**1. [MC, 5 pts] Know about the three primary principles of OOP (Object Oriented Programming)**

- The three primary principles of OOP are encapsulation, inheritance, and polymorphism.

- **Encapsulation:** Encapsulation involves bundling data (attributes) and the methods (functions) that operate on that data into a single unit called a class. It helps hide the internal details of an object and provides a public interface to interact with the object.

- **Inheritance:** Inheritance allows a new class (subclass or derived class) to inherit properties and behaviors from an existing class (superclass or base class). It promotes code reuse and the creation of hierarchical relationships among classes.

- **Polymorphism:** Polymorphism allows objects of different classes to be treated as objects of a common base class. It enables dynamic method binding, where the appropriate method is determined at runtime based on the actual type of the object. This can be achieved through method overriding and interfaces.

**2. [MC, 5 pts] Know about sequential, selection, and repetition control statements.**

- Sequential control statements are executed in order.

- Selection control statements (like if statements) allow you to choose between different paths of execution.   
- Repetition control statements (like loops) allow you to repeat a block of code multiple times.

**3. [MC, 5 pts] Same topic as (2)**

- This question appears to be a duplicate of the previous one, referring to control statements.

**4. [MC, 5 pts] Know about the abstract and final keywords – what they mean, what they do in the context of methods and classes.**

- **The "abstract" keyword** is used to declare abstract classes and methods, which cannot be instantiated but must be subclassed. Abstract classes cannot be instantiated directly; they serve as blueprints for subclasses to implement. Abstract methods are declared without an implementation and must be overridden in concrete subclasses.

- **The "final" keyword** can be applied to classes, methods, or variables. In the context of methods, it means the method cannot be overridden. In the context of classes, it means the class cannot be extended.

**5. [MC, 5pts] Know about the ADTs Bag, Stack, Queue, Deque, and Priority Queue**

- These are abstract data types (ADTs).

- A **Bag** is a collection of items with no specific order. It allows for adding and removing items without regard to their placement within the collection.

- A **Stack** is a collection with a last-in, first-out (LIFO) order. Elements are added and removed from the top of the stack.

- A **Queue** is a collection with a first-in, first-out (FIFO) order. Elements are added at the rear and removed from the front.

- A **Deque** (Double-Ended Queue) allows adding and removing elements from both ends. A Deque is a hybrid data structure that allows adding and removing elements from both ends, making it more versatile than a simple stack or queue.

- A **Priority** **Queue** is a collection where elements are ordered by priority. Elements with higher priority are retrieved first.

**6. [MC, 5pts] Know about the Big O notation, and the different categories of computational complexity – especially the primary ones discussed in class / documentation provided.**

- **Big O Notation:** It is used to describe the upper bound of the time complexity of an algorithm. It provides a way to analyze how the runtime of an algorithm grows with the size of the input data.

- Big O notation is used to describe the **upper bound of the time** **complexity** of an algorithm. Primary categories include O(1) (constant time), O(log n) (logarithmic time), O(n) (linear time), O(n log n) (linearithmic time), O(n^2) (quadratic time), and O(2^n) (exponential time), among others.

**7. [MC, 5pts] Given code, be able to find the Big O**

- You can analyze code to determine its time complexity using Big O notation. The Big O represents the upper bound of how the code's runtime scales with input size.

**8. [MC, 5pts] Same topic as (7)**

- This question appears to be a duplicate of question 7, focusing on analyzing code and determining its Big O notation.

**9. [TF, 5pts] Know about the ADTs Bag, Stack, Queue, Deque, and Priority Queue**

- This is a true/false question related to knowledge about abstract data types (ADTs) like Bag, Stack, Queue, Deque, and Priority Queue.

**10. [TF, 5pts] Know about how inheritance, superclasses, subclasses, and polymorphism work, as well as fundamental syntax involved.**

- This is a true/false question related to knowledge about inheritance, superclasses, subclasses, and polymorphism in object-oriented programming.

**11. [MC, 5pts] Know how to convert infix to postfix using the technique demonstrated in class.**

- Converting Infix to **Postfix:** Converting an **infix** expression (e.g., "2 + 3") to postfix (also known as Reverse Polish Notation) involves using a stack data structure to manage operators and operands. The algorithm typically employs rules to handle operator precedence and associativity.

**12. [MC, 5pts] Same topic as (11)**

- This question appears to be a duplicate of question 11, focusing on the conversion of infix to postfix expressions.

**13. [MC, 5pts] Know about generics, specifically the ArrayList**

- **Generics** allow you to create classes, interfaces, and methods that operate on type parameters. The ArrayList is a generic class in Java that can store elements of a specific type.

- Generics in Java allow you to create classes, interfaces, and methods that can operate on type parameters. This enables you to write code that is type-safe and can work with different data types without casting. The ArrayList is an example of a generic class that can store elements of a specified type.

**14. [MC, 5pts] Know about the compareTo method and the Comparable interface, how they are used, etc.**

- The `compareTo` method is used to compare objects in Java, and the `Comparable` interface is used to make objects comparable. This is often used in sorting and ordering objects.

**15. [MC, 5pts] Know about recursions – including recursive cases, base cases, etc.**

- **Recursion**: Recursion is a programming technique where a function or method calls itself to solve a problem. It involves two important components:

- **Recursive Cases:** These are situations where the function calls itself with a modified input to make progress toward a base case.

- **Base Cases:** These are stopping conditions that prevent the recursion from continuing infinitely.

**16. [MC, 5pts] Given syntax involving exceptions and exception handling, be able to determine what is run in the code and what is skipped. Specifically make sure you know what the try, catch, and finally statements do.**

- In exception handling, the "try" block contains the code that may throw an exception.   
 - The "catch" block handles the exception  
 - The "finally" block contains code that always executes, whether an exception is thrown or not.

**17. [MC, 5pts] Know about the different implementations of the ADT Bag, and the complexities of the different methods, depending on the implementation (e.g., Array, Linked-chain)**

- Different implementations of the Bag ADT can have varying complexities for methods like adding and retrieving elements, depending on whether they use arrays, linked lists, or other data structures.  
 - Different Implementations: The ADT Bag can be implemented using various data structures such as arrays, linked lists, or other containers. The choice of implementation impacts the complexities (e.g., time and space) of different methods like insertion, deletion, and retrieval.

**18. [MC, 5pts] Know about the methods and behavior of the ADT Stack.**

- The Stack ADT involves methods like push (to add an element), pop (to remove the top element), and peek (to view the top element), and follows the Last-In, First-Out (LIFO) principle.  
 - Stack Methods: The Stack ADT typically provides methods like:

* push(): Add an element to the top of the stack.
* pop(): Remove and return the element from the top of the stack.
* peek(): Return the element at the top of the stack without removing it. Follows the Last-In, First-Out (LIFO) principle.

**19. [MC, 8pts] Know about Wrapper classes and generics (such as the ArrayList).**

- Wrapper classes are used to wrap primitive data types as objects. Generics, like ArrayList, allow you to work with type-safe collections, and wrapper classes can be used with generics to store primitive data types in collections.  
 - Wrapper Classes: Wrapper classes are used to wrap primitive data types (e.g., int, float) as objects in Java. They allow primitive types to be used in generic classes and collections, as generics require objects. For example, Integer is a wrapper class for int.

- Generics with Wrapper Classes: You can use wrapper classes with generics, such as ArrayList<Integer>, to store primitive data types in a type-safe collection. This eliminates the need for manual boxing and unboxing.

**20. [MC, 2pts] Mystery question.**