Software Development

Michael Rao

2022-04-04

Contents

1 Principals of Object Oriented Programming

Java will be the language of choice for the OOP Principals

1.1 Abstraction

Abstraction is the idea of reducing objects to its essence. In Java, this is achieved through **interfaces** and **abstract classes**

```
Interface:
* 100% abstraction
* Implemented using the "implements" keyword
* Interface can extend another Interface using keyword "extends"
* Class can implement multiple interfaces
* All variables declared are final

Abstract Class:
* Not full abstraction as it allows Concrete Methods
* Can have Abstract Methods that are declared without implementation
* Can have both Abstract Methods and Concrete Methods
* No Objects allowed in Abstract Class
* Class can extend only one abstract Class
* Can have variables that are not final
```

1.2 Encapsulation

Encapsulation is the idea of information hiding. In Java, this is achieved through Mutators and Accessors and using Access Modifiers like **Public**, **Private** and **Protected**.

1.3 Inheritance

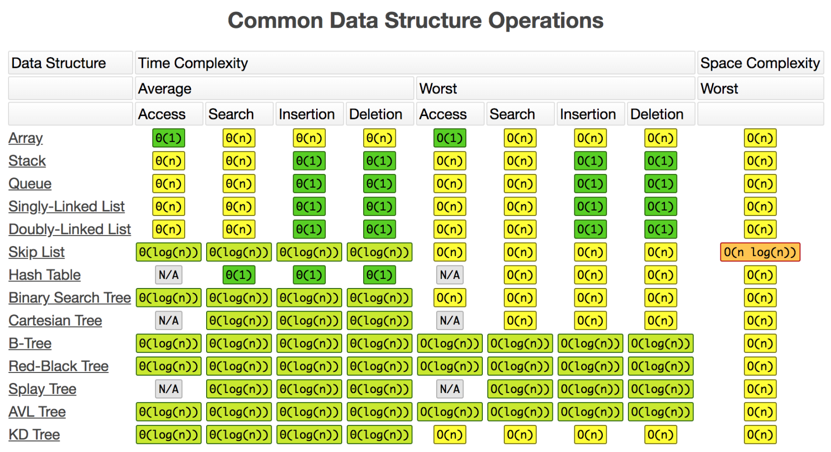
Inheritance in Java is when you extend from one class to another, allowing for the subclass to inherit all the methods in the parent class. This code design methodology provides great code re-usability and opens the door to Dynamic Polymorphism.

1.4 Polymorphism

Polymorphism in Java is the concept where one method can take multiple forms. From the idea of inheritance, overriding a method in the subclass allows it to be polymorphic to the parent class.

Java allows multiple polymorphism but not for **Dynamic Polymorphism**, e.g. overriding. Multiple polymorphism is allowed only for **Static Polymorphism**, e.g. overloading. Dynamic = Run-Time. Static = Compile-Time.

2 Data Structure Space and Time Complexity



Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

2.1 Hashmap

2.1.1 Why O(n) worst case?

Hashmaps are dependent on **Hash Functions** to determine how efficient they are.

For example, if we have a Hash Function as follows: $f(x) = 1$ where x is the input value and right side of the equation is position to store x. This will cause the **Hashmap** to effectively become a **Linked List**, which would make **Search, Insert, Delete** into time complexity of O(n).

3 Conceptual Comparisons

3.1 Heap Memory vs Stack Memory

#	Stack	Heap
1	Physical space in RAM	Physical space in RAM
2	Stores local variables and methods	Stores objects and JRE classes
3	Allocated by thread-level at run-time	Global memory shared across all threads
4	Memory Allocation taken care of by Compiler	Memory Allocation taken care of by Developer and Garbage Collector

3.2 Process vs Thread

#	Process	Thread
1	Collection of code, memory and other resources that make up a program	sequence of code within a process
2	Data is not shared between processes	Shares data and code segments with other threads but each contains its own local registers
3	Heavyweight	Lightweight

3.3 Run-time vs Compile-time

#	Compile-Time	Run-Time
1	Period when code is converted to binary	Period when program is running
2	Static but Fast (Overloading)	Dynamic but Slower (Overriding)

3.4 Overload vs Override

#	Overload	Override
1	Same method name with different parameters	Method Signature is same between subclass and parent class
2	Used to implement Compile-time Polymorphism	Used to implement Run-time Polymorphism
3	Used on constructors to create new Objects given different amounts of data	

4 Algorithm Comparisons

4.1 Merge Sort vs Quick Sort

#	Merge Sort	Quick Sort
Big O	Worst Time Complexity: O($nlogn$)	Worst Time Complexity: O(n^2)
Big O	Space Complexity: O(n)	Space Complexity: O(1)
Usage	Ideal for Arrays	Ideal for Linked Lists

Merge Sort cannot be worse than O($nlogn$) since Merge Sort is not dependent on pivot.

Quick Sort can have worst case of O(n^2) since it is possible the pivot is always the smallest or biggest.

4.2 Tree Traversals

#	Ordering	Usage
Inorder	Left SubTree -> Root -> Right SubTree	Inorder on BST always produces an ascending sorted list
Preorder	Root -> Left SubTree -> Right SubTree	Used to create copy of tree & get prefix expression on expression tree
Postorder	Left SubTree -> Right SubTree -> Root	Used to delete tree & get postfix expression on expression tree

4.3 DFS vs BFS

#	DFS	BFS
Recursive	Can be done	Not possible
Iterative	Use a Stack	Use a Queue

5 Behavioral Interview Questions

5.1 In your opinion, what are some principles every software engineer should follow?

Keep things simple and straightforward. Keep your mind to one task at a time, so each task is given the care it deserves.

5.2 Do you prefer to work alone or with others?

Software Developers exist to solve problems, I typically try to solve them myself first to test my own abilities but always welcome the help of others.

5.3 Can you give me an example of how you set goals for yourself?

As a Software Developer, it is important to constantly grow your skill set and keep that skill set sharp and fresh. When I joined Morgan Stanley, my team's job was to create applications that reduce manual efforts from our clients. So we had Database, Frontend and Middle-Tier. my tasks were to manage the Database, write Stored Procedures. However, at that point in time, my team was also low on Front-end Developers. So I took the opportunity to learn the core code and with my background in Java, I became a backend and frontend developer to ensure my team met deadlines. Not shortly after, my team's Middle-Tier developer went on vacation for 4 weeks, so we were then short on Middle-Tier developers. Again, I took the opportunity to study the basics of the Middle-Tier and became a Full Stack Developer for my team.

6 Management Interview Questions

6.1 Strength and Weaknesses

- **Weakness** - When I joined Morgan Stanley, one of my biggest weaknesses was my communication. After leading projects, software developers and communicating on a daily basis with business users. I've definitely improved a lot, but I still feel I have a lot of space to improve on communication.
- **Weakness** - I have an issue working with ambiguity. I have a need to know the ins and outs of a project and the requirements and why the project is necessary and why the requirements are the way they are.
- **Strength** - I am a fast learner. I was not attuned with any part of the technology stack used in Morgan Stanley outside of Java and Git and was able to quickly become the go-to senior full-stack developer and leading the team for automation.
- **Strength** - I am very open to changes. Technology evolves frequently and there are many ways to solve a problem, it's extremely important to keep an open mind.

6.2 What do you expect from a manager?

My manager at Morgan Stanley was the perfect manager. She was open minded, I always felt comfortable going to her about any concerns regarding the timelines or projects we were working on. She wasn't the most technically skilled person on the team but she was balanced in her knowledge in that she was able to properly communicate to business, technology and management. That is what I expect from a manager.

6.3 What was it like working for your manager?

I've yet to have a difficult manager. For my manager at Morgan Stanley, I always went to her with any queries or doubts I had on the time line or the projects. We designed many projects together to ensure maximum efficiency and delivery. We were very open and communicative with each other.

6.4 What strategies would you use to motivate your team?

My manager and I used to go on a weekly coffee break, trying our best to just talk about random things on our mind that normally do not come up during more strenuous situations. In addition, it is important to make sure the developers are as stress-free as possible. The most difficult part of being a Software Developer is not coding but designing and finding the optimal solution to each problem. In order to accomplish that, you can motivate the team by just giving them the space and time they need to optimize their development skills.

6.5 How would you address conflicts with another coworker?

Honestly I have never had any conflicts with any of my coworkers, so I don't really have any firsthand experience. If I was to have some conflict in the future, the most important thing is communication. So first thing would be to just get some time with them and just have an uninterrupted 1-on-1, be patient and listen to their perspective and smooth out any edges and any misunderstandings or any miscommunications.

6.6 How would you address productivity issues with your team?

I believe a software developer under too much stress is an ineffective software developer. Some people perform well under stress and some don't, but you don't take a 50/50 chance when you have deadlines. I'm focusing on stress here because if the productivity issue is stemming from an unfamiliarity with the technology, it can simply be accelerated by having a more senior developer providing some guidelines and pointers and knowledge transfers, etc. But when it comes to stress, it's important to help your team de-stress, the main thing here is again communication. Be open with your team, be transparent, listen to your teammates, provide whatever types of support they need.