Quiz 2 Review

Hayden, Esha, Bryan, Vlad
Concurrency and Parallelism

- Many tasks require many things to happen at once
  - E.g., Client/server
  - Concurrency — Tasks span overlapping time periods

- Many tasks go faster with many things happening at once
  - Parallelism — Tasks run at the same time
  - Amdahl's law
    
    \[ T_s = \text{serial time}, \quad T_p = \text{cumulative parallel time} \]
    
    Ideally, \( T(n) = T_s + T_p / n \)
    
    Maximum speedup is \( (T_s + T_p) / T_s \)
Threads

- Controlled from within a program
- Simulates a processor, and shares memory with other threads

```java
public interface Runnable {
    public void run();
}

public class A implements Runnable {
    public void run() { ... }
}

A a = new A();
Thread t = new Thread(a);
t.start();
```
Race conditions

- Jack and Jill deposit $100 to same bank account

<table>
<thead>
<tr>
<th>Jack's thread</th>
<th>Jill's thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get balance = $50</td>
<td>Get balance = $50</td>
</tr>
<tr>
<td>Add $100</td>
<td>Add $100</td>
</tr>
<tr>
<td>(balance = 50+100)</td>
<td>(balance = 50+100)</td>
</tr>
<tr>
<td>Write $150</td>
<td>Write $150</td>
</tr>
</tbody>
</table>

Correctness depends on timing of uncontrollable events.
Add locks!

- Depositing money is a *critical section*
  - We don't want any interleaving of threads

```java
class BankAccount {
  ...
  synchronized void deposit(int m) {
    balance = balance + m;
  }
}

— OR —

synchronized (obj) {
  balance = balance + m;
}
```
Deadlocks

- Jack transfers money to Jill and vice-versa

```
synchronized (from) {
    synchronized (to) {
        // transfer
    }
}
```

Jack's thread
- Lock Jack's account
- Get lock for Jill's account

Jill's thread
- Lock Jill's account
- Get lock for Jack's account

They get stuck waiting for each other. (Locking graph has a cycle.)
Rules to Remember

- Safe to access shared resources concurrently if you:
  - Don't alter resource's state (e.g., read)
  - Make an idempotent operation
  - Use locks to only access one at a time

- Avoid deadlocks if you:
  - Establish an ordering for locks
  - Acquire in increasing order
  - Release in decreasing order

- Coarse lock granularity gives bad performance; fine granularity can give race conditions or deadlocks
Message Passing

- Threads don't directly access each other's memory
- Threads use blocking queues to send requests
  - `java.util.concurrentBlockingQueue` is a thread-safe blocking queue
- Can still suffer from race conditions and deadlocks

Diagram:

Mouse thread → BlockingQueue → Event handling thread

Keyboard thread → BlockingQueue → Event handling thread
Testing Concurrent Programs

● Many more possible states, so sufficient test coverage is not practical

● Transitions are nondeterministic
  ○ Sometimes you'll see a bug, sometimes you won't

● Try to order locks, use immutability, etc.

It's really hard!
Processes & Sockets
Processes

- A process is an instance of a running program that is isolated from other processes on the same machine (particularly for resources like memory).

- Tries to make the program feel like it has the whole machine to itself – like a fresh computer has been created, with fresh memory.

- By default, processes have no shared memory.

- Automatically ready for message passing (standard input & output streams).
Processes

- Self-contained execution environment

- A complete, private set of basic run-time resources
  - Each process has its own memory space

- Synonymous with programs or applications
  - What the user sees as a single application may in fact be a set of cooperating processes
Processes

- Communication between processes
  - Support *Inter Process Communication* (IPC) resources, such as pipes and sockets
  - Not just for communication between processes on the same system, but processes on different systems

- Most implementations of the Java virtual machine run as a single process

- A Java application can create additional processes using a *ProcessBuilder* object
# Processes

![Windows Task Manager](image.png)

<table>
<thead>
<tr>
<th>Image Name</th>
<th>User Name</th>
<th>CPU</th>
<th>Mem Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Idle Process</td>
<td>SYSTEM</td>
<td>55</td>
<td>28 K</td>
</tr>
<tr>
<td>mysql.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>32</td>
<td>5.036 K</td>
</tr>
<tr>
<td>sqlservr.exe</td>
<td>NETWORK SERVICE</td>
<td>10</td>
<td>340.500 K</td>
</tr>
<tr>
<td>winamp.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>01</td>
<td>26.908 K</td>
</tr>
<tr>
<td>xampp-control.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>01</td>
<td>2.596 K</td>
</tr>
<tr>
<td>services.exe</td>
<td>SYSTEM</td>
<td>00</td>
<td>2.380 K</td>
</tr>
<tr>
<td>heidisql.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>00</td>
<td>1.408 K</td>
</tr>
<tr>
<td>ssms.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>00</td>
<td>9.896 K</td>
</tr>
<tr>
<td>taskmgr.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>00</td>
<td>3.492 K</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>SYSTEM</td>
<td>00</td>
<td>4.336 K</td>
</tr>
<tr>
<td>explorer.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>00</td>
<td>19.220 K</td>
</tr>
<tr>
<td>wmiprvse.exe</td>
<td>SYSTEM</td>
<td>00</td>
<td>532 K</td>
</tr>
<tr>
<td>alg.exe</td>
<td>LOCAL SERVICE</td>
<td>00</td>
<td>292 K</td>
</tr>
<tr>
<td>TSVNCache.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>00</td>
<td>1.492 K</td>
</tr>
<tr>
<td>ashWebSv.exe</td>
<td>SYSTEM</td>
<td>00</td>
<td>852 K</td>
</tr>
<tr>
<td>ashMailSv.exe</td>
<td>SYSTEM</td>
<td>00</td>
<td>908 K</td>
</tr>
<tr>
<td>mspaint.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>00</td>
<td>9.048 K</td>
</tr>
<tr>
<td>wampmanager.exe</td>
<td>Pop Adrian-Nicolae</td>
<td>00</td>
<td>3.904 K</td>
</tr>
<tr>
<td>ashDisk.exe</td>
<td>Pron Adrian-Nicolae</td>
<td>00</td>
<td>1.272 K</td>
</tr>
</tbody>
</table>

- Processes: 49
- CPU Usage: 44%
- Commit Charge: 1066M / 2166M
Sockets

● A socket is one end-point of a two-way communication link between two programs running on the network

● Socket classes are used to represent the connection between a client program and a server program

● The java.net package provides two classes
  ○ Socket and ServerSocket
Sockets

- A network interface is identified by an IP address
  - (or a hostname, which translates into an IP address)
    - Examples: 127.0.0.1, localhost; web.mit.edu

- An interface has 65536 ports
  - Numbered from 0 to 65535

- A server process binds to a port (the listening port)
  - Clients have to know which number it’s binding to.
  - Some numbers are well-known
  - When it’s not a standard port for the kind of server, you just treat it as part of the address
Sockets

Server
- socket()
- bind()
- listen()
- accept()
- recv()
- send()
- close()

Client
- socket()
- connect()
- send()
- recv()
- close()
Performance Engineering

- Why is performance important?
  - Acceptable response time
  - Ability to add more functionality
  - Ability to scale
  - Use less power / resources

- Performance engineering requires
  - Knowing that there is a performance problem
  - Identifying the performance bottlenecks
  - Establishing the leading cause of the problem
  - Eliminating the performance problem
Performance Engineering

● How do you know if your program is performing well?
  ○ Back of the envelope calculations
  ○ Performance debugging
  ○ Scalability testing
  ○ Comparisons to similar programs
  ○ Experience!

● Identifying Bottlenecks
  ○ Profile the programs
  ○ Scalability Testing
  ○ Measure without Perturbing
Performance Engineering

● Establishing Cause
  ○ Study the algorithm
  ○ Study the data structures & data layout
  ○ Study the program structure
  ○ Trial-and-error

● Eliminating Performance Problems
  ○ Performance cuts through abstraction boundaries
  ○ Need to understand the abstraction layers and their impact on overall performance
  ○ Adhere to good software-engineering principles
Event-based programming

● Composite pattern (view hierarchy)
● Publish-subscribe pattern
● Event-based programming control flow
● Model-view-controller pattern
● Concurrency in GUIs
Composite pattern

View hierarchy is an example of the Composite pattern

Primitive views don’t contain other views
- button, tree widget, textbox, thumbnail, etc.

Composite views are used for grouping or modifying other views
- JSplitPane displays two views side-by-side with an adjustable splitter
- JScrollPane displays only part of a view, with adjustable scrollbars

Key idea:
primitives and composites implement a common interface (actually an abstract class JComponent)
containers can hold any JComponent, - both primitives and containers
Publish-subscribe pattern

- An event source generates a stream of discrete events
- Listeners register interest in events from the source
- Listeners can unsubscribe when they no longer want events
- When an event occurs, the event source distributes it to all interested listeners
Event-based programming control flow

- A top-level event loop reads input from mouse and keyboard

- For each input event, it finds the right view in the hierarchy (by looking at the x, y position of the mouse) and sends the event to that view’s listeners

- Listener does its thing (e.g., modifying the view hierarchy) and returns immediately to the event loop
Model-view-controller

Model-View-Controller (MVC) separates responsibilities

● View displays output
● Controller handles input
● Model stores application data
Model-view-controller

**View handles output**
- calls observers on the model to display it
- listens for model changes and updates display

**Controller handles input**
- listens for input events on the view hierarchy
- calls mutators on model or view

**Model handles application state**
- implements state-changing behavior
- sends change events to views
Event-based programming pitfalls

1. Listener calls observers on the event source
   - event source may not have a valid state
2. Listener calls mutator on an event source
   - may result in infinite chain of events
3. Listener unsubscribes itself from the event source
   - may make iteration over listeners mess up
Concurrency in GUIs

- Event loop reads an input event from the queue and dispatches it to listeners on the view hierarchy.
- In Java, the event loop runs on a special event-handling thread, started automatically when a user interface object is created.
Functional programming

- Abstracts away control flow

- Functions are "first-class" values
  - functions can be stored in variables and passed around
  - functions can take other functions as parameters
  - functions can return other functions
  - etc.
map-filter-reduce

map, filter, and reduce are functions on sequences

map : (T → U) x Seq<T> → Seq<U>
filter : (T → boolean) x Seq<T> → Seq<T>
reduce : (U x T → U) x Seq<T> x U → U
    • a.k.a. fold-left
    • fold-right : (T x U → U) x Seq<T> x U → U

map: applies a function to every element and returns the resulting sequence

filter: applies a function to each element and keeps the ones that returned true

reduce: applies a function iteratively to (prev result, next element) and returns the final return. may use an initial value
other useful functions

- `flatten : List<List<T>> \rightarrow List<T>`
- `compose : (A \rightarrow B) \times (B \rightarrow C) \rightarrow (A \rightarrow C)`
- `chain : List<(T \rightarrow T)> \rightarrow (T \rightarrow T)`
Design Pattern Recap

- Composite
- Publish-subscribe
- Model-View-Controller

- Ingredients in a recipe: don't put mushrooms in a cherry pie!
Design Patterns: Breaking News

- You are watching news on TV in wake of an unexpected tragedy
  - **Model**: news channels supplied by cable provider
  - **View**: the television
  - **Controller**: your remote

(In memory of Sean A. Collier and the other victims of the Boston Marathon Bombings.)
Design Patterns of Old...

- Encapsulation
- Subclassing
- Iteration
- Exceptions
- Generics
- Factories
- Interpreters/Visitors
- Sentinel object/poison pill
Design Patterns of Old, cont.

- **Encapsulation:**
  - Private `balance`; public `getBalance`
  - Don't want a bank clerk setting your balance!

- **Subclassing:**
  - A `Cat` is an `Animal`; a `Human` is an `Animal`; a `King` is a `Human`
  - `Cat.meow()` or `Human.speak()` vs `Animal.move()`
Design Patterns of Old, cont.

- **Iteration:**
  - `.hasNext()` and `.next()`
  - Maybe dynamically generated!

- **Exceptions:**
  - Checked implies special result; either *throw* or handle
  - Unchecked: catastrophic failure! Not responsibility of programmer to handle
Design Patterns of Old, cont.

- **Generics:**
  - `List<T>`: List of what?
  - Why do we care? What's wrong with `MyCoolList<String>, MyCoolList<Integer>`?

- **Factories:**
  - `ImList<E> = Empty<E>() + Cons<E>(first: E, rest: ImList<E>)`
  - `makeEmpty()`: a factory, "somehow" making an empty ImList!
  - Representation independence
Design Patterns of Old, cont.

- **Interpreter:**
  - Declare new interpreter methods for each new operation; one method for each variant

- **Visitor:**
  - Declare Visitor interface and `accept()` methods for each variant
  - Create new Visitor class implementing the interface for each new operation; `on()` methods for each variant

- **Difference?**
  - Depends on foreseen changes -- change every variant per new operation or change every visitor per new variant
Design Patterns of Old, cont.

- Sentinel object; poison pill
  - Special value
  - Means that it is the end!