CPT204-Final Review

1.1 Arrays

- Motivation: Opening Problem
- Declaring Array Variables
- Creating Arrays
- length and Indexed Variables
- Default values and Shortcut initialization
- Common algorithms with arrays: initialization, printing, sum, min, max, shuffling, shifting, copying
- Enhanced for-Loops (for-each loops)
- Passing and returning arrays to/from methods (Pass By Value)
    - The Heap Segment and the Call Stack Memory
- Returning an Array from a Method: reverse an array
- Searching Arrays: Linear and Binary Search
- Sorting Arrays: Selection Sort and Insertion Sort

1.2 Objects and Classes

- Motivating Problems: Complex objects and GUIs
- Classes, objects, object state and behavior
- Object-oriented Design
- Constructors
- Accessing fields and methods
- Static vs. Non-static
- Static Variables and Methods
- Default values for Class Fields
- Primitive Data Types and Object Types, Copying
- Garbage Collection
- Example classes in Java API: the Date class

- The Random class
- Visibility Modifiers and Accessor/Mutator Methods

2.1 Thinking in Objects

- Immutable Objects and Classes
- Scope of Variables and Default values
- The `this` Keyword
- Calling Overloaded Constructors
- Class Abstraction and Encapsulation
- Designing and implementing the Loan Class
- Designing and implementing the BMI Class
- Designing and implementing the Course Class
- Designing and implementing the StackOfIntegers Class
- The String Class in detail
- Regular Expressions
- Command-Line Parameters
- StringBuilder and StringBuffer
- The Character Class
- Designing Classes

2.2 Inheritance and Polumorphism

- Motivation: Model classes with similar properties and methods
- Declaring a Subclass
- Constructor Chaining
- Calling Superclass Methods with super
- Overriding Methods in the Superclass
- The Object Class and Its Methods: toString()
- Overloading vs. Overriding
- Method Matching vs. Binding

- Polymorphism, Dynamic Binding and Generic Programming
- Casting Objects and instanceof Operator
- The equals method
- The ArrayList Class
- The MyStack Class
- The protected and final Modifiers

3 Abstract Classes and Interfaces

- Abstract Classes and Abstract Methods
- The abstract Calendar class and its GregorianCalendar subclass Interfaces
- Define an Interface
- Omitting Modifiers in Interfaces
- The Comparable Interface
- Writing a generic max Method
- The Cloneable Interface
- Shallow vs. Deep Copy
- Interfaces vs. Abstract Classes
- Conflicting interfaces
- Wrapper Classes: The Number Class and subclasses
- BigInteger and BigDecimal
- The Rational Class

4 Generics

- To know the benefits of generics
- To use generic classes and interfaces
- To declare generic classes and interfaces
- To understand why generic types can improve reliability and readability
- To declare and use generic methods and bounded generic types
- To use raw types for backward compatibility

- To know wildcard types and understand why they are necessary
- To convert legacy code using JDK 1.5 generics
- To understand that generic type information is erased by the compiler and all instances of a generic class share the same runtime class file
- To know certain restrictions on generic types caused by type erasure
- To design and implement generic matrix classes

5 List, Stacks, Queues, and Priority Queues

- To explore the relationship between interfaces and classes in the Java Collections Framework hierarchy.
- To use the common methods defined in the Collection interface for operating collections.
- To use the Iterator interface to traverse the elements in a collection.
- To use a for-each loop to traverse the elements in a collection. To explore how and when to use ArrayList or LinkedList to store elements.
- To compare elements using the Comparable interface and the Comparator interface.
- To use the static utility methods in the Collections class for sorting, searching, shuffling lists, and finding the largest and smallest element in collections.
- To distinguish between Vector and ArrayList and to use the Stack class for creating stacks.
- To explore the relationships among Collection, Queue, LinkedList, and PriorityQueue and to create priority queues using the PriorityQueue class.
- To use stacks to write a program to evaluate expressions

6 Map and Set

- Set: The Basics
- HashSet
  - Creation
  - Method + add()
  - Enhanced for loop
  - forEach()
  - Other Common Methods
    - remove(), size(), contains(), addAll(), removeAll(), retainAll()

- LinkedHashSet
  - LinkedHashSet Basics
  - Example
- TreeSet
  - TreeSet Basics
  - Creation
  - add()
  - Common Method in SortedSet
    - SortedSet: first(), second(), headSet(), tailSet()
    - Navigable: lower(), higher(), floor(), ceiling(), pollFirst(), pollLast()
- Performance of Sets and Lists
- Maps
  - Map Basics
  - Three Types
  - Creation
  - Method

8 Developing Efficient Algorithm

- To estimate algorithm efficiency using the Big O notation
- To explain growth rates and why constants and non-dominating terms can be ignored
- To determine the complexity of various types of algorithms
- To analyze the binary search algorithm
- To analyze the selection sort algorithm
- Polynomial complexity
- To analyze the insertion sort algorithm
- To analyze the Tower of Hanoi algorithm
- To describe common growth functions (constant, logarithmic, linear, log-linear, quadratic, cubic (polynomial), exponential)
- To design efficient algorithms for finding Fibonacci numbers using dynamic programming

- To find the GCD using Euclid's algorithm
- To finding prime numbers using the sieve of Eratosthenes
- To design efficient algorithms for finding the closest pair of points using the divide-and conquer approach
- To solve the Eight Queens problem using the backtracking approach
- To design efficient algorithms for finding a convex hull for a set of points

9 Sorting

- To study and analyze time complexity of various sorting algorithms
  - To design, implement, and analyze **bubble sort**
  - To design, implement, and analyze **merge sort**
  - To design, implement, and analyze **quick sort**
  - To design and implement a binary **heap**
  - To design, implement, and analyze **heap sort**

10 Graphs and Applications

- To model real-world problems using graphs
  使用图形对现实世界的问题进行建模
- To describe the graph terminologies: **vertices (nodes), edges, directed/ undirected, weighted/unweighted, connected graphs, loops,parallel edges,simple graphs,cycles,subgraphs** and **spanning tree**
  描述图术语：顶点（节点）、边、有向/无向、加权/未加权、连通图、循环、平行边、简单图、循环、子图 和 生成树
- To **represent vertices and edges** using **edge arrays**, **edge objects**, **adjacency matrices**, **adjacency vertices list** and **adjacency edge lists**
  使用 **边数组、边对象、邻接矩阵、邻接顶点列表** 和 **邻接边列表** 表示顶点和边
- To model graphs using the **Graph** interface and the **Unweighted Graph** class
  使用 **Graph** 接口和 **Unweighted Graph** 类对图形进行建模
- To design and implement **depth-first search**
  设计和实现 **深度优先搜索**
- To design and implement **breadth-first search**
  设计和实现 **广度优先搜索**

11 Binary Search Tree

- To **design and implement a binary search tree**
  设计和实现二叉搜索树
- To **represent binary trees using linked data structures**
  使用链接数据结构表示二叉树
- To **insert an element** into a binary search tree
  在二分查找树中 **插入元素**
- To **search an element** in binary search tree
  在二叉查找树中 **查找元素**
- To **traverse elements** in a binary tree
  在二叉树中 **遍历元素**
- To **create iterators** for traversing a binary tree
  要 **创建迭代器** 以遍历二叉树
- To **delete elements** from a binary search tree
  从二叉搜索树中 **删除元素**
- To **implement Huffman coding** for compressing data using a binary tree
  实现霍夫曼编码，使用二叉树压缩数据

12.1 AVL Tree

- To know what an **AVL tree** is
  了解 **AVL树是什么**
- To understand how to **rebalance** a tree using the **LL rotation**, **LR rotation**, **RR rotation**, and **RL rotation**
  了解如何使用**LL旋转**、**LR旋转**、**RR旋转**和**RL旋转**重新平衡树
- To know how to design the **AVLTree** class
  如何**设计 AVLTree** 类
- To **insert** elements into an AVL tree
  在AVL树中 **插入个元素**
- To implement node **rebalancing**
  实施节点 **再平衡**
- To **delete** elements from an AVL tree
  从AVL树中**删除个元素**
- To implement and test the **AVL Tree** class
  实现和测试AVLTree类

- To analyze the **complexity** of search, insert, and delete operations in AVL trees
  分析AVL树中搜索、插入和删除操作的**复杂性**

12.2 Hashing

- To understand what **hashing** is and what hashing is used for
  了解什么是**哈希**以及哈希的用途
- To obtain the hash code for an object and design the hash function to map a key to an index
  获取对象的哈希码，并设计哈希函数将键映射到索引
- To **handle collisions (conflicts) using open addressing**
  使用 **开放寻址** 处理冲突
- To know the differences among **linear probing**, **quadratic probing**, and **double hashing**
  了解 **线性探测**、**二次探测** 和 **双重散列** 之间的差异
- To **handle collisions using separate chaining**
  要 **使用单独的链接处理冲突**
- To understand the **load factor** and the need for **rehashing**
  了解 **负载系数** 和 **重新哈希** 的必要性
- To implement **MyHashMap** using hashing
  使用哈希实现 **MyHashMap**