Interrupt**
    - Stops the thread by interrupting it

Thread Pools in Java

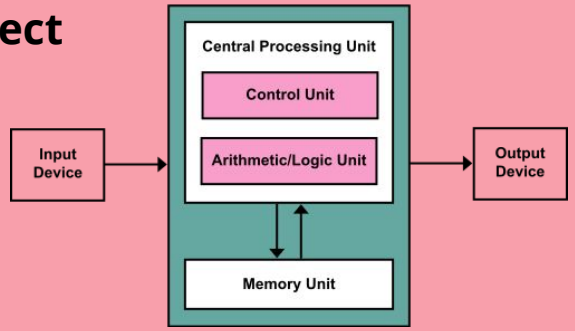
Thread Priority



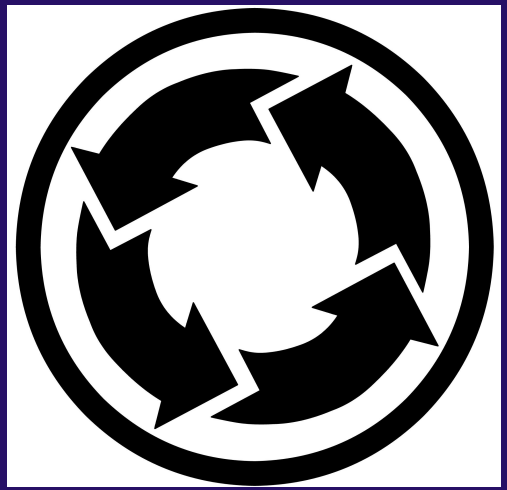
- Threads need some way of regulating themselves
  - In other words, we need a way of knowing how long and when a thread will get CPU time
  - Remember that each application can have multiple threads, and that a computer can have hundreds to thousands of threads going at once
- Most CPUs have a set of priorities ranging from 1-32, but the JVM has priorities from 1-10
  - Your platform specific compiler will figure out how to relate your thread priorities to your CPU
- We use the setPriority method to set the priority of each thread we create (1 is the lowest, 10 is the highest)

What are Thread Pools?

- Thread Pools are used to regulate multiple threads in an application without needing to manually needing to start a thread for each task
- Java uses the Executor interface for executing tasks in the thread pool
  - ExecutorService is a subinterface of executor that is used to manage a Thread Pool
  - We will use static methods of Executors to create a new Executor object



Race Conditions and Synchronization



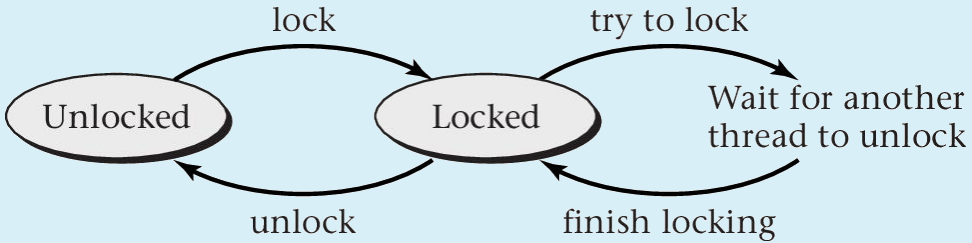
Race Conditions

- Race conditions are a major source of trouble when dealing with multithreaded applications
- What is a Race Condition?
  - A race condition happens when two threads are trying to modify the same memory object, and depending on the ordering in which they are executed, we will have a different result

- In our banking app, this will happen whenever we have different threads trying to modify a single account balance at the same time

Thread Synchronization

- **To avoid race conditions we have to synchronize the threads**
- **Thread Synchronization allows us to say that only one thread can simultaneously access a certain part of the program**
  - **This region is called the critical region**
  - **A method, loop, or any block of code can be deemed synchronized**

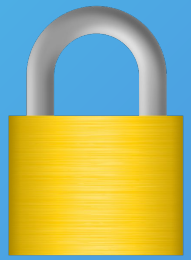


Thread Synchronization cont.

- Any other threads that try and access the thread at this time will be blocked and wait until the other thread using the synchronized region is done
  - Synchronization allows us to stop race conditions, but what if we want multiple threads to work together
- For instance methods, the synchronization occurs on the object
- For static methods, the synchronization occurs on the class

Cooperating Between Threads with Conditions

Lock Basics



- **When we synchronize a section of code, each thread acquires a lock on the object or class, executes the code, then releases the lock**
- **For another class to use this block of code, we need access to the lock**
- **Lock is an interface, ReentrantLock is the concrete implementation**
  - **Each lock must have a fairness policy**
    - If fairness is true, the longest waiting thread will get the lock next
    - Otherwise it will be passed randomly
    - A fair fairness policy will be slower but will have smaller variances in time to obtain locks and guarantees no starvation

Conditions Basics

- Conditions
  - Conditions are used to facilitate communications amongst threads
  - Conditions are created by calling newCondition() on a lock object
  - Await, signal, and signalAll are used for thread communication
  - Await causes the thread to wait until the condition is signaled
  - Signal wakes up one waiting thread
  - SignalAll wakes up all waiting threads
- If we don’t explicitly signal a thread, it won’t be accessible and the thread “starves” and dies
- In essence, we pass locks between threads to prevent race conditions, and locks are freed and locked by signals
- Locks/Conditions in their modern forms were introduced in Java 5

Wrap-Up our Project

Conclusion

- Remember we need to free up a lock, otherwise the other thread will never get the lock and get starved
- If we create threads in a Java application, the JVM will allow your computer to use multiple threads/CPU cores to tackle your program
    - It will automatically handle balancing your application with other system applications
- Generally, we want a new thread for each task that can execute separately, and when threads need to work together we can use synchronization
- We can also pass locks between objects/classes and use conditions and their methods to switch control in our application between various threads
- Normally a Java program has only the default main thread

Hope you enjoyed the course!

Please leave feedback so I can continue to improve my courses

- **Future Goals after this Course**
  - **Learn how to use multithreaded applications in client server apps to have multiple connections at once**
  - **Apply multithreading to one of your existing applications to increase its speed**
- **Remember, the best way to practice programming is by making programs**