Final Exam Review
FPP June 2025

Location: **V28**, Verill hall
Date: **June 19, 2025**
Time: **10:00 am – 12:30** pm

1. The final exam will be closed books, closed notes, and no use of laptops, smart watches, or electronic devices.
2. Structure of exam: **True/False**, **Multiple Choice**, **Short Answer,** and **Programming questions**.
3. Topics: **Lesson from 7 to 12, its Labs and Practice**

CS390 - Fundamental Programming Practices		
No.	**Lesson Name**	**Lesson topic details**
7	Recursion	Introduction to Recursion
		Base case and Recursive case
		Math Problems: Factorial and Fibonacci
		String Problems: Reversing characters in a string, Finding the minimum element in an array
		Recursive Utility Functions: Linear Search, Binary Search, FindMin recursion
		Junit Test and practice
8	The List Data Structure	List ADT, Array Lists and including sort and search
		Linked Lists – singly linked, doubly linked
		Java Collection Framework(List, ArrayList, LinkedList)
		The List interface, the AbstractList class, and Iterator
		Collections.sort, Collections.binarySearch, and RandomAccess
		Four ways of iterating through elements in a list(Regular loop, for each, Iterator, Lambda)
		Comparable vs Comparators
		Four ways to initialize a List(Usual insertion using add(), Arrays.asList(), List.of (), Anonymous)
9	Stacks and Queues	Stack ADT
		Stack Operations(push, pop, peek)
		Stack Implementation – Array and Queue
		Applications of Stack – Symbol balancing
		Java Collection Framework(Queue Interface, Deque Interface, ArrayDeque, PriorityQueue)
		Queue ADT
		Queue Operations(dequeue, enqueue, peek)
		Queue Implementation – Array and Queue
		Applications of Queue– Breadth First Search
10	Binary Search Trees	Introduction to Tree and Binary Search Tree

		Tree Terminologies
		BST Operations(insert, search, delete, Traversals, Printing a Tree)
		Java Collection Framework(Set interface, TreeSet API)
11	Hash Tables	Hashtable ADT(put, get)
		Collision Strategy – Chaining
		Importance of overriding hashcode() and equals()
		Getting good hashcodes
		Java Collection Framework (Map, HashMap, Hashtable, HashSet, TreeMap)
		Comparisons of all the data structures(Advantages, drawbacks and guidelines of choosing the appropriate data structures)
12	Exception-Handling in Java	What Is Exception-Handling All About?
		Classification of Error-Condition Classes(Errors, Checked and Unchecked Exceptions)
		Using/Creating Exception Classes
		Syntax Rules for Try/Catch
		finally Keyword
		Custom Exceptions
13	Working with Files and Databases	Java I/O: Byte, Character, Buffered, Data, and Object Streams (transient keyword)
		File Readers(APIs – Reader, InputStreamReader, BufferedReader)
		Alternative to Reader(Scanner)
		Writers(APIs – OutputStreamWriter, PrintWriter, FileWriter)
		File class, How to search Files
		Interacting with a Database using JDBC(Postgre SQL)
		Interfaces: Driver, Connection, PreparedStatement, CallableStatement, ResultSet
		Registering the Driver and Getting the Connection
		Creating a PreparedStatement(prevents SQL-Injection attacks)
		Execute the Statement (execute, executeQuery, executeUpdate)
		Process ResultSet

T/F, Multiple Choice, Short Answer. These will draw upon the following list of topics from Lessons 7-12.

a. ***Recursion***. Be familiar with the following points:
   - What must be true for a recursion to be a *valid* recursion?
   - Be able to implement (in code) a recursive strategy to solve a problem (as in practice).

b. Be familiar with the different **advantages and disadvantages** of the different **ADT**'s and implementations we have discussed in class: ArrayList, LinkedList, BST (TreeSet), Hashtable (HashMap).

c. Know the classes and interfaces involved in creating a user-defined Collection (like a List) in a way that can make use of **Collections sorting and searching** methods. Review when the List and **Random access** interfaces are implemented and the reasons for using the class **AbstractList**. Be familiar with what you need to do with a list that you create so that it can work with the sort and search functions available in

the **Collections class**.

d. Know how to use an **Iterator** object to iterate through the elements of a list. Know the difference between the **Iterable** and **Iterator** interfaces. Know the role of Iterator in the use of the for each construct.

e. Be familiar with the top level of the **exceptions** hierarchy provided in Java. Understand the difference between errors, **unchecked exceptions(**runtime exceptions**), and checked** exceptions**. Know the most common examples of each type.**

f. Understand the **finally** keyword and be able to think through the behavior of a code sample like the one given in the finally Exercise at the end of Lesson 12.

g. Know which **background data structures** are typically used to implement ArrayList, LinkedList, PriorityQueue, ArrayDeque, TreeSet, TreeMap, Hashtable, HashSet, HashMap.

h. Be able to use built-in classes: ArrayList, LinkedList, PriorityQueue, ArrayDeque, TreeSet, TreeMap, HashSet, HashMap and its iterators.

i. Understand how the **hashCode** function is used in a class to support the use of objects as keys in a **hashtable**. Understand how a hashtable transforms input keys to hashcodes to **hashvalues** (and know the difference between these). Know why equal objects must have equal hashCodes and why it is *desirable* for unequal objects to have different hashCodes. Be familiar with best practices concerning the creation of a hashCode. Make sure you can write code to override hashCode in any class that you create.

j. Given a sequence of ordered values (like integers or Strings), be able to follow the insertion rules to **insert** them into an initially empty **BST** and **delete** the elements. Be able to load a BST by hand using an insertion sequence.

k. be able to follow the insertion rules to **insert** them into an initially empty **PriorityQueue** and **delete** the elements.

l. Be able to write the code for a **linked list, stack, and queue implementations**. Be sure that you are familiar with the technique for iterating from a top **node** (like a header) to some target node in the list. You may be asked to use Node to implement other data structures.

m. Understand how each of the following types of data structures is designed and implemented (in a general way): list, stack, queue, hashtable, bst, set.

n. Be able to explain why, for ordering objects, sometimes the **Comparable** interface is not enough.

o. Be able to explain what it means for a **Comparator** to be consistent with equals, and why comparators *should be* consistent with equals. Be able to create your own Comparator so that it is consistent with equals and be able to use it in a call to Collections.sort.

1. **Programming Questions.** There will be two programming questions:

    1. *Implement data structure*. You will be given the type of background data structure to use; you will use that background data structure to implement some or all the main operations of the main data structure. (SinglyLinkedList, DoublyLinkedList, Queue, or Stack using Node) and *Solve a problem using recursion* (see lab 7)

    2. *Polymorphism*

2. **SCI.** You will be asked to write a short essay to explain how one or more SCI principles are displayed in the realm of computer science. Richer content will be awarded more credit.