Questions on the Exam - 9 Questions:

1. Testing
    a. Provide Values that will cover all the cases of the code. Coverage 100%. Example from the Sample Term Test.
2. JavaDoc
3. Recursion
    a. Write a recursive method by hand (Similar to Programming Test 1)
    b. Tracing Tree of a recursive method (Similar to the Written Test)
4. Time Complexity
    a. Find the Time Complexity for a piece of code
    b. Given the polynomial function, provide the constant c and n_0 such that $T(n) < cF(n) \mid n>n_0$.
5. Inheritance & Polymorphism
6. Encapsulation & Exception
7. Composition & Aggregation
8. Abstract Classes & Interfaces
9. Generics

- Chapter 6 - Inheritance:
    - When two (or more) classes share attributes and methods, an “is-a” relationship is created.
    - If no superclass constructor is invoked explicitly, then the superclass’s no-arg constructor super( ) is invoked automatically as the first statement of the extended class’s constructor.

- Chapter 7 - Polymorphism:
    - Polymorphism has two types:
        - Compile Time
        - Run Time
    - **Overloading**: Same class has multiple methods with the same name and different signatures.
    - **Overriding**: Subclass has same method with the same signature as the superclass with different implementation
    - Type Casting:
        - Widening (up casting) → Automatic:
            - Object shape = new Shape();
            - Person p = new Student();
        - Narrowing (down casting) → Explicit/forced:
            - Shape p = (Shape) object;
            - Student s = (Student) person;

- Static Methods with Polymorphism:
  - Animal a = new Dog();
    a.eat()
  - eat is a static method so the compiler will see it as: Animal.eat() and not Dog.eat() because a is of type Animal during compile time. Hence the eat method will be called through Animal and not Dog.
  - NO LATE BINDING WITH STATIC METHOD
- Instanceof method:
  - Only Allows object to be compared to the same superclass.
  - Can be used to give a boolean when comparing an object with the class type to know if a certain object is an instance of a certain class.
  - Animal a = new Animal();
    a instanceof Shape // **ERROR**
    a instanceof Animal // True
    a instanceof Dog // False
- With polymorphism a method‘s behavior changes depending on theobject calling it. ONLY IF THE METHOD IS NOT STATIC

- Chapter 8 - Exception Handling:
  - When an exception is not caught, no value is returned.
  - Types of Exceptions:
    - Checked Exception: Checked at compile time by the compiler.
    - Unchecked Exceptions: Runtime Exceptions
      - ArithmeticException
      - NullPointerException
      - IndexOutOfBoundsException
      - IllegalArgumentException
  - Exceptions should be ordered from most precise to the most general exception.
  - If an overridden method throws an exception, the super method also must throw the same or higher level of exception.
  - ToString():
    - Original implementation returns the address of the object in the hash table.
  - Equals():
    - Original implementation checks if the two objects have the same reference address.
    - Requirements:
      - Equality will null is false
      - Reflexive
      - Symmetric
      - Transitive
    - Parameter is always of type Object

- HashCode():
  - Whenever equals is overridden, hashCode should be overridden as well.
- compareTo & equals methods:
  - Those methods should follow those 3 rules: Reflexive, Symmetric & Transitive
  - Reflexive: aRa
  - Symmetric: aRb → bRa
  - Transitive: aRb & bRc → aRc

- Chapter 9 - Abstract Classes and Interfaces:
  - It is possible to take a private method and override it with a public method in a subclass. The opposite is not true. You can't override a public method with a private method.
  - Abstract Classes:
    - Abstract classes cannot be instantiated.
    - The constructor of an abstract class MUST be called by all subclasses (unless the subclass is also abstract).
  - Abstract method:
    - does not have code in it.
    - All the concrete subclasses **should** implement all abstract methods
    - If a method is abstract, the class in which the method is defined must be abstract.
    - Abstract method cannot be **private** as it should be inherited by another class.
    - Abstract method cannot be **static** because we can't override a static method. The static method should belong to the class and can't be overridden. Abstract method doesn't belong to the class and should be overridden → Contradiction.
    - Abstract method cannot be **final** because they should be overridden. Final methods can't be overridden.
  - Interfaces:
    - No constructor for interfaces.
    - All methods are abstract except static or default.
    - Default methods provide implementation and don't have to be overridden by the classes that implements them.
    - All variables in an interface have to be **public** (accessible to subclasses), **static** (belongs to the class) and **final** (cannot be changed).
  - Comparable:
    - The comparable method requirement are Symmetry and transitivity.
    - Symmetry is if obj1.compareTo(obj2) < 0 then obj2.compareTo(obj1) > 0.
    - Transitivity is if obj1.compareTo(obj2) < 0 and obj2.compareTo(obj3) < 0 then obj1.compareTo(obj3) < 0.

- Chapter 10 - Generics:
  - If T1 **IS-A** T2, then SomeClass<T1> **IS-NOT** SomeClass<T2>.

        - Meaning-Example: Circle **is a** subclass of Shape but List<Circle> **is not a** subclass of List<Shape>
    - Generics Types are **invariant** which means that generic<S> **is not** generic<T> where S is a subtype of T.
    - Arrays are said to be covariant, which means that an array of type S[] is an array of type T[], where S is a subtype of T.
    - Type Parameter Naming Convention
        - E: Element type in a collection
        - K: Key type in a map
        - V: Value type in a map
        - T: General Type
        - S, U: Additional general types
    - Generic Methods are often Static
    - <T extends Shape> **RELAXED:**
        - T Could be any subclass of class Shape (including Shape).
    - <T super Shape> **IMPOSED:**
        - T Could be any parent class of class Shape (including Shape).
    - Difference between sList<Shape> and sList<? extends Shape>:
        - sList<? extends Shape>: binds to a particular Shape subtype and allows ONLY that. It might store only Rectangles but not Circles
        - sList<Shape>: allows anything that is a subtype of Shape in the same list. It could store both Rectangles and Circles

- Chapter 11 - ADT & Collections
    - Sets can only be looped through a **for each** statement because the elements don't have an index.
    - Iterator class allows us to create a for each loop on any type of collection
        - Iterator<ObjectType> iterator = varname.iterator();
          while (setIterator.hasNext())
          {
                System.out.println(iterator.next());
          }
    - Search Algorithm
        - Linear Search (unsorted): Loop through every element.
        - Binary Search (sorted): Go the middle and move depending on the value.
    - Set Operations:
        - set1.addAll(set2): Union between set1 and set2 into set1.
        - set1.retainAll(set2): Intersection of set1 and set2 into set1.
        - set1.removeAll(set2): Difference between set1 - set2 into set1 → set1 = set1 - set2
- Other Information:

- State of an object: The values of the fields of the objects.
- Purpose of the no-argument constructor: Initialize the state of an object to a well-defined default state.
- The return part itself is not part of the method signature
- An obligatory method is a method that the implementer must override.
- Method signature doesn’t have a return type
- x.compareTo(y) returns a postive integer if x is greater than y