Java Developer & Indian IT Industry

Morgan--RBS--UBS---Citibank---MarkIt---IonTrading--BlackRock
My skills my job !!

Cultivate skills to create your own Path ...

This e-book covers following topics

1. Core Java as of Java 6
2. Concurrency and Java Collections Framework
3. Algorithms, Data structures and Puzzles
4. Object Oriented Design Problem for the open discussion
5. Sample Interview Questions

About Author
Munish Chandel

Munish is currently working as a Java software developer for an investment bank in India.

Linked In Profile
http://linkedIn.com/munish.chandel

1st edition, 2013

Java Developer & Indian IT Industry

This book attempts to address common problems faced by a Java Developer in Indian IT Industry

Topics Covered In This Book

(Core Java, Algorithms, Data Structures, Puzzles & Concurrency Problems)

Preface

This work is my sincere effort to consolidate solutions to some basic problems faced by my fellow mates in their day to day work.

This Book Isn't

- A research work, neither it is intended to be.
- Of much help to a fresher in IT industry as it expects some level of hands on experience. It doesn't even cover all the topics required by a newbie to start developing software from scratch.
- A reference book, one time read should be enough.

This Book Is

- Collection of excerpts discussing the common problems faced by an experienced Java Developer in his day to day work. The intent is not to provide with the concrete solution to a given problem, but to show the approach to get the problem solved. And there could definitely be more efficient ways to solve the given problem compared to what is mentioned in this book. The approach shown here is limited to the knowledge of the author.
- Collection of topics in core Java, OOD, algorithms, data structures & Puzzles.

Who should read this book ?

- Experienced candidates who are appearing for Java interviews specifically in investment banking domain (having approach for enterprise level applications)
- Experienced Java developers who want to brush up their skills to solve their day to day software problems.

What does this book cover ?

See Index for the coverage of topics

What this book does not cover ?

- It does not cover much stuff related to frameworks like Spring, Hibernate, Struts, EJB's, etc
- It does not cover much of database stuff like SQL & PLSQL.
- It does not provide with the concrete solution, though the authors tried his best to provide you with the most relevant pointers to the solution.

I hope this book adds value to your skills. Your reviews will be most valuable to me.

Munish Chandel
cancerian0684@gmail.com
http://linkedIn.com/munish.chandel
January 2013

This page is Intentionally left blank.

Contents

IT Industry and Psyche of Developers

My Opinion in Indian Context

Question : What is overall picture of Indian IT Industry ?
IT is outcome of Intellectual Minds from western civilization in modern era. The knowledge is still sourced from west in this domain east at the time of this writing. The main reason for outsourcing IT related work to India is economically cheap labour, though the trends are changing now. Typically there are two types of companies in India -

1. Product Based (Adobe, Google, Amazon, Oracle, etc)
2. Services Based (TCS , Infosys, Cognizant, Wipro, HCL, Sapient, etc) - Broker Companies

There is a big cultural and compensation differences in both of these types. Product based company provides better culture for the employee in terms of personal satisfaction. Compensation offered in Product based companies is higher than Service based companies, but service based companies compensate this by short/long term on site opportunities.

Question : What type of work people do in India IT Industry?
We can categorize typical IT work into two types - Green Fields and Maintenance & Support.
Most people in Indian IT industry work for services based companies where major fraction of work is of Maintenance & Support type. Even most product based companies build their basic block outside and then outsource work to India. Roughly speaking 80-90% people should be working for maintenance & support type projects and rest working for Green Field projects. You could find exceptions to my words at some places.

Question : What causes a typical developer to switch his/her job so frequent, Is that bad, Why that is not the case in best ?
A thought to switch Job comes to one's mind when one finds blockage in growth opportunities in their current organization. I could list few reasons for the same -

1. Salary disparity and scope for higher salaries in new company is the major reason for job switch. Most service based companies tries to maximize profit on their side by offering lower salaries to its employees (anything starting from 5000 $ a year in typical service based company), but as people acquire more skills they switch for higher roles in new company. Demand and supply typically governs the salaries in India.
2. The quality of work is another reason for switch. Work quality directly relates to stress (more manual work more stress)
3. Shift Timings and location preference also causes people to switch their jobs
4. Very few people switch their job because they are unable to meet expectations of their company or because they got an unexpected work.

To some extent this switch is fair because we can't expect someone to work for a company 10,000 $ a year for his lifetime (considering the increments that we currently get), isn't ? As the Industry will mature, job shift will reduce. Moreover, IT companies are not supposed to hinder one's growth in current Caste System.
The IT companies in west are mature, salaries are already saturated, people don't take much work stress, So western employes do not find many reasons for their Job switch.

Question : What is growth Pyramid of a typical Developer in Indian IT Industry ?
There are two main streams in It companies - Technical & Management. Most Indian people start their carrier as a software developer/qa, but over the time they shift to Management Side (60-70%). At least that is the trend in Service Based companies. This trend has started changing now, because of the saturation on number of jobs in management side. So people will experience longer stretch working as a developer in the coming decade. Employees working for startup and product based companies, tend to remain on technical side, because Individual Contribution gives them better growth opportunities.

Question : What is typical psychology of an average Indian Developer ? What kind of chaos pollute their mind ?
Most Indian opt for IT, not by choice but for money, because of large unemployment in India. Most others think that earning money in IT industry is easy and effortless compared to other opportunities. A great proportion wants to take IT as the jumping ground for their higher studies (MBA, MS, etc). An average fresher is polluted with the thoughts about his career development, and is unsure about his interests in IT field, trying various alternates in first few years.

Question : What is the Problem with Most Indian Developers in terms of Skills ?
Majority of IT crowd does not have good hold over their primary skills which are required for the work. The underlying cause for the faded skills is the type of Work which is fed to India. The majority of work does not require high quality of skills on developer's part. But still people can learn by their own, build their skills and fight for better quality work. One should have a very good hold over his primary skill set and look for work which is matching those skills, instead otherwise.

Question : What are advantages of acquiring skills ?
There are many. Listing few of them -
1. Very good understanding the basic computer science along the core language skills helps write very efficient software applications which can utilize the available hardware effectively, with minimum bugs.
2. Skills alleviates tension, because only skills level help us automate the mundane tasks.
3. We are on the better side of the crowd and thus remain positive minded.
4. Skills help us estimate the time for software development effectively.
5. When you have good understanding of hardware and software then you get a deep penetration into software development which otherwise is not possible.

Question : Would it help if I memorize all the questions ?
No, it will not. But if you memorize the most common **Patterns of software development,** that will definitely help to ease day to day work. A single pattern resolves n number of problems emerging from that pattern, and we should always look forward finding the patterns instead of solution to a particular problem.

Question : Why do interviewers ask rocket science questions in interviews even if the new role does not require any such skills ?
Hiring in IT industry is not regulated by any means, it is solely up to the interviewer to choose the topic for discussion by the interviewee. In today's intellectual world, people like intellectual war, and interview is a good place for that. I do not find any harm by such interview process unless interviewer hides the real picture of work that one needs to perform after joining the new role. For sure there is one plus point to such interview process that it will definitely raise our skill set.

Question : Why people take so many offers at the time of Job change, doesn't it add to chaos to the system ?
The main reason for doing so, is the disparity between work and salary across the companies, and this will continue to happen till a saturation point come in India. People feel insecure at financial level and try their best to grab the most paying Job opportunity, and that's fair. On the other hand, companies tend to maximize their profit by limiting the salary offer as per individual's previous company's salary. So it is a game, where both the employer and the employee are fighting to maximize their own profit. Ultimately, the Demand and Supply equation balances the fight between employer and the employee.

Java Developer Road Map

A deep understanding of basic software components is must for a developer who wants to write Scalable Enterprise/Internet Applications from scratch using Java. Below is the matrix showing a hierarchy of skills which are required for a senior software developer who are aiming a long term carrier in Software Development.

Priority	Category	Topics
6	**Web Tier**	Servlet basics, HTTP Basics, REST, MVC Design Pattern, Struts & Spring
5	**Database**	SQL, database Indexes, JPA/Hibernate QL, Table to Entity Mapping, Inheritance in JPA (Table per class, joined, single table), Transaction Isolation Level, embeddable, Mapped Super Classes, All relationships - OneToOne, OneToMany, ManyToMany.
4	**Core Java**	Inheritance, Generics, Garbage Collector, good hold over Concurrency (synchronizer, non-blocking algorithms using atomic package, executor service, Java Memory Model, Immutability, volatile, Fork/Join), good understanding of internals of Java Collections Framework (HashMap, LinkedList, ConcurrentHashMap, PriorityQueue, TreeMap)
3	**Algorithms**	Tree Traversal, Tree Insertion, Tree balancing using left & right rotation, Sorting (quick, merge, external, selection), Binary Search, BFS, DFS, Topological Sorting using Graph, Dijkstra's algorithm
2	**Data Structures**	ArrayList, LinkedList, Stack, Queue, Tree (Binary Search Tree, Red Black Tree, Binary Heap), Hashtable, Prefix Tree (Trie), Suffix Tree, Graphs.
1	**Concepts**	Object Oriented Programming & Design Test Driven Development (JUnit, Mockito, etc) Familiarity with Version Control System, Continuous Integration Design Patterns (Abstract Factory, Builder, Singleton, Observer, Template, Strategy, Visitor, Decorator, Facade, Flyweight, etc) Memory (Heap, Stack, Memory Generations in Java) Logarithm, Big-O notation, Bitwise Manipulation & Recursion Number System (2's complement, binary, hexadecimal, etc)

Skills Matrix for a Java Developer

Chapter 1

Concepts

Question : What are the Pros and Cons of Java (as of JDK 1.7)?

SOLUTION

Java Pros

It's free of cost
Open source with quite large community base
Lots of available third party libraries & frameworks
Platform independent, works on most modern platform (Unix, Windows, Mac)
Supports Object Oriented Programming
In built support for multi-threading, It can very efficiently utilize maximum of given hardware (Threads, Fork/Join, non-blocking algorithm using CAS, etc)
Very good support for Internationalization
Memory management is automatic by using garbage collector
Pure Java Byte code running on 32 bit jvm works perfectly fine on 64 bit platform
Its Improving year by year

Java Cons

Is not a good fit for desktop applications because of heavy memory footprint and huge vm startup time
Normal Java Is not good for real time systems because of "stop the world garbage collector pauses".
Memory foot print is large compared to C++
Does not provide very good support for functional programming as of JDK 1.7

Notes

Difference between 32 bit and 64 bit JVM

The Java language specifications are same for both the platform i.e. int will remain to be 4 bytes signed two's complement, char will remain single 16-bit Unicode, long will remain 64-bit signed two's complement, and so on. Hence any Pure Java code will not see any difference provided external native calls are not used. All that changes is the amount of addressable memory (good) and the amount of memory per Object (not that good). The size of the reference variables doubles from 32 bit to 64 bit, thus all the reference variable will take double the size when running on 64 bit JVM.
Theoretically, there are no class file differences between code compiled with the 32 bit and 64 bit versions of the same revision of Java.

Question : What are four principles of OOP ?

SOLUTION

There are 4 major principles that make an language Object Oriented. These are Encapsulation, Data Abstraction, Polymorphism and Inheritance.

Encapsulation

Encapsulation is the mechanism of hiding of data implementation by restricting access to public methods.

Abstraction

Abstract means a concept or an Idea which is not associated with any particular instance. Using abstract class/ interface we express the intent of the class rather than the actual implementation. In a way, one class should not know the inner details of another in order to use it, just knowing the interfaces should be good enough.

Inheritance

Inheritances expresses "is a" relationship between two objects. Using proper inheritance, In derived classes we can reuse the code of existing super classes.

Polymorphism

It means one name many forms. It is further of two types - static and dynamic. Static polymorphism is achieved using method overloading and dynamic polymorphism using method overriding.

What is aggregation, how is it different from composition[1] ?

Both of these are special type of association and differ only in weight of relationship.
Composition is stronger form of "is part of" relationship compared to aggregation "has a".
In composition, the member object can not exist outside the enclosing class while same is not true for Aggregation.

1 http://design-antony.blogspot.in/2007/07/aggregation-vs-composition.html

Question : What is Logarithm ? Why is it relevant in Software Development ?
SOLUTION

A logarithm[1] tells what exponent (power) is needed to make a certain number. In a way it is opposite of exponentiation. For example,

$\log_2 (8) = 3$	and	$2^3 = 8$
$\log_{10} (1000) = 3$	and	$10^3=1000$

Number	Logarithm (base 10)
1	0
10	1
100	2
1000	3
10000	4
100000	5

Why do we need Logarithm ?

- It is easy to handle small numbers compared to very large numbers, when our motive is just to compare them. Logarithm converts big values to small numbers.
- It makes multiplication and division of large numbers easy because adding logarithms is the same as multiplying and subtracting logarithms is same as dividing.

In pre modern era, when calculators were not there, logarithm tables were used for division and multiplication of large astronomical numbers.

Sum of logs = log of product
Subtraction of logs = log of division

Notes

Logarithm was discovered in India in ancient times around 2 BC and was used to express astronomical units. It is called as **Laghuganak** in hindi.

Logarithmic spirals are common in nature. Examples include the shell of a nautilus or the arrangement of seeds on a sunflower.

The Richter scale measures earthquake intensity on a base 10 logarithmic scale.

In astronomy, the apparent magnitude measures the brightness of stars logarithmically, since the eye also responds logarithmically to brightness.

1 http://simple.wikipedia.org/wiki/Logarithm

Question : What do you understand by Big O Notation, Why is it important in software development ?

SOLUTION

Big O Notation[1] is a mechanism used to measure the relative efficiencies of Algorithms in terms of space and time. It makes us understand how execution time & memory requirements of an algorithm grow as a function of increasing input size.

In this notation, O stands for the order of magnitude.
Following are the examples of Big O, in increasing order of their magnitude.

Big O Notation	Name	Example
O (1)	Constant-time	Searching from a HashMap, check a number for even/odd
O (log n)	Logarithmic	Find an item inside sorted array using Binary Search
O (n)	Liner	Printing all elements from an array
O (n log n)	Loglinear	Sorting using Merge Sort
O (n^2)	Quadratic	Bubble Sorting Algorithm
O (2^n)	Exponential	Shortest Path Problem Djigstraw Algorithm
O (n!)	Factorial	Solving Travelling sales man problem

Importance of Big O

We should always keep time efficiencies in mind while designing an algorithm using existing data structures, otherwise there could be sever performance penalties for using wrong data structure for a given scenario.

Notes

Time efficiency in Big O notation for few Java Collections

ArrayList (ignoring the time taken by array resize operation)
O(1) for add, size and get
O(n) for toString() method

PriorityQueue
O(1) for peek, element and size
O(log n) for offer, poll, remove() and add
O(n) for remove(Object) & contains(Object)

HashMap (with no collisions)
O(1) for get operation
O(1) for put operation

LinkedList
O(1) for removal
O(1) for add & poll method
O(n) for toString() method

1 http://en.wikipedia.org/wiki/Big_O_notation

Question: List down sorting algorithms by their time & memory complexity in Big O notation ? When do we call a sorting algorithm stable ?

SOLUTION

Sorting is an algorithmic technique to put all the collection elements in certain order.[1]

Algorithm	Average Time Complexity	Worst Time Complexity	Memory Complexity
Quicksort	n log n	n^2	log n
Binary Tree Sort	n log n	n log n	n
Merge Sort	n log n	n log n	n
Selection Sort	n^2	n^2	1
Bubble Sort	n^2	n^2	1
Heap Sort	n log n	n log n	1

When is a sorting algorithm Stable ?

An algorithms is said to be stable if it maintains the relative order of records where keys are equal. For example, suppose the following key-value pair need to be sorted based on key in ascending order
INPUT --> [(2,3) (1,2) (1,3) (3,1)]

Now there are two solution possible for the first two elements
OUTPUT1 --> [(1,2) (1,3) (2,3) (3,1)] --> stable sort because order is maintained
OUTPUT2 --> [(1,3) (1,2) (2,3) (3,1)] --> unstable sort because order changed from the original

Examples of Stable Sort algorithms are : Binary Tree Sort, Bubble Sort, Merge Sort, Insertion Sort, etc
Unstable Sorting Algorithms : Heap Sort, Selection Sort, Quick Sort

TIP - Which Sorting Algorithm is used by Java's Collections.sort(List<E>) in Java 1.7
Collections.sort() uses Iterative merge sort that requires far fewer than n lg(n) comparisons when the input array is partially sorted and is guaranteed to be Stable.

1 http://en.wikipedia.org/wiki/Sorting_algorithm

Question : What is left shift <<, right shift >> and Unsigned right shift operator in Java? How are these useful ?

SOLUTION

All Integer in Java are of signed type (negative numbers are represented in 2's complementary notation), hence Java provides both signed and unsigned bit shift operators to support signed and unsigned shift of bits.

Left Shift Operator << (Signed)

It shifts the underlying bits of an integer to left by the given distance filling the right most bits with zero always.
X = a << b means the same as X = a*2^b
a is given Number and b is the shift amount.
Here is an example of 8 bit representation of number 5. and when we left shift it's bit by 3 then the right most 3 bits are filled by zero.
And the number becomes
$5*2^3 = 40$.

$$\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{1} \xrightarrow{\qquad} \boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}$$

The same thing happens for negative numbers which are represented in 2's complementary notation. for example -5 becomes -40 as follow
11111011 becomes 11011000

Right Shift Operator >> (Signed)

Shifts the bits to left by specified amount maintaining the sign of underlying integer i.e. It fills the left most bits with 0 if the number is positive otherwise with bit 1.
X = a >> b means same as arithmetic operation X = a / (2^b)

Unsigned right shift Operator >>> (does not respect sign of Number)

Unsigned right shift operator >>> is effectively same as >> except that it is unsigned, it fills the left most positions with bit 0 always. (Irrespective the sign of the underlying number)
For example,
So 10000000 >>> 3 becomes 10000 in binary
256 >> 3 becomes 256 / 2^3 = 16.

Notes

- Eight-bit type ***byte*** is promoted to ***int*** in shift-expressions. To mitigate such effects we can use bit masking to get the result as byte for example, (b & 0xFF) >>> 2. Casting can also help achieving the same.

- **Why there is no need of unsigned left shift ?**
  Because there is no need to have that. Sign bit is the right most bit of an integer and shifting bits to right only require the decision of sign. Logical and arithmetic left-shift operations are identical so << signed solves the purpose of unsigned left shift as well.

- **Uses of bitwise operators:** bitwise operators are used for few very efficient mathematical calculations in Big O(1). Bloom Filter, fast mathematical calculations, hashing functions of HashMap are some of applications.

Question : What is 2's complement notation system for Binary Numbers ?

SOLUTION

It is a number format for storing negative numbers in Binary[1]. This system is the most common method of representing signed numbers on computers. An N-bit two's-complement numeral system can represent every integer in the range −(2N−1) to +(2N−1 − 1)

Why 2's Complement ?

The two's-complement system has the advantage that the fundamental arithmetic operations of addition, subtraction, and multiplication are identical to those for unsigned binary numbers (as long as the inputs are represented in the same number of bits and any overflow beyond those bits is discarded from the result). This property makes the system both simpler to implement and capable of easily handling higher precision arithmetic. Also, zero has only a single representation, obviating the subtleties associated with negative zero, which exists in ones'-complement systems.

Calculating 2's complement

Positive numbers are represented as the ordinary binary representation in 2's complementary notation. The most significant bit (leftmost bit) is always 0 for positive number, otherwise number is negative.

To get 2's complement of a negative numbers, the bits are inverted (using bitwise NOT operator) and then value of 1 is added to get the final result.

For example, **Let's convert 5 to -5 using 2's complement notation**

Using 8 bit system, number 5 is represented as
00000101
Now invert all the bits (1's complement)
11111010
Finally add 1 to get the result
11111011

So this is binary representation of -5 in 2's complement notation.

To convert it back to a Positive Number, calculate 2's complement of a negative number
For example, lets convert -5 to 5
11111011 represents -5
Invert all the bits
00000100
then add 1 to get the result
00000101

1 http://en.wikipedia.org/wiki/Two's_complement

Question : How Heap space is divided in Java. How does Garbage Collector cleans up the un-used Objects ? Why shouldn't we use System.gc() command in production code ?

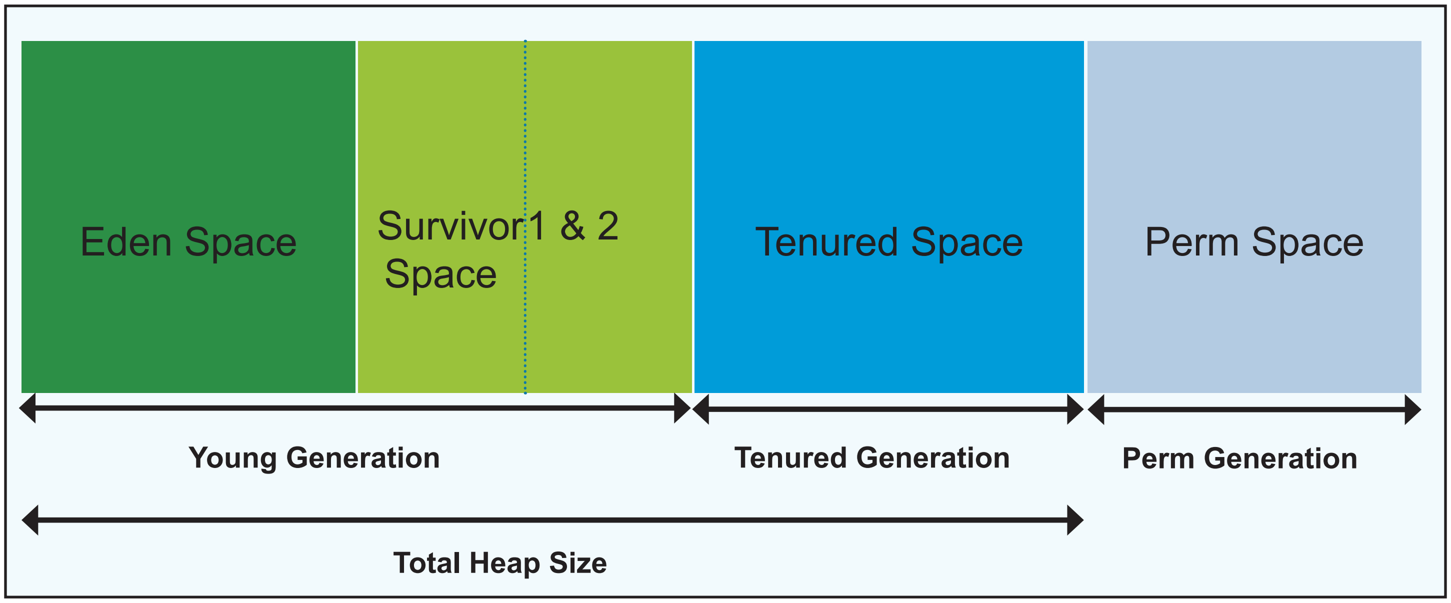
SOLUTION

Memory taken up by the JVM is divided into Stack, Heap and Non Heap memory areas. Stacks are taken up by individual threads for running the method code while heap is used to hold all class instances and arrays created using new operation. Non-heap memory includes a method area shared among all threads and is logically part of the heap but, depending upon the implementation, a Java VM may not GC or compact it.

Java HotSpot VM defines two generations[1]

The Young generation - This further consists of one Eden Space and ***two*** survivor spaces. The VM initially assigns all objects to Eden space, and most objects die there. When VM performs a minor GC, it moves any remaining objects from the Eden space to one of the survivor spaces.

Tenured/Old Generation - VM moves objects that live long enough in the survivor spaces to the "tenured" space in the old generation. When the tenured generation fills up, there is a full GC that is often much slower because it involves all live objects.



Permanent Generation - The permanent generation holds all the reflective data of the virtual machine itself, such as class and method objects.

Memory Spaces

Eden Space (heap): The pool from which memory is initially allocated for most objects.

Survivor Space (heap): The pool containing objects that have survived the garbage collection of the Eden space.

1 http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html
http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html

Tenured Generation (heap): The pool containing objects that have existed for some time in the survivor space.

Permanent Generation (non-heap): The pool containing all the reflective data of the virtual machine itself, such as class and method objects. With Java VMs that use class data sharing, this generation is divided into read-only and read-write areas.

Code Cache (non-heap): The HotSpot Java VM also includes a code cache, containing memory that is used for compilation and storage of native code.

Performance Tuning GC[2]

Set appropriate heap using -Xms and -Xmx VM parameter. Unless you have GC pause problem, you should give as much memory as possible to the virtual machine.

-XX:+DisableExplicitGC
Disable Sysytem.gc() which cause the Full GC to run and thus causing the JVM pauses.

-verbose:gc
-XX:+PrintGC
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
This will print every GC details

-XX:NewRatio
The ratio between the young space and the old is set by this parameter. For example, -XX:NewRatio=2, would make old generation 2 times bigger than the young generation (ratio between the young and tenured generation is 1:2), or we can say that the young generation is 1/3rd the size of total heap size(young + old)

-XX:SurvivorRatio
This command line parameter sets the ratio between each survivor space and eden. For example, -XX:SurvivorRatio=6 will make each survivor space one eighth of the young generation. (there are two survivor space and 6 eden spaces in this case, hence 1/8)

-XX:NewSize=n
Sets the initial size of young generation, it should typically be 1/4th of total heap size. The bigger the young generation, the less frequent the minor collection happens. (though for a bounded heap size, it may cause more frequent major collections)

-XX:PermSize=n -XX:MaxPermSize=n
Sets the permanent generation size (non-heap) which stores Classes, methods and other metadata.

We should carefully design the object pool because they fool the garbage collector by keeping the live reference to the unused objects, thus causing application to demand more memory.

Default Values as of JDK 1.6 on server VM

New Ratio = 2 (old generation is 2 times bigger than young generation)
New Size = 2228K
Max New Size = Not limited
Survivor ratio = 32 (survivor space will be 1/34th of young generation)

2 http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html#generation_sizing.young_gen.survivors

The rules of thumb for server applications are[3]

- First decide the maximum heap size you can afford to give the virtual machine. Then plot your performance metric against young generation sizes to find the best setting.
  - Note that the maximum heap size should always be smaller than the amount of memory installed on the machine, to avoid excessive page faults and thrashing.
- If the total heap size is fixed, increasing the young generation size requires reducing the tenured generation size. Keep the tenured generation large enough to hold all the live data used by the application at any given time, plus some amount of slack space (10-20% or more).
- Subject to the above constraint on the tenured generation:
  - Grant plenty of memory to the young generation.
  - Increase the young generation size as you increase the number of processors, since allocation can be parallelized.

Notes

Question : We have a application which creates millions of temporary large StringBuilder Objects from multiple threads. But none of such object is really required after extracting useful information from them. Somehow we started facing frequent gc pauses. What could be the problem, and how would you approach it ?

Solution

Performance tuning GC may solve this problem to some extent. Let's first understand memory requirements of this application.

This application create lots of short lived objects - thus we would require a large young generation for lowering the frequency of minor garbage collection. If our young generation is small, then the short lived objects will be promoted to Tenured Generation and thus causing frequent major collection. This can be addressed by setting appropriate value for -XX:NewSize parameter at the JVM startup.

We also need to adjust the survivor ratio so that the eden space is large compared to survivor space, large value of Survivor ratio should help solve this problem.

We can also try increasing the Heap size if we have sufficient memory installed on our computer.

3 http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html#generation_sizing.young_gen.survivors

Question : What is a Binary Tree? Why and where is this used in Java ?

SOLUTION

Binary Tree is a tree data structure made up of nodes. Each node has utmost two children.

Why a Binary Tree and not any other linear data structure ?

Binary trees are a very good option (not the best) for storing data where faster search/retrieval is required based on certain criteria. It does so by storing its elements in sorted order offering low time complexity compared to any other linear data structure. Any un-sorted collection can be inserted into Binary Search Tree in O (n log n) time complexity. Though the insertion time is increased per element from O(1) in Random Access array to O(log n) in Binary Search Tree, but we get a major advantage when we want to search/retrieve a particular element from the tree data structure.

Worst-case Search time complexity is logarithmic in a balanced Binary Search Tree.

Balanced Binary Tree

Binary Tree is useful only when the tree is balanced, because only in that case a Binary Tree provides O(log n) search complexity, otherwise a binary tree will behave more like a linear data structure with O(n) time complexity for searching. A tree is called balanced when the height of the tree is logarithmic compared to number of its elements.

Binary Search Tree

Left child of root is less in value than the right child. And the same is true for left and right sub tree in case of Binary Search Tree. BST is build for efficiently sorting & searching. In Order Traversal of a Binary Search Tree results in Ascending Order sorting of its elements.

Binary Tree Implementations used in JDK 1.6

Red-black-tree (TreeMap) and binary heap (PriorityQueue) implementation of Binary Tree is provided by Java Collection Framework, both of which are thoroughly tested and easy to use.

Red-black-tree is a height balanced binary tree where root is colored black and every other element is colored either black or red with the following two rules,
1. If an element is colored red, none of its children can be colored red.
2. The number of black elements is the same in all paths from the root to the element with one child or with no children.

It is useful for maintaining the order of elements in the collection based on the given comparator. It also provide efficient mechanism to find the neighboring elements which are either big or small compared to given number, because those numbers are stored physically closer in the data structure.

Question : How does Recursion works ?
SOLUTION

A general structure of any recursion program is like this :

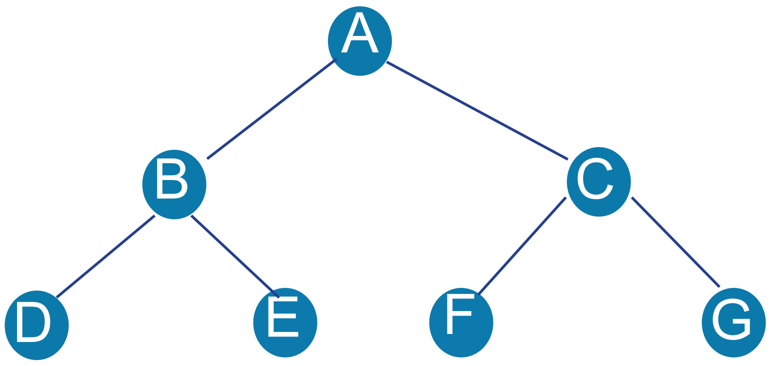
```
if(base condition ...) // return some simple iterative expression
else // recursive case
{
//some work before call
//recursive call
//some work after call
}
```

Recursion is helpful in writing complex algorithms in easy to understand manner. But normally iterative solutions provide better efficiency compared to recursive one because of so much overhead involved in executing recursive steps.
For example, we would use the following code to calculate the **Fibonacci** series using recursion

```
public int fib(int n){
      if(n <= 1)                    //Base Condition
         return 1;
      else {                        //Recursive case
         return fib(n-1) + fib(n-2);
      }
   }
```

Question : How many elements a complete binary tree could hold for a depth of 10?
SOLUTION

A binary tree is said to be complete if it is fully populated, so that each node has two child except the child nodes.



From the figure shown, we can conclude that maximum
Nodes at level 0 = 1
Nodes at level 1 = 2
Nodes at level 2 = 4
Nodes at level n = 2^n

So maximum number of nodes nl_{max} at level l of a binary tree is $\mathbf{nl_{max} = 2^l}$

And the maximum number of nodes in a Tree with L levels
$\mathbf{nL_{max} = 1+2+4+...+2^l = 2^L-1}$
or in other words $\mathbf{L = \log_2(N+1)}$

Hence, we can say
Total Nodes in 1 level tree = 1
Total Nodes in 2 level tree = 3
Total Nodes in 5 level tree = 31

Question : How does a Hash table work ?

SOLUTION

HashMap is a hashing data structure which utilizes object's hashcode to place that object inside map. It provides best case time complexity of O(1) for insertion and retrieval of an object. So it is a best suited data structure where we want to store a key-value pair which later on can retrieved in minimum time.

HashMap is not a thread safe ADT, so we should provide necessary synchronization if used in multi-threaded environment.
HashMap is basically an array of buckets where each bucket uses linked list to hold elements.

Initial Capacity
The default initial capacity of a hashmap is 16 (the number of buckets) and it is always expressed in power of two (2,4,8,16, etc) reaching maximum of 1 << 30 (2^{30})

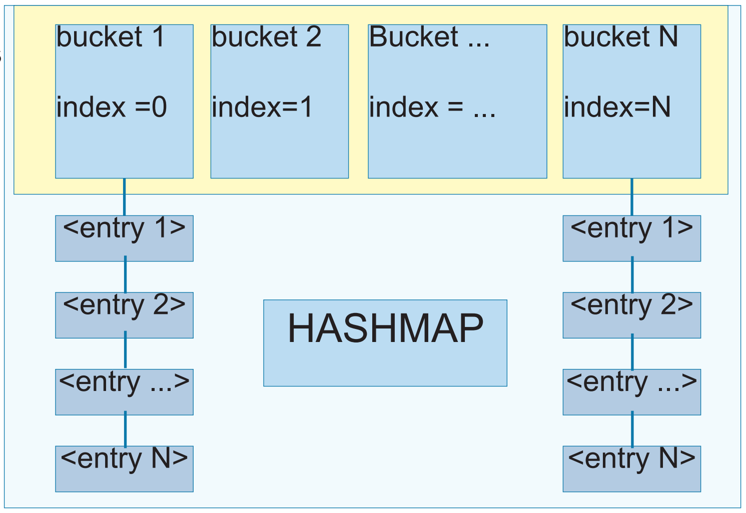
PUT Operation - Big O(1)
When we add a key-value pair to hashmap, it queries key's hashcode. Hashmap uses that code to calculate the bucket index in which to place the key/value. For example, if hashcode is zero then hashmap will place the key value in 0th bucket. Hashmap strips down the key's hashcode to fit into the existing count of buckets using a bitwise hack which is equivalent to the shown below,

bucket index = hashcode % (number of buckets)

The actual method is implemented as

```
    static int indexFor(int hashcode, int length) {
            return hashcode & (length-1);
        }
```

The number of buckets in a hashmap is always power of two, i.e. 2,4,8,16, etc, otherwise the above mentioned bitwise method will not work correctly.
Once the bucket is identified, the key-value pair is added to the List lying at that bucket address.
When multiple objects map to same bucket, we call that phenomenon Collision.



A Typical Hashing Data Structure

GET Operation - Big O(1)
get operation takes a key and then calculates the index of bucket using the method mentioned above. Then that bucket's List is searched for the given key using key's equals() method, and finally the result is returned.

Load factor and Rehashing
Rehashing occurs automatically by the map when the number of keys in the map reaches threshold value.
threshold = capacity* (load factor of 0.75)
In this case a new array is created with more capacity and all the existing contents are copied over to it.

Question : Discuss internal's of a concurrent hashmap.

SOLUTION

In Java 1.7, A ConcurrentHashMap is a hashmap supporting full concurrency of retrieval via volatile reads of segments and tables without locking, and adjustable expected concurrency for updates. All the operations in this class are thread-safe, although the retrieval operations does not depend on locking mechanism (non-blocking). And there is not any support for locking the entire table, in a way that prevents all access. The allowed concurrency among update operations is guided by the optional concurrencyLevel constructor argument (default is16), which is used as a hint for internal sizing.

ConcurrentHashMap is similar in implementation to that of HashMap, with resizable array of hash buckets, each consisting of List of HashEntry elements. Instead of a single collection lock, ConcurrentHashMap uses a fixed pool of locks that form a partition over the collection of buckets.

Here is the code snippet showing HashEntry class

```
static final class HashEntry<K,V> {
        final int hash;
        final K key;
        volatile V value;
        volatile HashEntry<K,V> next;
...
```

HashEntry class takes advantage of final and volatile variables to reflect the changes to other threads without acquiring the expensive lock for read operations.
The table inside CHM is divided among Segments (which extends ReentrantLock), each of which itself is a concurrently readable hash table. Each segment requires uses single lock to consistently update its elements flushing all the changes to main memory.

put() method holds the bucket lock for the duration of its execution and doesn't necessarily block other threads from calling get() operations on the map. It firstly searches the appropriate hash chain for the given key and if found, then it simply updates the volatile value field. Otherwise it creates a new HashEntry object and inserts it at the head of the list.

Iterator returned by the ConcurrentHashMap is fail-safe but weakly consistent. keySet().iterator() returns the iterator for the set of hash keys backed by the original map. The iterator is a "weakly consistent" iterator that will never throw ConcurrentModificationException, and guarantees to traverse elements as they existed upon construction of the iterator, and may (but is not guaranteed to) reflect any modifications subsequent to construction.

Re sizing happens dynamically inside the map whenever required in order to maintain an upper bound on hash collision. Increase in number of buckets leads to rehashing the existing values. This is achieved by recursively acquiring lock over each bucket and then rehashing the elements from each bucket to new larger hash table.

Reference

http://www.ibm.com/developerworks/java/library/j-jtp08223/

Question : What are database transaction Isolation levels ?

SOLUTION

Let's first get familiar with the most common problems occurring in concurrent database applications, for example

Dirty Read
Occurs when uncommitted results of one transaction are made visible to another transaction.

Unrepeatable Reads
Occurs when the subsequent reads of same data by a transaction results in seeing different values.

Phantom Reads
One transaction performs a query returning multiple rows, and later executing the same query again sees some additional rows that were not present the first time.

We also call above three as Isolation Hazards, and the Transaction Isolation levels are related to these three problems.

Isolation Level	Dirty read	Unrepeatable read	Phantom read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No

For most of databases, the default Transaction Isolation Level is Read Committed.
(Read Committed does not see any inconsistent state of other transaction, with a fair amount of concurrency)

Please be noted that the above four isolation levels are in decreasing order of their concurrency. So for scalability reasons, Serializable is rarely a good choice of design, as it offers only a single thread to work at a given time.

General practice to choose Isolation level
Choose the lowest isolation level that can keep our data safe.

References
http://www.ibm.com/developerworks/java/library/j-jtp0514/index.html

Question : What is difference between Primary key and Unique Key ?

SOLUTION

Differences

1. The main purpose of a Primary key is to uniquely identifies a given record in a Table, while the same is not true for Unique Key constraint.
2. A table can have one and only one Primary Key, while there could be multiple unique keys inside a single table.
3. Primary Key can not have Null values while Unique key column can contain a Null value
4. Primary key creates the Clustered index, but unique key creates the Non clustered index.

Question : What are clustered and non-clustered indexes in database ?

SOLUTION

Clustered Index

Clustered index physically rearrange the data that users inserts in your tables. It is nothing but a dictionary type data where actual data remains.

Non-Clustered Index

It Non-Clustered Index contains pointers to the data that is stored in the data page. It is a kind of index backside of the book where you see only the reference of a kind of data.

Question : How does Session handling works in Servlet environment ?

SOLUTION

There are multiple ways to handle session by a server framework. For example following methods can be used,

1. Storing Cookies on the client side
2. URL Rewriting
3. Hidden form fields

Servlets use cookies as the default mechanism for session tracking, but in case cookies are disabled on the client, Server can use URL re-writing for achieving the same.

When server calls request.getSession(true), then server generates and sends JSESSIONID back to the client for all future session references. JSESSIONID will then be stored by the client and sent back to the server using any of the above mentioned mechanisms.

To ensure that your Servlets support servers that use URL rewriting to track sessions, you must pass all the URL's used in your servlet through the

HttpServletResponse.encodeURL() method like :

```
out.println("<form actionb ='"+res.encodeURL("/example/htmlpage")+"'>");
```

this will append the sessionID to the form's action.

Question : What is difference between HTTP's Redirect and Forward ?

SOLUTION

Forward

1. Control is forwarded to the resource available within the server from where the call is made, the transfer of control is made internally by the container, where client is completely unaware that a forward is happening.
2. When forward is done, the original request and response objects are transferred along with the additional parameters if needed.
3. Forward can't transfer control to some other domain.
4. Original URL at the client side remains intact, hence refreshing the page will cause the whole step to repeat again.
5. Session object is not lost in forward or redirect.

Redirect

1. A redirect is a two step process where web application instructs the browser client to fetch the fetch the second URL which differs from the original.
2. Server sends Http Status Code of 301 to the client, and then client follows the instructions.
3. If someone reloads the page on browser, then original request will not be repeated. Just the second url will be fetched again.
4. Redirect is marginally slower than forward, since it requires two requests.
5. Objects placed in request scope are not available to second request.
6. There are several ways to perform a redirect for example,

```
Http/1.1 301 moved permanently
Location : http://www.foobar.org/

<meta http-equiv="refresh" Content="0; URL=http://www.example.org/"/>

<script type="text/javascript"> location.href ="http://www.foobar.org" </script>
```

Redirect Should be used when

1. If you need to transfer of control to a different domain.
2. To achieve separation of tasks.

For example, database update and data display can be separated by redirect. In this case if user presses F5 button the browser then only display part will execute again and not the database update part.

Do the PaymentProcess and then redirect to the display payment info, if the client refreshes the browser, only the displayInfo will be done again and not the payment process.

Use Forward when

For database SELECT operations, use a forward.

Question : Discuss Visitor, Template, Decorator, Strategy, Observer and Facade Design Patterns ?

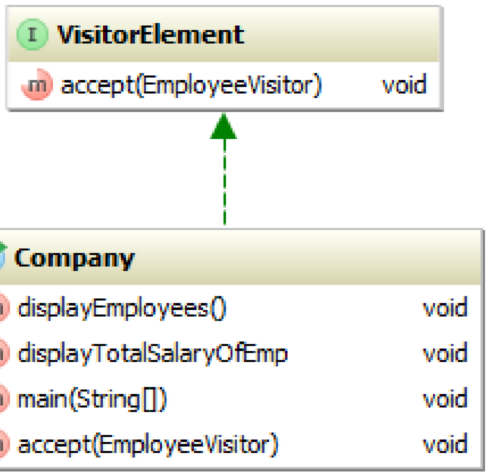
SOLUTION

Visitor Design Pattern

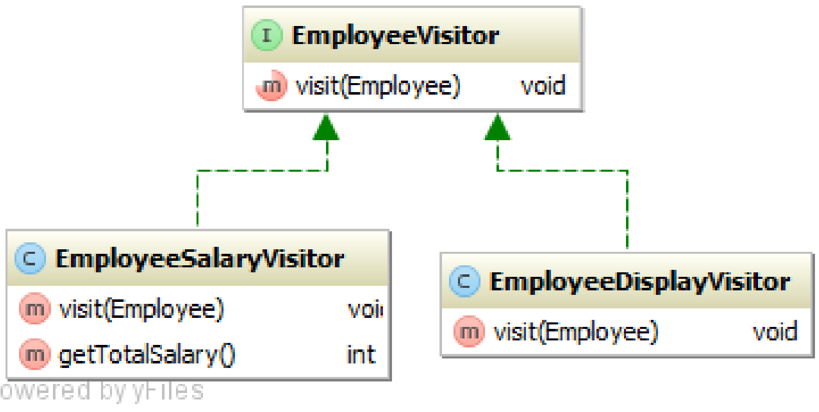
Visitor design pattern is a way of separating an algorithm from an object structure on which it operates. This results in ability to add new operations to existing object structures without modifying those structures.

For example,
Suppose, A company wants to display the first name of all its employees. In order to achieve a separation between the data model and the algorithm to display employee, Visitor pattern can be utilized. Moreover if a new requirement comes in for calculating the total salary of an employee, then it can be easily added without making any changes to the model.

Similarly
Calculating taxes in different regions on sets of invoices would require many different variations of calculation logic. Implementing a visitor allows the logic to be de-coupled from the invoices and line items. This allows the hierarchy of items to be visited by calculation code that can then apply the proper rates for the region. Changing regions is as simple as substituting a different visitor.


VisitorElement
accept(EmployeeVisitor) void
Company
displayEmployees() void
displayTotalSalaryOfEmp void
main(String[]) void
accept(EmployeeVisitor) void

```
public interface EmployeeVisitor {
    abstract void visit(Employee emp);
}

class EmployeeDisplayVisitor implements
EmployeeVisitor{

    @Override
    public void visit(Employee emp) {
        System.out.println(emp.getName());
    }
}

public interface VisitorElement {
    void accept(EmployeeVisitor visitor);
}
```



```
public class Company implements
VisitorElement {
private final List<Employee> employees = new ArrayList<>();
public void displayEmployeeNames(){
        EmployeeDisplayVisitor visitor = new EmployeeDisplayVisitor();
        accept(visitor);
}
```

Continued on 29

```
@Override
public void accept(EmployeeVisitor visitor){
        for (Employee employee : employees) {
            visitor.visit(employee);
        }
}
```

Template Design Pattern

The template design pattern defines program skeleton of an algorithm in a method called a template method, which defers some steps to subclasses. It lets one redefine certain steps of an algorithm without changing the algorithm's structure.

For example,
A online computer buying shop (www.dell.com) provides with a template specifications of computer, like Monitor, HDD, RAM, CPU, Disk Drive, etc. An individual buyer can customize the HDD capacity, Monitor size, RAM size, CPU clock speed keeping the overall structure same.

Decorator Design Pattern

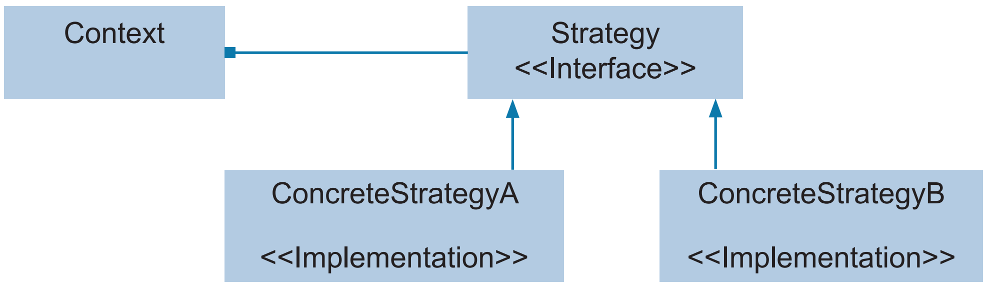
A design pattern that allows behavior to be added to an existing object dynamically.

Observer Design Pattern

A pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notify them automatically of any state changes, usually by calling one of their methods.

Strategy Design Pattern

Strategy is a design pattern whereby an algorithm's behavior can be selected at runtime, and making them interchangeable. Strategy lets the algorithm vary independently from clients that use it.



Class Diagram For Strategy

Context uses strategy to perform a calculation, whose behavior can be changed at runtime by substituting ConcreteStrategyA with ConcreteStrategyB.

Facade Design Pattern

Facade provides a single interface to a set of interfaces in the system, like in case of DAO Facade where we hide the internal details of an ORM framework and provide the end programmer with a simple interface to find/ delete and update an Object into the database.
It should be used when a simple interface is needed to provide access to a very complex underlying system.

Another good example of facade design pattern could be : exposing a set of functionalities using web services (SOA architecture). Client does not need to worry about the complex dependencies of the underlying system after building such API.

Question : What is a strong, soft, weak and Phantom reference in Java ? Where are these used ?

SOLUTION

Interviewer's Intent - wants to know your understanding for GC, automatic memory allocation and de-allocation.3

SoftReference, WeakReference & PhantomReference are are reference-object classes, which supports limited degree of interaction with the GC. A programmer may use these classes to maintain a reference to some other object (referent) in such a way that the object may still be reclaimed by GC.

Reference Queues

Reference queue is used to track the objects claimed by GC. We can use the reference objects to check whether the objects referred by these are still active or are claimed by GC.

SoftReference

If the strongest reference to an object is a soft reference then GC will not reclaim the object until the JVM is falling short of memory, though it must be reclaimed before throwing an Out Of Memory Error. So the object will stay longer than a weakly referenced object. It is mostly used for writing memory sensitive caches.

WeakReference

Is similar to soft reference with the only difference that it will be GC'ed in the next GC cycle if the strongest reference to the object is a weak reference. When a weak reference has been created with an associated reference queue and the referent becomes a candidate for GC, the reference object (not the referent) is enqueued on the reference queue after the reference is cleared. The application can then retrieve the reference from the reference queue and learn that the referent has been collected so it can perform associated cleanup activities, such as expunging the entries for objects that have fallen out of a weak collection.

WeakHashMap

It is a HashMap which store its keys (not the values!) using WeakReferences. An entry in this map will be automatically removed when there are not other non-weak references to keys.
It can be used to store associative objects like transient object & its metadata, as soon as the object is claimed by the GC, the associated metadata will also be removed by the map. Other application could be in a servlet environment where as soon as the session expire's, clear all the session data/attributes.

PhantomReference

PhantomReference are garbage collected when the strongest reference to an object is a phantom. When an object is phantomly reachable, the object is already finalized but not yet reclaimed, so the GC enqueues it in a reference queue for post finalization processing. A Phantom Reference is not automatically cleared when it is enqueued., so we must remember to call its clear() method or to allow phantom reference object itself to be garbage collected. get() method always return null so as not to allow resurrect the referent object.
Phantom references are safe way to know an object has been removed from memory and could be thought of as a substitute for finalize() method.

Continued on 31

Automatically-cleared references

Soft and weak references are automatically cleared by the collector before being added to the queues with which they are registered, if any. Therefore soft and weak references need not be registered with a queue in order to be useful, while phantom references do. An object that is reachable via phantom references will remain so until all such references are cleared or themselves become unreachable.

Reachability levels from strongest to weakest : strong, soft, weak, phantom. Java 6 docs states that -

- An object is strongly reachable if it can be reached by some thread without traversing any reference objects. A newly-created object is strongly reachable by the thread that created it.
- An object is softly reachable if it is not strongly reachable but can be reached by traversing a soft reference.
- An object is weakly reachable if it is neither strongly nor softly reachable but can be reached by traversing a weak reference. When the weak references to a weakly-reachable object are cleared, the object becomes eligible for finalization.
- An object is phantom reachable if it is neither strongly, softly, nor weakly reachable, it has been finalized, and some phantom reference refers to it.
- Finally, an object is unreachable, and therefore eligible for reclamation, when it is not reachable in any of the above ways.

Notes

WeakHashMap is not a solution for implementing cache, SoftReference's could be better utilized for implementing cache.

Applications of a WeakHashMap

WeakHashMap stores its keys using WeakReference, and can be used to map transient objects with their metadata. Let's suppose we have a socket application which creates sockets on client's request and socket lives there for sometime. Now if we want to associate some metadata with this socket such as identity of the user, then WeakHashMap is a ideal container for storing such associative information. Since we are not managing the lifecycle of the socket in this case, WeakHashMap will automatically remove all the metadata as soon as the socket dies.

Applications of SoftReference

Soft references can be used to build memory sensitive cache which automatically collects items as soon as the cache is under high memory load, which otherwise has to be achieved by the programmer.

References

http://www.ibm.com/developerworks/java/library/j-jtp11225/index.html
http://docs.oracle.com/javase/6/docs/api/java/lang/ref/package-summary.html
http://www.ibm.com/developerworks/java/library/j-jtp01274/index.html
http://java.dzone.com/articles/finalization-and-phantom
http://neverfear.org/blog/view/150/Strong_Soft_Weak_and_Phantom_References_Java
http://tech-tauk.blogspot.in/2010/05/soft-reference-weak-refernce-phantom.html

Question : Tell something about TreeSet ?

SOLUTION

Interviewer's Intent - Algorithm & Data Structure, sorting, red black tree

TreeSet is a navigable set implementation based on TreeMap. All the elements are ordered using their Natural ordering or by comparator provided at TreeSet construction time.

NavigableSet provides us with methods like first(), last(), floor(), ceiling(), headSet(), tailSet() which can be used to search the neighboring elements based on element's ordering.

TreeMap is Red-Black Binary Search Tree which guarantees logarithmic time for insertion, removal and searching of an element. All the elements in this collection are stored in sorted order and the tree is height balanced using Red black algorithm. If two elements are nearby in order, then TreeSet places them closely in the data structure.

TIP *TreeSet provides guaranteed Big O (log n) time complexity for add(), remove() and contains() method.*

Uses

It is a best collection if we need to search the nearby elements of a given item based on their ordering.

Notes

- Note that this implementation is not synchronized. If multiple threads access a tree set concurrently, and at least one of the threads modifies the set, it **must** be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the set.
  If no such object exists, the set should be "wrapped" using the Collections.synchronizedSortedSet method. This is best done at creation time, to prevent accidental unsynchronized access to the set:

```
SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));
```

- If we are looking for high throughput in a multi-threaded application then we can prefer **ConcurrentSkipListSet** which is scalable concurrent implementation of NavigableSet.
- Iterator returned by this class are fail-fast.
- TreeSet does not allow duplicates, it just replaces the old entry with the new one if both are equal (using compareTo method)
- TreeSet does not preserve the insertion order of its elements.

Question : What are Inheritance strategies in JPA ?

SOLUTION

JPA defines three inheritance strategies namely, SINGLE_TABLE, TABLE_PER_CLASS and JOINED.

Single table inheritance is default, and table per class is optional so all JPA vendors may not support it.
JPA also defines mapped super class concept defined through the @MappedSuperClass annotation. A Mapped Super Class is not a persistent class, but allows a common persistable mapping to be defined for its subclasses.

Single Table Inheritance
In this inheritance, a single table is used to store all the instances of the entire inheritance hierarchy. The Table will have a column for every attribute of every class in the hierarchy. Discriminator columns identifies which class a particular row belongs.

Table Per Class Inheritance
A table is defined for each **concrete** class in the inheritance hierarchy to store all the attribute of that class and all its super classes.

Joined Table
This inheritance replicates the object model into data model. A table is created for each class in the hierarchy to store only the local attributes of that class.

Notes

Question - We want to extract common behavior in a super class in JPA entities but we do not want to have table for that super class. How would you achieve this ?
Answer - If we create a normal class as the super class, then as per JPA specifications, the fields for that class are not persisted in the database tables. We need to create a super class extracting the common fields and then annotate that class with @MappedSuperClass in order to persist the fields of that super class in subclass tables. A mapped super class has no separate table defined for it.

References

http://en.wikibooks.org/wiki/Java_Persistence/Inheritance

Chapter 2

Core Java

Question : What is difference between method overloading and overriding, method and variable hiding ?

SOLUTION

Rules for Overriding[1]

- The argument list must exactly match that of the overridden method. If they don't then overloading will be the result instead of overriding
- The return type must be of covariant type (same or sub-class) -- we can narrow down return type
- Overriding method can only throw the same or narrowed checked exception --> narrow down exception
- Access level can be less restrictive --> broader visibility of the method
- Free to throw any kind of Runtime exception
- Private and final methods are not inherited and hence can't be overridden
- Static methods can't be inherited and can't be overridden
- No inheritance no overriding
- The type of the actual object on the heap decides which method is selected at runtime

Rules for Overloading

- Method name must be the same
- Argument List must change in overloaded method
- Overloaded method can change the return type
- Access modifier of overloaded method can change
- New or Broader exceptions can be thrown by overloaded method
- Inherited method from the super class (non private) can be overloaded in the subclass
- Reference type determines which overloaded version is selected based on argument types at compile time

Method Signature

Method name, plus the number and type of its parameters constitute the method signature

Best Practice to avoid any confusion for a overridden method

When overriding a method, you might want to use the @Override annotation that instructs the compiler that you intend to override a method in the superclass. If, for some reason, the compiler detects that the method does not exist in one of the superclasses, it will generate an error.[2]

Hiding variables

Overriding works for Instance methods, In case of Class methods If a subclass defines a class method with the same signature as a class method in super class, the method in subclass hides the one is super class.

Similarly variable names are never overridden by the sub class, but they can be hide. If the super class contains a variable named x and subclass also contains x (irrespective of the type of variable) then the subclass variable hides the super class variable. Remember that all non-private super class variable can always be referenced by the subclass using super.variable.

Notes

In a subclass, you can overload the methods inherited from the superclass. Such overloaded methods neither hide nor override the superclass methods—they are new methods, unique to the subclass.

Question : If a method throws NullPointerException in super class, can we override it with a method which throws RuntimeException? The answer is Yes - there is no policy for RuntimeException in method overriding.

1 SCJP Sun® Certified Programmer for Java™ 6 Study Guide Exam (310-065) Page 106
2 http://docs.oracle.com/javase/tutorial/java/IandI/override.html

Question : Why wait is always used inside while loop as shown in the below snippet ? Discuss all the probable reasons.

```
public synchronized void put(T element) throws InterruptedException {
        while(queue.size() == capacity) {
            wait();
        }
        queue.add(element);
        notify();
}
```

SOLUTION

There are two main reasons that force use to use wait() method inside a while loop[1].

Spurious WakeUp[2]

In certain rare scenarios, a thread can wakeup spuriously even when no other thread signaled the condition. To gracefully handle those scenarios, we must recheck for the required condition before proceeding to execute the rest of the condition dependent code.

Multiple Threads Waiting for the Single Signals

If a thread calls notifyAll() upon meeting certain condition, then all the consumer threads will wakeup, even though only one thread will be expected to proceed in that scenario.
Let's analyze problem with the below mentioned Queue's take() method. Suppose there are 2 consumer threads awaiting for any new item on this shared queue. As soon as the Producer thread will put a single element into this queue, it will invoke notifyAll() and thus resuming all the 2 Consumer threads. Both the Consumer threads will come out of waiting state and will fight to acquire lock executing the rest of the code (line 5-7) one at a time. This will cause the second thread to throw exception because there was just one element in the queue.

```
1.   public synchronized T take() throws InterruptedException {
2.           if(queue.isEmpty()) {                          <=== Problematic If condition
3.               wait();
4.           }
5.           T item = queue.remove();
6.           notify();
7.           return item;
8.       }
```

Replacing **if** condition with a **while** loop can solve this problem without much effort. While loop will force each resuming thread to *test the condition on wakeup*, and putting the thread to waiting state again if required condition is not met.
So always remember to use wait() method from inside the while loop testing the condition that caused the thread to awaken, as shown below.

```
synchronized (obj) {
    while (<condition does not hold>)
        obj.wait(timeout);
    ... // Perform action appropriate to condition
 }
```

1 see Section 3.2.3 in Doug Lea's "Concurrent Programming in Java (Second Edition)" , or Item 50 in Joshua Bloch's "Effective Java Programming Language Guide" (Addison-Wesley, 2001)
2 http://en.wikipedia.org/wiki/Spurious_wakeup

Question : What is Order of calling constructors in case of Inheritance?

SOLUTION

Constructors are called from the top to down hierarchy. For example as shown in the below code snippet, A is super class of B. Creating a instance of new B() will invoke B's constructor which will in-turn call super() and this constructor of A will get invoked. Call to super() is inserted by the compiler on our behalf and is not visible to us. So instance of A will be created first, and if the constructor of A contains any method which is overridden in sub class, the subclass version will be invoked except for the static method calls.

Compiler introduces the call to super() only if the default no-arg constructor is present in the super class, otherwise its responsibility of the programmer to introduce such call with proper constructor arguments.

Java Source

```
class A {
    A() {
        greeting();
        prints();
    }
    void greeting() {
        System.out.println("instance method from A");
    }
    static void prints() {
        System.out.println("Static method from A");
    }
}

class B extends A {
    B() {
        greeting();
        prints();
    }
    void greeting() {
        System.out.println("instance method from B");
    }
    static void prints() {
        System.out.println("Static method from B");
    }
}

public static void main(String[] args) {
        new B();
}
```

Program Output

```
instance method from B
Static method from A
instance method from B
Static method from B
```

Question : Iterator interface provides remove() method but no add() method. What could be the reason ?

SOLUTION

Iterator interface contains three methods namely remove(), hasNext() and next().

It intentionally does not provide any add() method because it should not !
Iterator does not know much about the underlying collection. Underlying collection could be of any type (Set, ArrayList, LinkedList, etc) and might be offering the guaranteed ordering of its elements based on some algorithm. For example TreeSet maintains the order of its element using Red Black Tree datastructure. Now if iterator tries to add an element at a given location, then it might corrupt the state of the underlying data structure. And that is not the case while removing elements.
Thus Iterator does not provide any add() method.

List Iterator does provide the add() method because it know the location where it needs to add the newly created element as List preserves the order of its elements.

Question : What does Collections.unmodifiableCollection() do ? Is it a good idea to use it safely in multi-threading scenario without synchronization, Is it immutable ?

SOLUTION

Collections.unmodifiableCollection() returns a unmodifiable dynamic view of underlying data structure. Any attempt direct or via iterator to modify this view throws UnsupportedOperationException, but any changes made in the underlying data structure will be reflected in the view.
This method is no substitute for the other thread safety techniques because iterating over a collection using this view may throw ConcurrentModificationException if original collection is structurally modified during the iteration.

So **external synchronization is must** if we are going to modify the underlying collection.
For example, the following code will throw ConcurrentModificationException in the for loop.

```
public class UnModifiableCollection {
    private List<String> names = new ArrayList<>();
    public void testConcurrency(){
        names.add("1");
        names.add("2");
        names.add("3");
        names.add("4");
        Collection<String> dynamicView = Collections.unmodifiableCollection(names);
        for (String s : dynamicView) {              <=== will throw ConcurrentModification in 2nd iteration
            System.out.println("s = " + s);
            names.remove(0);                        <=== The culprit line modifying the underlying collection
        }
    }
    public static void main(String[] args) {
        UnModifiableCollection test = new UnModifiableCollection();
        test.testConcurrency();
    }
}
```

Question : If hashcode() method of an object always returns 0 then what will be the impact ?

SOLUTION

Hashcode is used to fairly distribute elements inside a map into individual buckets. If the hashcode returned is zero for each element then the distribution will no more be fair and all the elements will end up into a single bucket. Each bucket in a HashMap contains list of HashEntry objects, so in a way HashMap will act as a map with single bucket holding all of its elements in a list. That will drastically reduce HashMap's performance to that of a LinkedList for get and put operations.
So time complexity of get and put method will become : Big O(n) instead of Big O(1)
Although, functionally it will still behave correctly.

Question : If we don't override hashcode() while using a object in hashing collection, what will be the impact ?

SOLUTION

Then the Object's default hashcode() method will be used to calculate the hashcode, which in turn will return the memory address of the object in hexadecimal format. So in a way the hashmap will behave like a identity hashmap which will consider two elements equal if and only if two objects are same as per their memory address (and not logically). For example two String Objects with same contents might be treated different by this hashmap if they are different on heap.

Question : How would you detect a DeadLock in a running program ?

SOLUTION

Deadlock occurs in software program due to circular dependencies on shared resources by multiple threads. This causes the partial or complete hanging of the software program and the program itself can not recover from the dead lock situation. But still its possible using multiple ways to detect the Dead Lock in a running JVM.

1. Using Jconsole - JDK installation ships with jconsole tool which can connect to a running java process using JMX protocol. Jconsole can tell us whether there is a dead lock in the program or not.
2. Using JMX Management package as shown below

```
JDK 1.5 Has some API from java.lang.management package

ThreadMXBean tmx = ManagementFactory.getThreadMXBean();
long[] ids = tmx.findDeadlockedThreads();
```

This will list down the thread ids for the troubleshooting purpose.

Question : Which data type would you choose for storing currency values like Trading Price ? What's your opinion about Float, Double and BigDecimal ?

SOLUTION

Float & Double are Bad for financial world, never use them for monetary calculations.

There are two main reasons supporting this statement -

- All floating point values that can represent a currency amount (in dollars and cents) can not be stored exactly as it is in the memory. So if we want to store 0.1 dollar (10 cents), float/double can not store it as it is. Let's try to understand this fact by taking this simple example

```
 public class DoubleForCurrency {
  public static void main(String[] args) {
      double total = 0.2;
      for(int i=0;i<100;i++){
          total+=0.2;
      }
      System.out.println("total = " + total);
  }
}
```

```
OUTPUT : total = 20.19999999999996
The output should have been 20.20 (20 dollars and 20 cents), but floating point calculation made it 20.1999999999..
```

- There is not much flexibility provided by Math.round() method for rounding the given calculation result compared to functionality offered by MathContext. RoundingMode provides options such as ROUND_UP, ROUND_DOWN, ROUND_CEILING, ROUND_FLOOR, ROUND_UNNECESSARY, etc

BigDecimal For the Rescue
BigDecimal represents a signed decimal number of arbitrary precision with an associated scale. BigDecimal provides full control over the precision and rounding of the number value. Virtually its possible to calculate value of pi to 2 billion decimal places using BigDecimal.

That's the reason we should always prefer BigDecimal or BigInteger for financial calculations.

Notes

Primitive type - int and long are also useful for monetary calculations if decimal precision is not required

We should really avoid using BigDecimal(double value) constructor instead prefer BigDecimal(String) because BigDecimal (0.1) results in 0.100000...5..3 being stored in BigDecimal instance. In contrast BigDecimal ("0.1") stores exactly 0.1

Question : What is Precision and Scale ?
Precision is the total number of digits (or significant digits) of a real number
Scale specifies number of digits after decimal place

For example, 12.345 has precision of 5 and scale of 3

How to format BigDecimal Value without getting exponentiation in the result & Strip the trailing zeros?

We might get exponentiations in the calculation result if we do not follow some best practices while using Bigdecimal. Below is the code snippet which shows a good usage example of handling the calculation result using Bigdecimal.

```
public class BigDecimalForCurrency {
    public static void main(String[] args) {
        int scale = 4;
        double value = 0.11111;
        BigDecimal tempBig = new BigDecimal(Double.toString(value));
        tempBig = tempBig.setScale(scale, BigDecimal.ROUND_HALF_EVEN);
        String strValue = tempBig.stripTrailingZeros().toPlainString();
        System.out.println("tempBig = " + strValue);
    }
}
```

How would you print a given currency value for Indian Locale (INR Currency)?

NumberFormat class is designed specifically for this purpose. Currency symbol & Rounding Mode is automatically set based on the locale using NumberFormat. Lets see this example

```
public static String formatRupees(double value){
         NumberFormat format = NumberFormat.getCurrencyInstance(new Locale("en", "in"));
         format. setMinimumFractionDigits(2);
         format. setMaximumFractionDigits(5);
         return format.format(value);
    }

public static void main(String[] args) {
        BigDecimal tempBig = new BigDecimal(22.1214);
        System.out.println("tempBig = " + formatRupees(tempBig.doubleValue()));
    }
```

Output

```
tempBig = Rs.22.12
```

Thus everything is taken care by NumberFormat, nothing lese to worry about.

Some precautions

- BigDecimal(String) constructor should always be preferred over BigDecimal(Double)
- Convert Double value to string using Double.toString(double) method
- Rounding mode should be provided while setting the scale
- StripTrailingZeros chops off all the trailing zeros
- toString() may use scientific notation but, toPlainString() will never return exponentiation in its result

Reference
https://blogs.oracle.com/CoreJavaTechTips/entry/the_need_for_bigdecimal

Question : How great is the Idea of synchronizing the getter methods of a shared mutable state ? What if we don't ?

SOLUTION

From **Effective Java 2nd Edition - Item 66**
When multiple threads share mutable data, each thread that reads or writes the data must perform synchronization. In fact, synchronization has no effect unless both read and write operations are synchronized.

Synchronization serves two major purposes in a multi-threaded scenario, one is atomicity of the operation and second is the memory visibility of the changes made by one thread to all other threads (Brian Goetz article on read and write barriers)[1]. In case of getters the changes made to the shared variable will get reflected to the new thread if the code block is synchronized, otherwise dirty reads may happen and the thread may see the stale state of the shared object.

So all the methods returning the mutable protected state of the shared object must be synchronized unless the field returned is immutable, final or volatile.

Let's take example of a simple Counter Class.

```
public class Counter {
        private int c = 0;
        public synchronized void increment() {
                c++;
        }
        public synchronized int getValue() {  <=== Must be synchronized to see the guaranteed correct value
                return c;
        }
}
```

That's the reason that get() method of vector class is synchronized & must be synchronized.

Question : Can the keys in Hashing data structure be made Mutable ?

SOLUTION

Interviewer's intent is to know how good you know about Hashing Data Structure

The answer is **NO**. If we make the keys mutable then the hashcode() of the key will no more be consistent over time which will cause lookup failure for that object from the data structure. Let's analyze this example.

```
public void testMutableKey(){
        Map<MutableKey, Object> testMap = new HashMap<>();
        MutableKey mutableKey = new MutableKey();
        mutableKey.setName("TestName");
        testMap.put(mutableKey, new Object());
        Object o = testMap.get(mutableKey);
        System.out.println("before changing key = " + o);
        mutableKey.setName("abc");                          <==== Problematic Instruction
        o = testMap.get(mutableKey);
        System.out.println("after changing key = " + o);
    }
```

1 http://www.ibm.com/developerworks/library/j-jtp08223/

```
        public static void main(String[] args) {
            MutableHashKey test = new MutableHashKey();
            test.testMutableKey();
        }
```

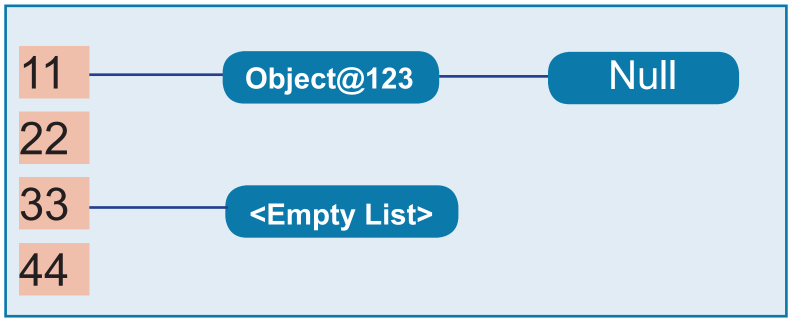
Program Output :

```
before changing key = java.lang.Object@489bb457
after changing key = null
```

From the above example we see that as soon as we change the key, we are not able to get the associated object from the Map.

Let's see what's happening inside

When we put the mutableKey to HashMap then hashcode() is calculated for the key, suppose it comes out to be 11. So the Object123 is successfully inserted into the HashMap at bucket Location 11.



HashMap Internal Structure

Then we modify the key and try to get the object. HashMap's get() method again calculates the hashcode of the Key, since the Key is changed in between, so suppose hashcode() comes out to be 33 this time. Now the get() method goes to the bucket at address 33 and tries to retrieve the object, but it find nothing over there and returns the null.

Never make changes to the hashmap's key, otherwise the associated object can not be fetched using get() method. Though it will be accessible using other methods which iterate over the entire collection.

Question : Is it safe to iterate over collection returned by Collections.synchronizedCollection() method, or should we synchronize the Iterating code ?

SOLUTION

We should synchronize the code block doing any kind of iteration as stated by the Java Docs

```
public static <T> Collection<T> synchronizedCollection(Collection<T> c)
```

Returns a synchronized (thread-safe) collection backed by the specified collection. In order to guarantee serial access, it is critical that all access to the backing collection is accomplished through the returned collection. It is imperative that the user manually synchronize on the returned collection when iterating over it:

```
Collection c = Collections.synchronizedCollection(myCollection);
  ...
synchronized (c) {
    Iterator i = c.iterator(); // Must be in the synchronized block
    while (i.hasNext())
       foo(i.next());
}
```

Failure to follow this advice may result in non-deterministic behavior.

Question : What are different type of Inner classes in Java ? How to choose a type with example ?

SOLUTION

An inner class is a class defined within another class, or even within an expression. They are mostly used to simply our code by putting closely related classes together in one source file, instead creating class burst.

Event handlers are best examples of Inner Classes.

Types of Inner Class -

- Regular Inner Classes (classes defined within the curly braces of a regular class)
- Static Inner Classes (that can be accessed without having an instance of outer class)
- Method-local Inner Classes (Inner class defined within a method body)
- Anonymous Inner Classes (Without any class name)

Question : Why do we need a static inner class rather than creating a top level class in Java ?

SOLUTION

A static Class interacts with the instance members of its outer class and other classes just like any top level class. In fact, a static nested class is behaviorally a top level class that has been nested in another top level class for packaging convenience.

If we take a example of LinkedList.Entry class, there is no need of it being a top level class as it is only used by LinkedList. Otherwise it will cause class burst inside a package, moreover there are other static inner classes by the same name as well like Map.Entry

And since these does need access to LinkedList/Map's internal so it makes sense to make them static inner classes.

Why to use it ?

1. It is a way of logically grouping the classes that are only used in one place. If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together.
2. It increases encapsulation.
3. Nested classes can lead to more readable and maintainable code. Nesting small classes within top-level classes places the code closer to where it is used.

Examples

Iterator in most of the collection types are implemented as a inner class and Entry is implemented as static inner class.

Question : Is it possible to write a method in Java which swaps two int or Integer ?

Solution

The answer is **No**.

For knowing the exact answer you must be knowing how Parameter Passing works in Java.

Incase of primitive int
Parameters to the method are passed by value in Java. In case of primitive data types, a copy of the value is passed to the method, so any changes in the method will not reflect in the calling code.

Incase of Integer Wrapper Class
For objects, the reference to the Object are copied by value to the calling method. If we reassign these reference copies then the changes will not be reflected to the method calling this swap(x,y).

```
// This code will never work as intended
public void swap(Integer x, Integer y){
      Integer tmp =x;
       x=y;
       y=tmp;
   }
```

TIP *The called method can't change the caller's variable, although for object reference variables, the called method can change the object the variable referred to.*

The only way to have this possible was using some kind of setter on Integer class which could have modified the underlying value. But Java declares all Wrapper classes as Immutable for thread-safety perspective, thus there is no way to swap Integers in Java.

Question : What all collections use hashcode() method ?

SOLUTION

Only hashing data structures uses hashcode() method along with equal() though the later is used by many others collections as well.

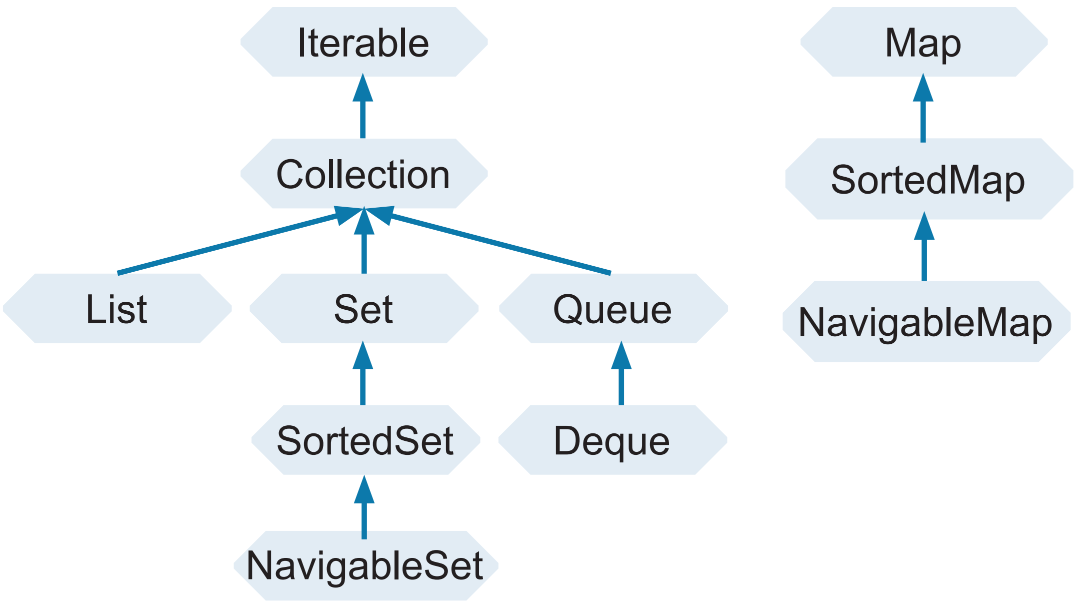
hashcode is useful for creating hashing based datastructure like HashMap, LinkedHashMap, ConcurrentHashMap, HashSet. Basically any Java collection that has Hash inside the name of it.

Hashcode is used to provide best case O(1) time complexity for searching the stored element.
TreeMap, TreeSet uses Comparator/Comparable for comparing the elements against each other.

Question : Provide a diagram for collections framework.

SOLUTION

Collections framework contains two main Interfaces Collection and Map. Collection is further extended by List, Set and Queue Interface.



Java Collections Framework Overview

There is a separate utility class named *Collections* which provides various static factory methods for playing with collections (Algorithm Part)

There are multitude Implementations provided for the above mentioned interfaces, few implementations implements more than one such Interface.

Set - A collection that does not allow duplicate elements (models mathematical set abstraction) and represents entities such as courses making up of student's schedule, ISBN number of books, Social security Number, PAN number, processes running on a machine, etc

List - A collection that maintains order of its elements. Lists can contain duplicate elements. ListIterator provides precise control over where to add the new item to the collection.

Queue - Queue is a First In First Out data structure which maintains order of its original elements. Most List implementations like LinkedList implements Queue interface as well.

Question : What is Immutable Class. Why would you choose it ? How would you make a class immutable ?

SOLUTION

What is Immutable Object

When the state of object can not be changed after its construction then the object is called Immutable.

Why do we need it

Immutable objects are inherently thread-safe, thus help writing multi-threading code without much worries. Immutable questions are meant for multi-threading program. *If someone is talking bout immutability then indirectly he is talking about multi-threaded context*. Immutable classes are easy to understand, as they possess a single state, which is controlled by their constructor. Immutable objects are good candidate for hash keys because their hashcode can be cached and reused for better performance.

Which Objects should be Immutable

Immutable classes are ideal for representing ADT's (Abstract Data Type) value.

Joshua Bloch suggests that

"*All classes should be designed to be immutable unless there is a specific reason not to do so*"

Guidelines for Making a class Immutable

1. All fields should be declared final
2. Class itself is declared final so that the derived classes do not make it Mutable.
3. ***this*** reference should not be allowed to escape during object construction such as in anonymous inner classes (for example adding action listener)
4. Any field that contains reference to mutable objects (such as arrays, collections, StringBuffer, etc)
   i. Are private
   ii. Are never returned or exposed to the caller
   iii. Are the only reference to the Objects that they refer
   iv. Do not change the state of the referenced object after the construction.
   v. If mutable fields must be returned to the caller, then a defensive copy should be returned so that the changes do not reflect in the inner data structure.
   ```
   public List<String> getList() {
      return Collections.unmodifiableList(list);   <=== defensive copy of the mutable field before returning it to caller
   }
   ```
   vi. If a mutable Object is passed in the constructor (like an array), then Immutable class should first make a defensive copy of the mutable object before storing its reference.

TIP *How does this reference escape during object construction ?*
When we construct a non static anonymous inner class, the class always has a pointer to the outer class, thus making all the fields available to that inner class. The compiler substitutes the reference of the outer class on behalf of us. You can find more details here - http://www.javaspecialists.eu/archive/Issue192b.html

An Object is said to be Immutable when,

Its state can't be changed after construction, all its fields are final and this reference does not escape during construction.

Example of Immutable Classes in JDK

All primitive Wrapper classes (Number, Integer, Long, Float, Double, etc), String Class, Color Class, BigInteger & BigDecimal class, CopyOnWriteArrayList & CopyOnWriteArraySet

Question : Why shouldn't we prefer mutable static variables in our Java Code ?

SOLUTION

Using mutable static variables might introduce Bugs in your software at some point in time

- Problem sharing a mutable static variable in multi-threaded environment. It's very tough to write & maintain a thread safe code with Mutable non-private static fields.
- Problem in Single Threaded design because we have to be very careful while updating static variable, since the next bit of code might expect some other state for the same.
- Code that relies on static objects can't be easily unit tested, and statics can't be easily mocked and hence does not promote TDD.

TIP - *If you are using static keyword without final for declaring a fields then you should reconsider your design, since the mutable static fields can be just dangerous !!*

For more details please refer to -
http://www.offthehill.org/articles/2011/06/03/java-static-variables-are-wrong-almost-always/

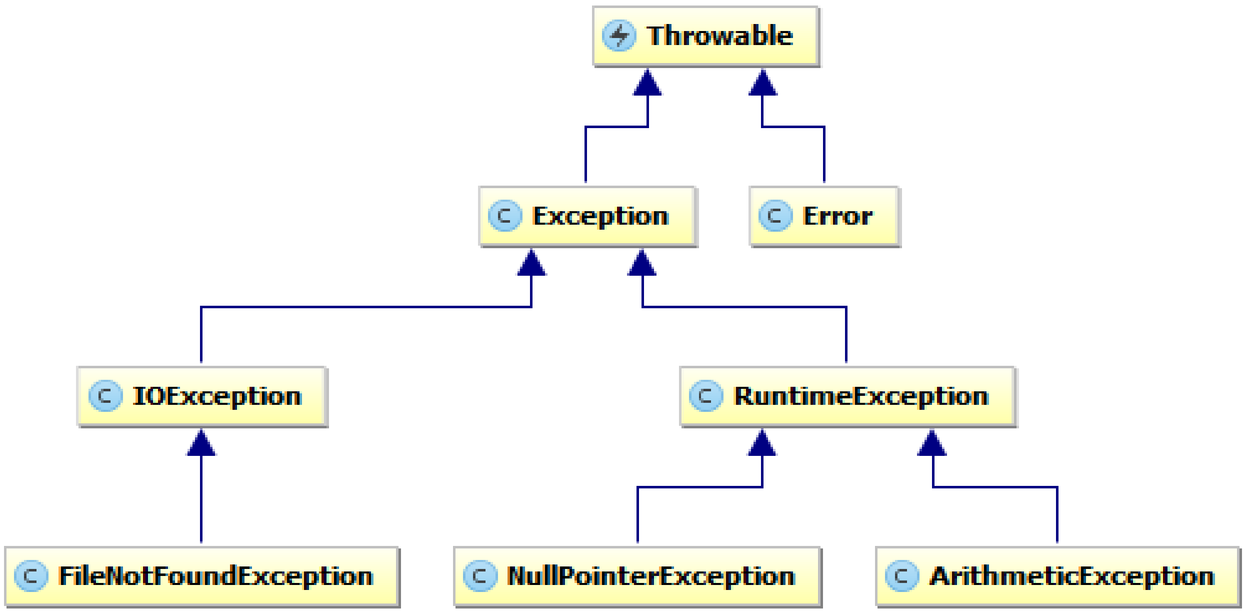
Question : Discuss Exception class hierarchy in Java.

SOLUTION

Checked Exceptions Represents exceptional scenario which if occurred, must dealt with in some way. example is IOException, FileNotFoundException. We need to declare these exceptions along with the code dealing with such scenarios.

Unchecked/Runtime Exceptions Represents an error in our program's logic which can not be reasonably recovered from at run time, for example NullPointerException, ArrayIndexOutOfBoundsException. We do not need to declare such exception in the method signature because these are not expected by any programmer.

Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch



Question : How does method parameter passing works in Java ?

SOLUTION

Method parameters are always passed by value in Java language irrespective of the type of variable (primitive or objects).
Actually Java always passes a copy of the bits in the variable. So for a primitive variable, you're passing a copy of the bits representing the value. For example, if you pass an int variable with the value of x, you're passing a copy of the bits representing x. The called method then gets its own copy of the value, to do with it what it likes.

And if you're passing an object reference variable, you're passing a copy of the bits representing the reference to an object. The called method then gets its own copy of the reference variable, to do with it what it likes. But because two identical reference variables refer to the exact same object, if the called method modifies the object (by invoking setter methods, for example), the caller will see that the object the caller's original variable refers to has also been changed.

The bottom line on pass-by-value: the called method can't change the caller's variable, although for object reference variables, the called method can change the object the variable referred to. What's the difference between changing the variable and changing the object?

That's the reason we can never write a method in Java which can swap two Integers.

Question : Why Prime Numbers are given much importance in writing certain algorithms like hashcode() ?

SOLUTION

Prime numbers are very useful for generating hashcode, RSA algorithms, random number generators.
String class's hashcode method multiplies its hash value by prime number 31 :

```
 int hash =0;
for (char ch : str.toCharArray()) {
   hash = 31 * hash + ch;
}
```

A number is either prime number or a composite number (can be factorized into prime numbers). Prime numbers are always unique and can not be divided by any other number except 1. The product of prime number with any other number has the best chances of being unique (though not as unique as Prime number itself) due to the fact that prime number is used to compose it. This property makes them very suitable for use in hashing function so as to obtain *fair distribution* in its hashcode output and thus achieving low collisions. Multiplying by the prime number will not tend to shift information away from the low end, as it would multiplying by a power of 2, thus achieving a fair randomness.

Question : What is instance level locking and class level locking ?

SOLUTION

In Java, every object has a built in lock (or monitor) that gets activated when Object has synchronized method code. It is basically of two types

Instance Lock

When we enter a non-static synchronized code then JVM acquires instance level lock on the Object whose method we are executing. Instance lock can also we acquired using the following syntax with block level synchronization.

Notes about instance locking

- Lock is mutually exclusive, which means that only one thread can acquire it and other threads have to wait for their turn until first thread releases it.
- Each Java object has just one lock (or monitor)
- Non-synchronized methods (& a single static synchronized method) can be executed in parallel with a single synchronized method.
- If a thread goes to sleep then it holds any lock it has.
- A single thread can acquire multiple lock on multiple objects.
- Lock is Reentrant, meaning that same thread can acquire the same lock multiple times (2^{31} times)

```
private static final Object lock = new Object();
synchronized(lock){
…
}
```

Class Lock (Locking on static methods or on Class object)

It is a lock acquired over java.lang.Class of an Object and is used to protect static methods of that class from thread safety point of view. There are two ways to acquire such lock - First one is using a static synchronized method and second one using block level synchronization over Object.getClass() as shown below.

```
synchronized(MyClass.class){
…
}
```

Example

Lets take this sample example to understand class and instance level Lock.

```
public class MyClass{
public static synchronized classMethod(){ ... }

public synchronized void instanceMethod(){ ... }
}
private MyClass reference = new MyClass();
```

1. One Thread can call reference.classMethod() and other thread can call reference.instanceMethod() in parallel because class level and instance level locks do not interfare.
2. But both the threads can't call the same instanceMethod() or classMethod() in parallel, because of the Mutual Exclusiveness of the Instance Lock and Class Lock.

Question : Explain threading jargons ?

SOLUTION

Race Condition
A race condition occurs when the correctness of a computation depends on the relative timing of multiple threads by the runtime. In this scenario Getting the right result relies on the lucky timings.

Dead Lock
Dead lock occurs when two or more threads are blocked forever, waiting for each other to release up the shared resource. For two threads, it happens when two threads have a circular dependency on a pair of synchronized shared resources.

Starvation
Describes a situation where a thread is unable to gain regular access to shared resource and is unable to make any progress
This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.

Mutex
Mutex stands for mutually exclusive, only one kind of operation (READ or WRITE) is allowed at a given time frame.

Live Lock
A thread often acts in response to the action of other threads, if the other thread's action is also in response to another thread, then live lock may result. No progress but threads are not blocked.

Synchronizer
A synchronizer is any object that coordinates the control of flow of threads based on its state.

Latch
A synchronizer that can delay the progress of threads until it reaches the terminal state.

Semaphore
Counting semaphore are used to control the number of activities that can access a certain shared resource or perform a given action at the same time. Semaphores are normally used implement resource pool or to impose a bound on a collection.

Exchanger
A two party barrier in which the parties exchange data at the barrier point.

Question : What is float-int implicit conversion while doing calculation on mixed data type in Java?

SOLUTION

As per Java Language Specifications

If at least one of the operands to a binary operator is of floating-point type, then the operation is a floating-point operation, even if the other is integral.

For example let's consider the following example

```
int i1 = 5;
float f = 0.5f;
int i2 = 2;
System.out.println(i1 * f);              // Result will be a float
System.out.println(i1 / i2);             // Result will be an integer
System.out.println(((float) i1) / i2);   // Result will be a float
```

Result

```
2.5
2
2.5
```

Primitive Casting
Casting lets you convert primitive values from one type to another. Casts can be explicit (narrowing conversions) or implicit (widening the conversions).
Compiler does implicit conversion when you try to put smaller item into bigger bucket but not the other way.

By default all literal integers are implicitly interpreted as int by the compiler. for example,

```
int x = 27;      //Literal assignment
```

Question: discuss Comparable and Comparator ? Which one should be used in a given scenario ?

SOLUTION

Comparable and Comparator both are used for allowing sorting a collection of objects.

Comparable should be used to define the natural ordering behavior of an Object.

Comparator should be used to provide an external controllable ordering behavior which can override the default ordering behavior (natural ordering) and when we might require different type of ordering behavior for the same Object. Comparator is implemented like an Adaptor Design Pattern where a separate class is dedicated for providing the comparison behavior.

Question : Will WeakHashMap's entry be collected if the value contains the only strong reference to the key ?

SOLUTION

```
public class Key {};
public class Value{
        final public Key key;
        public Value (Key key){
        this.key = key;
        }
}
```

It will not be collected.

Let's understand why ?

The value object in the WeakHashMap are held by ordinary strong references. Thus care must be taken to ensure that Value objects do not strongly refer to their own keys, either directly or indirectly, since that will prevent the keys from being garbage collected. If it is strongly required to store the key's reference in Value object then we can assign a Weak reference of Key in value object, as shown follow.

```
final public WeakReference<Key> key;
public Value(Key key){
        this.key = new WeakReference<Key>(key);
}
```

Question : Why HashMap's initial capacity must be power of two ?

SOLUTION

HashMap's initial capacity must be power of 2, and its default value is set to 16.
The reason is that, hashmap utilizes binary shift operation for calculating the modulus (%) of x/n for optimization reasons. But the bitwise hack expects the initial capacity to be power of two.

Even if we specify a custom initial capacity, it will round it to the nearest power of two using the following code.

```
int capacity = 1;
while(capacity < intialCapacity){
capacity = capacity << 1;
}
```

Question : Can we traverse the list and remove its elements in the same iteration loop ?

SOLUTION

Yes, that is feasible, provided

1. No other thread is modifying the collection at that traversal time (it should be single threaded model)
2. Iterator is used to traverse and to remove the elements from within that loop

Here is the perfect working example,

```
public class RemoveViaIterator {
    private List<String> names = new ArrayList<>(asList("1st", "2nd", "3rd", "4th"));

    public void remove(){
        Iterator<String> iterator = names.iterator();
        while (iterator.hasNext()) {
            Object next = iterator.next();
            System.out.println("next = " + next);
            iterator.remove();
        }
        System.out.println(names.size());
    }

    public static void main(String[] args) {
        RemoveViaIterator test = new RemoveViaIterator();
        test.remove();
    }
}
```

Question : Do I need to override object's equals() and hashcode() method for its use in a TreeMap ?

SOLUTION

No

Only hashing data structures require equals() and hashcode() method for achieving Big O(1) for data retrieval. TreeMap, on the other hand uses comparator for sorting its elements. HashMap, hashtable, ConcurrentHashMap, LinkedHashMap are few of the hashing data structures which require hashcode() and equals() method.

Question : Implement a BlockingQueue using intrisic locking mechanism.

SOLUTION

Here is the rough implementation of Blocking Queue using Java Intrisic Locking aka synchronized keyword.

```
public class BlockingQueue<T> {
    private Queue<T> queue = new LinkedList<T>();
    private int capacity;
    public BlockingQueue(int capacity) {
        this.capacity = capacity;
    }
    public synchronized void put(T element) throws InterruptedException {
        while(queue.size() == capacity) {
            wait();
        }
        queue.add(element);
        notify();
    }
    public synchronized T take() throws InterruptedException {
        while(queue.isEmpty()) {
            wait();
        }
        T item = queue.remove();
        notify();
        return item;
    }
}
```

Question: Is there a way to acquire a single lock over entire ConcurrentHashMap object ?

SOLUTION

Acquiring a single lock over entire ConcurrentHashMap may not serve any purpose to us, because CHM uses multiple internal locks to make it thread-safe. CHM's dynamic re-sizing recursively acquires lock over all of its buckets in order to make them bigger, but that's of no concern to the API user.
If we really want a single lock for the entire collection, then we should really prefer synchronized hashmap instead of ConcurrentHashMap.

Question : Implement a blocking queue using Lock and Condition Interface.

SOLUTION

The same implementation using the Lock and Condition class. This class uses a different semantics for achieving the same functionality.

```
public class BlockingQueue<T> {
    private Queue<T> queue = new LinkedList<T>();
    private int capacity;
    private Lock lock = new ReentrantLock();
    private Condition notFull = lock.newCondition();
    private Condition notEmpty = lock.newCondition();

    public BlockingQueue(int capacity) {
        this.capacity = capacity;
    }

    public void put(T element) throws InterruptedException {
        lock.lock();
        try {
            while(queue.size() == capacity) {
                notFull.await();
            }

            queue.add(element);
            notEmpty.signal();
        } finally {
            lock.unlock();
        }
    }

    public T take() throws InterruptedException {
        lock.lock();
        try {
            while(queue.isEmpty()) {
                notEmpty.await();
            }

            T item = queue.remove();
            notFull.signal();
            return item;
        } finally {
            lock.unlock();
        }
    }
}
```

Question : What is difference between intrinsic synchronization and explicit locking using Lock ?

SOLUTION

JVM provides intrinsic synchronization through monitor locks. Each object in Java owns a monitor on which the threads can be synchronized. JDK 1.5 introduced concept of explicit synchronization using Lock[1] and Condition classes which offers advanced features over intrinsic synchronization[2].

```
public interface Lock {
    void lock();
    void lockInterruptibly() throws InterruptedException;
    boolean tryLock();
    boolean tryLock(long time, TimeUnit unit) throws InterruptedException;
    void unlock();
    Condition newCondition();
}
```

Usage semantics for Lock

```
Lock lock = ...;
        if (lock.tryLock()) {
           try {
              // manipulate protected state
           } finally {
              lock.unlock();
           }
        } else {
           // perform alternative actions
        }
```

Intrinsic synchronization	Explicit Locking using Lock and Condition
• Its easy to use technique with code more readable and compact • It is not possible to interrupt a thread waiting to acquire a lock, or attempt to acquire a lock without being willing to wait for it forever. • Responsibility of releasing a lock is handled by JVM even in case a exception occurs • JVM can do some performance optimization if synchronized keyword is used like lock elision & lock coarsening • As per Java Documentation "*The use of synchronized methods or statements provides access to the implicit monitor lock associated with every object, but forces all lock acquisition and release to occur in a block-structured way: when multiple locks are acquired they must be released in the opposite order, and all locks must be released in the same lexical scope in which they were acquired.*"	• Provides same mutual exclusion & memory visibility guarantee as the synchronized block • Provides option for timed, polled or Interruptible locks helping avoid probabilistic deadlock. lock.tryLock() used for polling tryLock(long time, TimeUnit unit) for timed locking lockInterruptibly() for interruptible locking Interruptible lock acquisition allows locking to be used within cancelable activities. • Offers choice of fairness to the lock acquisition when multiple threads try to acquire the shared lock setting fairness flag to true as shown below. Lock lock = new ReentrantLock(true); This has significant performance cost when sued. • Ability to implement non-block-structured locking. Lock doesn't have to be released in the same block of code, unlike synchronized locks. • Lock has to be released manually in a finally block once we have modified the protected state

1 http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Lock.html
2 refer to http://denizdemir.com/2010/07/05/synchronization/

Question: What is difference between Callable and Runnable Interface ?

SOLUTION

As per Java documentation :

"*Callable interface is similar to Runnable, in that both are designed for classes whose instances are potentially executed by another thread. A Runnable, however, does not return a result and cannot throw a checked exception.*"

```
public interface Callable<V> {
        V call() throws Exception;
}
```

In order to convert Runnable to Callable use the following utility method provided by Executors class

```
Callable callable = Executors.callable(Runnable task);
```

Callable, however must be executed using a ExecutorService instead of Thread as shown below.

```
result = exec.submit(aCallable).get();
```

Submitting a callable to ExecutorService returns Future Object which represents the lifecycle of a task and provides methods to check if the task has been completed or cancelled, retrieve the results and cancel the task.

Here is the source for Future Interface

```
   public interface Future<V> {
      boolean cancel(boolean mayInterruptIfRunning);
      boolean isCancelled();
      boolean isDone();
      V get() throws InterruptedException, ExecutionException;
      V get(long timeout, TimeUnit unit)
         throws InterruptedException, ExecutionException, TimeoutException;
   }
```

Question : What will happen when an exception occurs from within a synchronized code block ?

SOLUTION

When an exception occurs from within a synchronized code block, then JVM smartly releases all the locks acquired by the current thread and will start unwinding the execution stack, till the exception is handled using catch block, otherwise killing the thread.

But the same does not happen when we write explicit locking code using Lock interface. In that case we need to release the lock manually in the finally block.

Question : There is a object state which is represented by two variables. How would you write a high throughput non-blocking algorithm to update the state from multiple threads ?

SOLUTION

Non-blocking algorithms provide better throughput at low thread contention compared to the locking counterpart. This can only be achieved in Java using CAS[1] (compare and swap) utilities provided in atomic package. AtomicReference along with Immutable object can be used to write a non-blocking algorithm maintaining a current state of Trade Volume.

There are key points to be noted while writing non-blocking algorithm[2] are:

- Immutability of TradeVolume as in below example is must to ensure proper initialization at it's assignment time. Immutability is achieved by making all fields final and providing constructor initialization.
- compareAndSet operation must be called repetitively in a while loop till the time it returns true.

```
public class NonBlockingTradeUpdate {
    /***
     * This TradeVolume Class must be Immutable otherwise it may create Java Memory Model Problems
     * while using with Atomic Reference. All the fields must be final to guarantee the initialization
     * and assignment at the same time from other thread.
     */
    @Immutable
    private static class TradeVolume {
        final long quantity;
        final long price;
        private TradeVolume(long quantity, long price) {
            this.quantity = quantity;
            this.price = price;
        }
    }
    private final AtomicReference<TradeVolume> tradeVol = new AtomicReference<>(new TradeVolume(100, 200));
    public long getQuantity() {
        return tradeVol.get().quantity;
    }
    public long getPrice() {
        return tradeVol.get().price;
    }
    /**
     * A non-blocking update method which updates the TradeVolume Object using AtomicReference.
     * This method is likely to perform better under multi-core environment with low thread contention.
     * @param quantity Quantity of the Trade
     * @param price    Price of the Trade
     */
    public void update(long quantity, long price) {
        while (true) {
            TradeVolume oldValue = tradeVol.get();
            TradeVolume newValue = new TradeVolume(quantity, price);
            if (tradeVol.compareAndSet(oldValue, newValue))
                return;
        }
    }
}
```

1 See Concurrency In Practice chapter 15.3 - Atomic Variable Classes
2 https://www.ibm.com/developerworks/java/library/j-jtp04186/

Question : What is difference between sleep(), yield() and wait() method ?

SOLUTION

Object.wait() and Thread.sleep() are entirely two different method used in two different contexts. Here are few differences between these two methods.

1. wait() method releases the acquired lock when the thread is waiting till someone calls notify() while Thread.sleep() method keeps the lock even if thread is waiting.

```
synchronized(monitor) {
   Thread.sleep(1000); // LOCK is held by the current thread
}
synchronized(monitor) {
   monitor.wait(); // LOCK is released by current thread
}
```

2. wait() can only be called from synchronized context otherwise it will throw IllegalMonitorStateException, while sleep can be called from any code block.

3. wait() is called on an Object while sleep is called on a Thread

4. waiting thread can be awaken by calling notify()/notifyAll() methods while sleeping thread can't be awaken[1] (though can be interrupted)

5. Incase of sleep() Thread immediately goes to Runnable state after waking up while in case of wait(), waiting thread first fights back for the lock and then go to Runnable state.

6. Major difference between yield and sleep in Java is that yield() method pauses the currently executing thread temporarily for giving a chance to the remaining waiting threads of the same priority to execute. If there is no waiting thread or all the waiting threads have a lower priority then the same thread will continue its execution.

sleep(n) *says "I'm done with my timeslice, and please don't give me another one for at least n milliseconds." The OS doesn't even try to schedule the sleeping thread until requested time has passed.*

yield() *says "I'm done with my timeslice, but I still have work to do." The OS is free to immediately give the thread another timeslice, or to give some other thread or process the CPU the yielding thread just gave up.*

wait() *says "I'm done with my timeslice. Don't give me another timeslice until someone calls notify()." As with sleep(), the OS won't even try to schedule your task unless someone calls notify() (or one of a few other wakeup scenarios occurs).*

Read more: *http://javarevisited.blogspot.com/2011/12/difference-between-wait-sleep-yield.html*

1 Thread.interrupt() would cause InterruptedException on both sleep() and wait() methods.

Question : Where should we use GET, PUT, POST and DELETE method ?

SOLUTION

To retrieve a resource, use GET
To create a resource on the server, use POST
To change the state of a resource or to update it on the server, use PUT
To remove or delete a resource on server, use DELETE

References

https://www.ibm.com/developerworks/webservices/library/ws-restful/

Question : What is difference between HashMap, TreeMap and LinkedHashMap ?

SOLUTION

All three classes (HashMap, TreeMap and LinkedHashMap) implements Map interface, and therefore represents mapping from unique key to values. But there are different usage scenarios for each of them -

1. HashMap is a hashing data structure which works on hashcode of keys. Keys must provide consistent implementation of equals() and hashCode() method in order to work with hashmap. Time complexity for get() and put() operations is Big O(1).

2. LinkedHashMap is also a hashing data structure similar to HashMap, but it retains the original order of insertion for its elements using a LinkedList. Thus iteration order of its elements is same as the insertion order for LinkedHashMap which is not the case for other two Map classes. Iterating over its elements is lightly faster than the HashMap because it does not need to traverse over buckets which are empty. LinkedHashMap also provides a great starting point for creating a LRU Cache object by overriding the removeEldestEntry() method, as shown in the following code snippet.

```
private static final int MAX_ENTRIES = 100;

protected boolean removeEldestEntry(Map.Entry eldest) {
   return size() > MAX_ENTRIES;
}
```

3. TreeMap is a SortedMap, based on Red-Black Binary Search Tree which maintains order of its elements based on given comparator or comparable. Time complexity for put() and get() operation is O (log n).

Question : How would you cancel a method execution after time-out expires using Java Future ?

SOLUTION

This can be easily achieved using Future utility class provided by concurrent package in JDK 1.5

Future represents the life cycle of a task and calling cancel() on future attempts to cancel the task execution, if not already in completed state. Thread can even be interrupted if we pass true as a parameter to cancel() method. And this interrupt can be checked using Thread.interrupted() method inside the running task.

JAVA Source

```
public class TimedExecution {
    private ExecutorService executorService = Executors.newCachedThreadPool();

    public void timedRun(Runnable runnable, long timeout, TimeUnit unit) throws InterruptedException,
ExecutionException {
        Future<?> task = executorService.submit(runnable);
        try {
            task.get(timeout, unit);
        } catch (TimeoutException e) {
            System.err.println("Timeout occurred.");
        } finally {
            task.cancel(true);
        }
    }

    public void stop(){
        executorService.shutdown();
    }

    public static void main(String[] args) throws ExecutionException, InterruptedException {
        TimedExecution timedExecution = new TimedExecution();
        timedExecution.timedRun(new Runnable() {
            @Override
            public void run() {
                while (!Thread.interrupted()){
                    System.out.println("Test me..");
                }
            }
        }, 100, TimeUnit.MICROSECONDS);
        timedExecution.stop();
    }
}
```

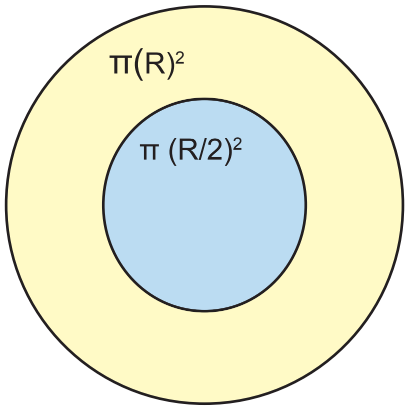
This task will be cancelled after 100 Microseconds. Do not forget to call task.cancel(true), otherwise the thread will continue executing the task in background.

Question : What is probability of a daat, hitting closer to centre of a circle rather than circumference ?

SOLUTION

Answer is 25%

Let's understand this question using the figure shown here. Daat will hit closer to centre of circle than the circumference when daat hits in a area whose radius is half the area of circle. The area of interest is shown in blue color in the given figure, and total area is the blue + yellow area.



Probability of hitting the daat closer to circle

= (expected area) / (total area)

= π x $(R/2)^2$ / (π x R^2)

=1/4

=25%

Question : How does an ArrayList expands itself when its maximum capacity is reached ?

SOLUTION

When the internal array of an ArrayList becomes full, then new array with double the capacity is created efficiently by the ArrayList using the following method.

```
elementData = Arrays.copyOf(elementData, newCapacity);
```

If it needs to shift the elements in order to add something over the existing index, then it displaces the elements using following System method -

```
System.arraycopy(elementData, index, elementData, index + 1, size - index);
```

Question : What is StringPool In Java ?

SOLUTION

JVM has a string pool where it keeps at most one object of any String. String literals objects are always created in the string pool for reusability purpose. String objects created with the new operator do not refer to objects in the string pool.

Pool generally resides in PermGen space as of JDK 1.6 (not much documentation is found on this)
The pooling concept has been made possible because the String objects are Immutable in Java.

Question : How does Generics work in Java ?

SOLUTION

Why to choose Generics ?

- Generics add stronger compile time type safety to our code, thus reducing the number of production bugs.
- It tries to eliminate explicit type casting to a greater extent, making our code more readable.
- Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods, allowing us to implement generic algorithms which works for different types.

Example of Generic class

class name<T1, T2, ..., Tn> { /* ... */ }

Naming Convention

E - element
K - key
N - number
T - type
V - value
S,U,V etc - 2nd, 3rd & 4th type

Can we add a Double value to List<Number> ?

Because an Integer is a kind of Number, so this is perfectly allowed due to inheritance.

```
Box<Number> box = new Box<Number>();
box.add(new Integer(10));   // OK
box.add(new Double(10.1));  // OK
```

Why List<String> can not be assigned to List<Object> ?

Let's consider the following example,

```
List<String> stringList =null ;
List<Object> objectList = stringList;   //ERROR
```

Here List<String> is not a sub type of List<Object>, hence as per inheritance rule, this is not allowed.

Inheritance in Generics - Can we assign List<String> to Collection<String> ?

If the type parameter is same, then we can very well extend the classes as per Java Inheritance rules. So it is perfectly legal to assign List<String> to Collection<String> reference.

Can we do T a = new T[100][1] ? Why not ?

No, this is not possible, because Arrays in Java contains Type information, at runtime, about its component type. So we must know the component type when we create an array. Because Type information is not retained at runtime using generics, thus we can't create an array like this. Please refer to Type Erasure in Generics Java for more information.

What is Type Erasure ?

Generics provide compile time safety to our Java code. Type erasure happens at compile time, to remove those generic type information from source and adds casts needed and deliver the byte code. Thus the java byte code will be no different than that the non-generic Java code.

1 http://stackoverflow.com/questions/2927391/whats-the-reason-i-cant-create-generic-array-types-in-java

Question : What are Upper and Lower bounds in Generics ? Where to choose one ?

SOLUTION

Upper and Lower bounded wildcard are used in Generics to relax the restriction on a variable. For example, if we want to write a method that accepts Integer, Double and Number, then we can use Upper bounded wildcard.
Below is the sample signature of upper bounded wildcard method.

```
public static void process(List<? extends Number> list) { /* ... */ }
```

Similarly, the lower bounded wildcard restricts the unknown type to be a specific type or super type of that type. Say you want to write a method that puts Integer objects into a list. To maximize flexibility, you would like the method to work on List<Integer>, List<Number>, and List<Object> — anything that can hold Integer values.

```
public static void addNumbers(List<? super Integer> list) {/*.....*/}
```

When to choose what ?

Lets first define In and Out terminology
In variable
An "in" variable serves up data to the code. Imagine a copy method with two arguments: copy(src, dest). The src argument provides the data to be copied, so it is the "in" parameter.
Out Variable
An "out" variable holds data for use elsewhere. In the copy example, copy(src, dest), the dest argument accepts data, so it is the "out" parameter.

Wildcard Guidelines

•An "in" variable is defined with an upper bounded wildcard, using the extends keyword.
•An "out" variable is defined with a lower bounded wildcard, using the super keyword.
•In the case where the "in" variable can be accessed using methods defined in the Object class, use an unbounded wildcard.
•In the case where the code needs to access the variable as both an "in" and an "out" variable, do not use a wildcard.

Multiple Bounds

A type parameter can have multiple bounds as described below.
<T extends B1 & B2 & B3>

References

http://docs.oracle.com/javase/tutorial/java/generics/wildcardGuidelines.html

Question : Discuss memory visibility of final fields in multi-threading scenario.

SOLUTION

As per Java Lang Specifications for final fields[1]

"Fields declared final are initialized once, but never changed under normal circumstances. The detailed semantics of final fields are somewhat different from those of normal fields. In particular, compilers have a great deal of freedom to move reads of final fields across synchronization barriers and calls to arbitrary or unknown methods. Correspondingly, compilers are allowed to keep the value of a final field cached in a register and not reload it from memory in situations where a non-final field would have to be reloaded.

final fields also allow programmers to implement thread-safe immutable objects without synchronization. A thread-safe immutable object is seen as immutable by all threads, even if a data race is used to pass references to the immutable object between threads. This can provide safety guarantees against misuse of an immutable class by incorrect or malicious code. final fields must be used correctly to provide a guarantee of immutability.

An object is considered to be completely initialized when its constructor finishes. A thread that can only see a reference to an object after that object has been completely initialized is guaranteed to see the correctly initialized values for that object's final fields.

The usage model for final fields is a simple one: Set the final fields for an object in that object's constructor; and do not write a reference to the object being constructed in a place where another thread can see it before the object's constructor is finished. If this is followed, then when the object is seen by another thread, that thread will always see the correctly constructed version of that object's final fields. It will also see versions of any object or array referenced by those final fields that are at least as up-to-date as the final fields are."

Briefly speaking

Memory visibility of final fields are guaranteed by the Java Memory Model and hence are thread safe in multi-threaded scenario. Thus we prefer to use Immutable objects in the concurrent application to avoid any memory visibility related problems.

Notes

Question

We have a legacy application in which a thread reads a config file and starts creating beans. It does so by first creating the object and then setting the required properties on them. All this happens without any synchronization because everything is happening serially. Once the objects are created, other threads pick those objects and start processing. But somehow, we got the problem that one of the client thread is not seeing the Correct Value for a bean.

Analysis

This could be a typical memory visibility issue in a multi-threaded environment. A properly constructed bean would have to make its fields final in absence of synchronization, otherwise other threads may see the default values for its fields. Thus moving the bean initialization code (setters) into constructor and making the fields final should solve this problem. Otherwise we might need to synchronize the access to this particular bean -

1 http://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html#jls-17.7

read articles about proper publishing of an object in multi-threaded environment.[2]

A Properly Constructed Object in Multi-threaded scenario[3]

The values of final fields are populated inside an object's constructor. And if the object is constructed properly (this reference does not escape during construction, don't start thread from within constructor, fields are final), then the values assigned to the final fields in the constructor are guaranteed to be visible to all other threads without the need of any synchronization. In addition, the visible values for any other object or array referenced by those final fields will be at least as up-to-date as the final fields. Let's understand the following example,

```
class FinalFieldExample {
  final int x;
  final ArrayList<String> names;
  int y;
  static FinalFieldExample f;
  public FinalFieldExample() {
    x = 3;
    y = 4;
    names= new ArrayList<>();
    names.add("First");
    names.add("Second");
  }

  static void writer() {
    f = new FinalFieldExample();
    f.names.add("third");
  }

  static void reader() {
    if (f != null) {
      int i = f.x;
      int j = f.y;
      ArrayList<String> myNames = f.names;
    }
  }
}
```

In this example, suppose Thread A calls FinalFieldExample.write() method and thus creates the object. Thread B on other hand calls FinalFieldExample.reader() method and thus access's the object. Further suppose that Thread B calls reader() once the writer() is finished creating the object. As per new Java Memory Model (JSR 133),

- Thread B executing the reader is guaranteed to see the value of 3 for field f.x - its a final primitive variable
- Thread B is guaranteed to see the value of "First" and "Second" for field f.names because names array is referenced by a final field, and the value assigned to such object inside the constructor boundary will be visible to other threads.
- There is not guarantee that Thread B will see "third" inside array f.names
- There is not guarantee that Thread B will see value of 4 for field f.y

2 http://www.ibm.com/developerworks/java/library/j-jtp0618/index.html
3 http://www.cs.umd.edu/~pugh/java/memoryModel/jsr-133-faq.html

Question : Where would you choose LinkedHashSet provided by Java Collections framework ?

SOLUTION

LinkedHashSet maintains the order of its original elements over and above the the uniqueness of its elements. Its more efficient when we iterate over its elements because it maintains the elements in a linked list. This is not possible with the HashSet because the in order to iterate over its elements, every bucket in a hash table must be visited whether it is occupied or not.

So we would choose it over HashSet when
1. We require the collection to preserve the original order of its elements.
2. We require better performance for iteration, next operation is performed in constant time O(1).

Question : What do you think is the reason for String Class to be Immutable ?

SOLUTION

String class in Java is implemented as Immutable and there are various reasons for doing that,

- Immutability brings inherent thread-safety to the usage of String class in a concurrent program. So multiple threads can work on the same String object without any data corruption. There is absolutely no need to synchronize the code because of String objects.
- StringPooling is possible only because of immutability because the underlying contents of the String will never change. This helps reducing the overall memory footprint of the application using lots of String objects.
- Hash code for Immutable objects are calculated lazily on first usage and then cached for future reference. This gives the benefit of performance when we use Immutable Key's in any hashing data structure.

Question : How is String concatenation implemented in Java using + operator ? for example, String name = "munish" + "chandel"

SOLUTION

String concatenation is implemented internally through the StringBuilder(as of JDK 1.5) class and its append method. As of Java 5, Java will automatically convert the following string concatenation

```
    String h = "hello" + "world";
```
to either
```
    String h ="helloworld";
```
or into
```
    String i = new StringBuilder().append("hello").append("world").toString();
```

Thus no temporary objects will be created for this type of concatenation. This JVM optimization may improved further in upcoming releases.

Does that mean, we should never prefer StringBuilder over String ?

No, we never meant that. StringBuilder has its own importance, but in slightly different scenario. Like the code we discussed above, if the same thing is happening inside a loop then O(n) StringBuilder objects will be created (one per iteration), causing overall time complexity Big O(n^2) where n is the number of strings that we are concatenating. So in that case, for performance reasons, we should create a single StringBuilder Object outside the loop and then append the data to that StringBuilder Object. That's the reason, experts advice to use StringBuilder inside a loop. Using StringBuilder, the code should look like this :

```
StringBuilder result = new StringBuilder(10000);
for(int i=0; i<=1000;i++){
    result.append("Hello"+i);
}
return result.toString();
```

Time Complexity : Big O (n) where n is the number of strings, 1000 in this case.

Question: Which Java data type would you choose for storing sensitive information, like passwords, and Why ?

SOLUTION

Normally, a character array should used for storing passwords. Here is the reason for choosing char array over String -

- There is no way to clear a String Object from the memory, its up to GC to collect it.
- String objects are immutable and stored in a String Pool (may reside inside a PermGen space) which may not at all be gc'd.
- Any person taking the heap dump can easily see the String literals.
- In case of an char array, we can always nullify it once we are done with the information, so not much dependency on the GC, thus we are narrowing the time window for the life of sensitive data.

Question : What is difference between using Serializable & Externalizable interface in Java ?

SOLUTION

- In case of Serializable, default serialization is used, while in case of Externalizable, the complete serialization control goes to the application. Stating that means, we can not benefit from the default serialization process when we choose Externalizable interface.
- We generally choose Externalizable when we want to save the output in our custom format which is other than Java default serialization format like, csv, database, flat file, xml, etc
- readExternel() and writeExternal() methods are used to handle the serialization in case of Externilizabel interface.
- In case of externalizable interface, we need to handle super type state, default values in transient variable and static variables.
- Incase of Serialization, object is reconstructed using data read from ObjectInputStream but incase of Externalizable, public no-arg constructor is used to reconstruct the object.
- Externalizable extends Serializable.

Question : How would you produce DeadLock in Java ?

SOLUTION

DeadLock happens in multi-threaded scenario when two more threads have mutual dependencies on two or more shared resources. Let's understand with the following code,

```
public class DeadLock {
    static class Resource {
        final String name;
        Resource(String name) {this.name = name;}
        synchronized void print() {
            System.out.println("this is resource " + name);
        }

        synchronized void print(Resource another) {
            try {TimeUnit.MILLISECONDS.sleep(100);} catch (InterruptedException e) {}
            System.out.println("Thread "+Thread.currentThread().getName()+" acquired resource " + name);
            another.print();                    ==> The line that could cause a deadlock
      }
    }

    public static void main(String[] args) {
        final Resource r1 = new Resource("r1");
        final Resource r2 = new Resource("r2");
        new Thread(new Runnable() {
            @Override
            public void run() {
                r1.print(r2);
            }
        }).start();
        new Thread(new Runnable() {
            @Override
            public void run() {
                r2.print(r1);
            }
        }).start();
    }
}
```

In the above code, two threads operate over two shared Resources r1 and r2. Resource class has two synchronized methods (which will require the threads to obtain lock over the instance) and unfortunately r1 has a inter-dependency on r2. There is a great probability that the above code will block for ever causing a deadlock.
Using jconsole we can detect the deadlock, below is the message shown in jconsole for this java process

```
Name: Thread-1
State: BLOCKED on org.shunya.power.interview.DeadLock$Resource@354949 owned by: Thread-0
Total blocked: 2  Total waited: 1
Name: Thread-0
State: BLOCKED on org.shunya.power.interview.DeadLock$Resource@661a11 owned by: Thread-1
Total blocked: 1  Total waited: 1
```

Chapter 3

Concurrency

Question : What is Concurrency, how is it implemented in Java Programs ?

SOLUTION

Concurrency is the property of an software program to run several computations in parallel. Java provides us with the multiple mechanisms to create Threads so as to utilize the multiple processor cores of a given hardware in order to achieve high throughput.

Java provides various ready to use utilities for writing concurrent programs, which otherwise is difficult to implement.

Some of the Utility classes provided as of JDK 1.6 are[1]
Executors, Queues, TimeUnit, Synchronizers (semaphore, latch, barrier, exchanger), Concurrent collections (ConcurrentHashMap, CopyOnWriteArrayList, ConcurrentSkipListMap), Atomic package for non-blocking algorithms, locks (ReentrantLock), and well defined Java Memory Model for memory consistency in concurrent environment.

What is Thread-Safety and how to achieve it ?
A Class is thread safe when it behaves correctly when accessed & modified from multiple threads in parallel without any changes in the code calling it.

There are certain ways to make our class thread-safe, as follow
1. Use synchronization mechanism (intrinsic or explicit) on the accessor methods
2. Make the class Immutable
3. Don't expose the shared state across threads (for e.g. Keep objects local to thread using ThreadLocal)

What is Synchronization ?
Synchronization avoids thread interference and memory consistency errors by providing serialized access to the shared state.

Synchronization has two major aspects
1. It makes sure that the compound actions executes atomically by providing mutually exclusive access to the shared state across the threads.
2. It ensures the memory consistency by making the changes visible to all the threads upon method exit.

Lets examine the following program for its correctness in concurrent environment, It maintains the integer counter.

Counter.java
```
@NotThreadSafe
class Counter {
        private int c = 0;
        public void increment() {
                c++;
        }
        public int value() {
        return c;
        }
}
```

1 http://www.ibm.com/developerworks/java/library/j-5things4/index.html

This program will work absolutely fine in single threaded environment but will not behave correctly in multi-threaded environment, because

1. increment() method will not be executed atomically so data race may corrupt the counter value.
2. value() method may not return the latest value of counter because of caching in processor's registers.

So lets make this program thread-safe.

```
Counter.java
@ThreadSafe
class Counter {
        private int c = 0;
        public synchronized void increment() {    //this will make the operation execution atomic across the threads
                c++;
        }
        public synchronized int value() {           //this will make sure the changes are visible to the calling thread
        return c;
        }
}
```

Please Make Sure

That you make the getter method as synchronized until the getter returns an immutable object. Otherwise the memory effects may not be consistent and the calling thread may see a stale value of the variable.

Weaker form of synchronization using volatile

Volatile variable can not make the method execution atomic, but it can make sure that the updates to the variable are propagated predictably to the other threads. volatile variables basically, are not cached into the processor registers, rather they are always fetched and written to the main memory on heap.

Question : There are two Threads A and B operating on a shared resource R, A needs to inform B that some important changes has happened in R. What technique would you use in Java to achieve this ?

SOLUTION

Object R's method wait(), notify() & notifyAll(), can be used for inter-thread communication. This will allow all threads which hold lock over R, to communicate among them selves. You can explore a typical Producer-Consumer problem to see how it works.

Question: What do you understand by Java Memory Model ? What is double-checked locking? What is different about final variables in new JMM ?

SOLUTION

Interviewer's Intent - *Interviewer wants to understand if you can write concurrent code.*

Java Memory Model[1] defines the legal interaction of threads with the memory in a real computer system. In a way, it describes what behaviors are legal in multi-threaded code. It determines when a Thread can reliably **see** writes to variables made by other threads. It defines semantics for volatile, final & synchronized, that makes guarantee of visibility of memory operations across the Threads.

Let's first discuss about Memory Barrier which are the base for our further discussions. There are two type of memory barrier instructions in JMM - read barriers & write barrier.

A read barrier invalidates the local memory (cache, registers, etc) and then reads the contents from the main memory, so that changes made by other threads becomes visible to the current Thread.
A write barrier flushes out the contents of the processor's local memory to the main memory, so that changes made by the current Thread becomes visible to the other threads.

JMM semantics for synchronized

When a thread acquires monitor of an object, by entering into a synchronized block of code, it performs a read barrier (invalidates the local memory and reads from the heap instead). Similarly exiting from a synchronized block as part of releasing the associated monitor, it performs a write barrier (flushes changes to the main memory)
Thus modifications to a shared state using synchronized block by one Thread, is guaranteed to be visible to subsequent synchronized reads by other threads. This guarantee is provided by JMM in presence of synchronized code block.

JMM semantics for Volatile fields

Read & write to volatile variables have same memory semantics as that of acquiring and releasing a monitor using synchronized code block. So the visibility of volatile field is guaranteed by the JMM. Moreover afterwards Java 1.5, volatile reads and writes are not reorderable with any other memory operations (volatile and non-volatile both). Thus when Thread A writes to a volatile variable V, and afterwards Thread B reads from variable V, any variable values that were visible to A at the time V was written are guaranteed now to be visible to B.

Let's try to understand the same using the following code

```
Data data = null;
volatile boolean flag = false;

Thread A
-------------
data = new Data();
flag = true;              <-- writing to volatile will flush data as well as flag to main memory

Thread B
-------------
if(flag==true){           <-- reading from volatile will perform read barrier for flag as well data.
use data;                 <--- data is guaranteed to visible even though it is not declared volatile because of the JMM
                               semantics of volatile flag.
}
```

Continued on 75

1 http://www.ibm.com/developerworks/library/j-jtp03304/

JMM semantics for final fields & Initialization safety

JSR 133 (new JMM with JDK 1.5 onwards) provides a new guarantee of initialization safety - that as long as an object is properly constructed (this reference does not escape during the construction), then all threads will see the correct value for its final fields that were set in its constructor, regardless of whether or not synchronized is used to publish the object from one thread to another. Further, any variable that can be reached through a final field of a properly constructed object, such as fields of an object referenced by a final field, are also guaranteed to be visible to the other threads. For example, if a final field contains reference to a ArrayList, in addition to the correct value of the reference being visible to other thread, also the contents of that ArrayList at construction time, would be visible to other threads without synchronization.
For all the final fields, when a constructor completes, all of the writes to final fields and to the variables reachable through those final fields becomes frozen, and any thread that obtains a reference to that object after the freeze is guaranteed to see the frozen values for all frozen fields. So it is a kind of *happen-before* relationship between the write of a final field in the boundary of constructor and the initial load of a shared reference to that object in another Thread.

Double-Checked Locking Problem

In earlier times (prior to JDK 1.6) a simple uncontended synchronization block was expensive and that lead many people to write double-checked locking to write lazy initialization code. The double-checked locking idiom tries to improve performance by avoiding synchronization over the common code path after the helper is allocated. But the DCL never worked because of the limitations of pervious JMM. This is now fixed by new JMM (JDK 1.5 onwards) using volatile keyword.

NonThreadSafe Singleton (This will not work under current JMM), so **never use it**

```
public class Singleton
{
  private Singleton() {}
  private static Singleton instance_ = null;        ==> A global static variable that will hold the state

 public static Singleton instance()
 {
  if(instance_==null)              ==> un-synchronized access to this fields may see partially constructed objects because
  {                                       of instruction reordering by the compiler or the cache
    synchronized(Singleton.class)
    {
      if(instance_==null)
        instance_= new Singleton();
    }
  }
  return instance_;
 }
}
```

JMM will not guarantee the expected execution of this static singleton.

Why above code idiom is broken in current JMM ?

DCL relies on the un synchronized use of _instance field. This appears harmless, but it is not. Suppose Thread A is inside sycnhronized block and it is creating new Singleton instance and assigning to _instance variable, while thread B is just entering the getInstance() method. Consider the effect on memory of this initialization. Memory for the new Singleton object will be allocated; the constructor for Singleton will be called, initializing the member fields of the new object; and the field resource of SomeClass will be assigned a reference to the newly created object. There could be two scenarios now

- Suppose Thread A has completed initialization of _instance and exits synchronized block as thread B enters getInstance(). By this time, the _instance is fully initalized and Thread A has flushed its local memory

Continued on 76

to main memory (write barriers). Singleton's member fields may refer other objects stored in memory which will also be flushed out.. While Thread B may see a valid reference to the newly created _instance, but because it didn't perform a read barrier, it could still see stale values of _instance's member fields.
- Since thread B is not executing inside a synchronized block, it may see these memory operations in a different order than the one thread A executes. It could be the case that B sees these events in the following order (and the compiler is also free to reorder the instructions like this): allocate memory, assign reference to resource, call constructor. Suppose thread B comes along after the memory has been allocated and the resource field is set, but before the constructor is called. It sees that resource is not null, skips the synchronized block, and returns a reference to a partially constructed Resource! Needless to say, the result is neither expected nor desired.

Fixed double-checked Locking using volatile in new JMM (multi-threaded singleton pattern JDK 1.5)

The following code makes the helper volatile so as to stop the instruction reordering. This code will work with JDK 1.5 onwards only.

```
class Foo {
        private volatile Helper helper = null;
        public Helper getHelper() {
            if (helper == null) {
                synchronized(this) {
                    if (helper == null)
                        helper = new Helper();
                }
            }
            return helper;
        }
    }
```

If Helper is an **immutable** object, such that all of the fields of Helper are final, then double-checked locking will work without having to use volatile fields. The idea is that a reference to an immutable object (such as a String or an Integer) should behave in much the same way as an int or float; reading and writing references to immutable objects are atomic.

Alternatives to DCL[2]

Now a days JVM is much smarter and the relative expense of synchronized block over volatile is very less, so it does not really make sense to use DCL for performance reasons.

The easiest way to avoid DCL is to avoid it. We can make the whole method synchronized instead of making the code block synchronized.
Another option is to use *eager initialization* instead of lazy initialization by assigning at the creation time
Here is the example demonstrating eager initialization

```
class MySingleton {
  public static Resource resource = new Resource();
}
```

Using Initialization On Demand Holder idiom[3]
Inner classes are not loaded until they are referenced. This fact can be used to utilize inner classes for lazy initialization as shown below

```
public class Something {
        private Something() {
        }
```

2 http://en.wikipedia.org/wiki/Double-checked_locking#Usage_in_Java
3 http://en.wikipedia.org/wiki/Initialization_on_demand_holder_idiom#Example_Java_Implementation

```
        private static class LazyHolder {
              private static final Something INSTANCE = new Something();
        }
        public static Something getInstance() {
              return LazyHolder.INSTANCE;
        }
  }
```

This code is guaranteed to be correct because of the initialization guarantees for static fields; if a field is set in a static initializer, it is guaranteed to be made visible, correctly, to any thread that accesses that class.

Using final wrapper to hold the Instance

Semantics of final field in Java 5 can be employed to safely publish the helper object without using volatile.

```
public class FinalWrapper<T> {
    public final T value;
    public FinalWrapper(T value) {
        this.value = value;
    }
}

public class Foo {
   private FinalWrapper<Helper> helperWrapper = null;
   public Helper getHelper() {
      FinalWrapper<Helper> wrapper = helperWrapper;
      if (wrapper == null) {
          synchronized(this) {
              if (helperWrapper == null) {
                  helperWrapper = new FinalWrapper<Helper>(new Helper());
              }
              wrapper = helperWrapper;
          }
      }
      return wrapper.value;
   }
}
```

The local variable wrapper is required for correctness.

And finally using Enum for Thread-Safe Singleton

```
public enum Singleton{
    INSTANCE;
}
```

References

http://www.ibm.com/developerworks/library/j-jtp03304/
http://en.wikipedia.org/wiki/Double-checked_locking#Usage_in_Java
http://en.wikipedia.org/wiki/Initialization_on_demand_holder_idiom#Example_Java_Implementation
http://en.wikipedia.org/wiki/Double-checked_locking#Usage_in_Java
http://jeremymanson.blogspot.co.uk/2008/04/immutability-in-java.html
http://stackoverflow.com/questions/5938163/singleton-using-atomicreference
http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html
http://www.javaworld.com/jw-02-2001/jw-0209-double.html
http://www.ibm.com/developerworks/java/library/j-jtp0618/index.html
http://www.ibm.com/developerworks/library/j-jtp02244/index.html
http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html
http://www.cs.umd.edu/~pugh/java/memoryModel/jsr-133-faq.html
http://www.cs.umd.edu/~pugh/java/memoryModel/jsr133.pdf

Question : Is i++ thread-safe ?

SOLUTION

No, because the ++ operator is non atomic.

Let's understand the following code -

```
public class Counter{
private int i;

public int increment(){
        i++;
}
}
```

The method increment() in the above code is not thread safe, because i++ require multiple cpu instruction cycles to compute the summation. Data race condition may happen if the shared object is incremented from multiple threads, simultaneously.

How To Make It thread-safe ?

There are mainly two ways to make it thread-safe in Java -

1. By making the increment method synchronized. (Preferred when thread contention is moderate to high)
2. By using AtomicInteger to maintain the increment, utilizing CAS under the hood. (Preferred for single CPU, and for low to moderate thread contention)

If you want, you can write your custom **AbstractQueueSynchronizer** to achieve the same, but that discussion is out of scope for this writing.

Question: What happens when wait() & notify() method are called ?

SOLUTION

When wait() method is invoked from a synchronized context, the following things happen

- The calling thread gives up the lock.
- The calling thread gives up the CPU.
- The calling thread goes to the monitor's waiting pool.

And in case of notify() method, following things happen

- One of the waiting thread (may be a random thread) moves out of the monitor's waiting pool.
- Thread comes into ready state (RUNNABLE).
- Tries its best to require the monitor lock before it can proceed to the method execution.

Question : What are the different states of a Thread ? What does those states tells us ?

SOLUTION

A thread in JVM can have 6 different states as defined in Thread.State enum. At any given time, thread must be in any of these states.

NEW
This state is for a thread which has not yet started.

RUNNABLE
This state is for the currently running thread which is executing in java virtual machine, but it may be waiting for the other resources from operating system such as processor.

BLOCKED
Thread state for a thread blocked waiting for a monitor lock. A thread in this state can be waiting for a monitor lock to enter a synchronized block/method or reenter a synchronized method after calling Object.wait.

WAITING
A thread is waiting due to calling on one of the method -
Object.wait with no timeout
Thread.join with no timeout
LockSupport.park

A Thread in this state is waiting for another thread to perform a particular action. For example, a thread that has called Object.wait() on an object is waiting for another thread to call Object.notify() or Object.notifyAll() on that object. A thread that has called Thread.join() is waiting for a specified thread to terminate.

TIMED_WAITING
Thread state for a waiting thread with a specified waiting time. A thread is in the timed waiting state due to calling one of the following methods with a specified positive waiting time -

Thread.sleep
Object.wait with timeout
Thread.join with timeout
LockSupport.parkNanos
LockSupport.parkUntil

TERMINATED
Thread state for a terminated thread. The thread has completed execution.

References

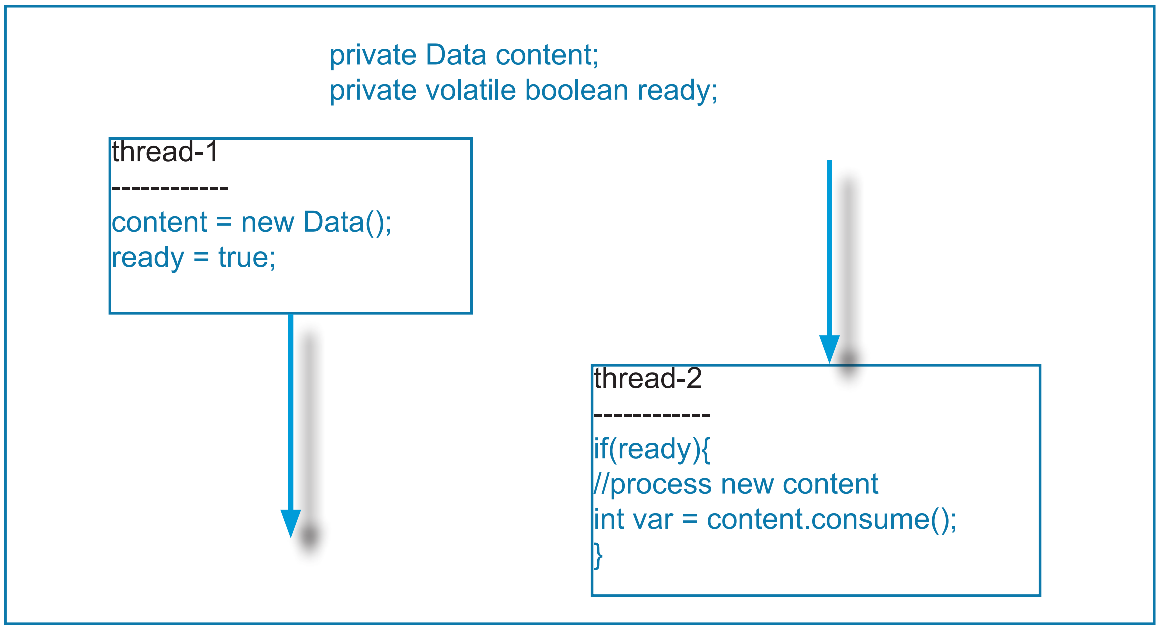
This content has been taken directly from the Java 7 Docs - Thread.State enum.

Question : Discuss about volatile keyword and Java Memory Model ?

SOLUTION

Volatile is a mechanism for lighter weight synchronization where memory visibility of the protected state is guaranteed to all consecutive threads.

A write to volatile variable not only flush changes of the volatile variable but all the non volatile variables changed before write to volatile. Thus a simple flag of volatile type can serve the memory visibility guarantee of all the other variables changed before. The following figure explain it in entirety.



```
private Data content;
private volatile boolean ready;
```

```
thread-1
------------
content = new Data();
ready = true;
```

```
thread-2
------------
if(ready){
//process new content
int var = content.consume();
}
```

Figure : *Effects of volatile variables on time scale*

Non-atomic treatment of long and double

For the purposes of the Java programming language memory model, a single write to a non-volatile long or double value is treated as two separate writes: one to each 32-bit half. This can result in a situation where a thread sees the first 32 bits of a 64-bit value from one write, and the second 32 bits from another write.

Writes and reads of volatile long and double values are always atomic.

Writes to and reads of references are always atomic, regardless of whether they are implemented as 32-bit or 64-bit values.

It is safe to perform read-modify-write operations on a shared volatile variable as long as you ensure that the volatile variable is only written from single thread.
volatile variables are liable to get into race conditions because atomicity is required to solve race condition

Question : What is a CAS ? How does it help writing non-blocking scalable applications ? Tell something about Atomic Package provided by Java 1.6

SOLUTION

Problem with Traditional Locking (using synchronized keyword or Lock objects)[1]

If a thread tries to acquire a lock that is already held by some other thread, then the thread has to block until the lock becomes available. This could lead to scalability hazards if the new thread was performing high priority tasks. Deadlock is second problem when dealing with locking due to inconsistency in acquiring multiple locks. Thirdly for managing very lightweight operations (in terms of CPU cycles) like increment counter & concurrent updates to a variable, acquiring lock could cause more overhead than the real computation logic.

Compare and Swap (CAS) for the rescue

It provides us with the finer-grained mechanism for managing lock-free thread safe concurrent updates to individual variables, after seeking some hardware level support to achieve the same. A CAS operation includes three commands - a memory location, the expected old value and a new value. The underlying processor will atomically update the given memory location to new value if there matches the expected old value, otherwise it will do nothing.

Atomic Package (java.util.concurrent.atomic)

Atomic package is a small toolkit of classes that exposes CAS operations and thus helping us write lock-free thread safe programming on single variables. Most atomic classes provide the following method for CAS

```
boolean compareAndSet(expectedValue, updateValue);
```

This method (which varies in argument types across different classes) atomically sets a variable to the updateValue if it currently holds the expectedValue, reporting true on success.

Memory effects of Atomic Classes

Memory effects for read and update of a atomic variable generally follow the rules for the volatile -

- get() has memory effects of reading a volatile variable.
- set() has memory effects of writing a volatile variable.
- compareAndSet, getAndIncrement has memory effects of read and write to volatile variable.

Lock-free and wait-free Algorithms using Compare and Swap (CAS)

In a lock-free algorithm, at least some thread always make progress
In a wait-free algorithm, every thread will continue to make some progress in face of arbitrary delay of other threads.
Below is a small example utilizing CAS for Implementing a non-blocking sequence generator.

```
class Sequencer {
  private final AtomicLong sequenceNumber
    = new AtomicLong(0);
  public long next() {
    return sequenceNumber.getAndIncrement();
  }
}
```

We should keep it in mind that CAS operations should be preferred to locking code only when :

1. The operation is very lightweight and confined to a single variable update
2. Thread contention is low to moderate, under heavy contention, performance will suffer dramatically, as the JVM spends more time dealing with scheduling threads and managing contention and queues of waiting threads and less time doing real work, like incremental counters.

1 http://www.ibm.com/developerworks/java/library/j-jtp11234/

Question : How would you implement AtomicFloat /AtomicDouble using CAS ?

SOLUTION

Java does not provide Atomic implementation for Double/Float as they have stated in their documentation[1]

"Atomic classes are not general purpose replacements for java.lang.Integer and related classes. They do not define methods such as hashCode and compareTo. (Because atomic variables are expected to be mutated, they are poor choices for hash table keys.) Additionally, classes are provided only for those types that are commonly useful in intended applications. For example, there is no atomic class for representing byte. In those infrequent cases where you would like to do so, you can use an AtomicInteger to hold byte values, and cast appropriately. You can also hold floats using Float.floatToIntBits and Float.intBitstoFloat conversions, and doubles using Double.doubleToLongBits and Double.longBitsToDouble conversions."

There are two ways to implement AtomicDouble/AtomicFloat.

First method is to use AtomicReference to hold Double value as shown in below snippet

```
import java.util.concurrent.atomic.AtomicReference;

public class AtomicDoubleUpdater {
    private final AtomicReference<Double> curr = new AtomicReference<>();

     public void add(Double aDouble) {
        for (; ; ) {
            Double oldVal = curr.get();
            Double newVal = oldVal + aDouble;
            if (curr.compareAndSet(oldVal, newVal))
                return;
        }
    }
}
```

Second method is to use AtomicInteger for storing Float bit values as hinted by above Java docs

```
import java.util.concurrent.atomic.AtomicInteger;
import static java.lang.Float.*;

class AtomicFloat extends Number {
    private AtomicInteger bits;
    public AtomicFloat() {
        this(0f);
    }

    public AtomicFloat(float initialValue) {
        bits = new AtomicInteger(floatToIntBits(initialValue));
    }

    public final boolean compareAndSet(float expect, float update) {
        return bits.compareAndSet(floatToIntBits(expect),
                                  floatToIntBits(update));
    }
```

1 http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/atomic/package-summary.html

```
    public final void set(float newValue) {
      bits.set(floatToIntBits(newValue));
    }

    public final float get() {
        return intBitsToFloat(bits.get());
    }

    public float floatValue() {
        return get();
    }

    public final float getAndSet(float newValue) {
        return intBitsToFloat(bits.getAndSet(floatToIntBits(newValue)));
    }

    public final boolean weakCompareAndSet(float expect, float update) {
        return bits.weakCompareAndSet(floatToIntBits(expect),
                              floatToIntBits(update));
    }

    public double doubleValue() { return (double) floatValue(); }
    public int intValue()       { return (int) get();           }
    public long longValue()     { return (long) get();          }

}
```

Notes

Further thoughts
How would you implement AtomicBigDecimal ?
Why didn't Java provide default implementation for AtomicDouble/AtomicFloat ?
When should be prefer CAS over traditional blocking synchronization ?

References
http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/atomic/package-summary.html
http://stackoverflow.com/questions/5505460/java-is-there-no-atomicfloat-or-atomicdouble
http://stackoverflow.com/questions/8567596/how-to-make-updating-bigdecimal-within-concurrenthashmap-thread-safe

Question : Can we implement check & update method (similar to compare and swap) using volatile alone ?

SOLUTION

No, this is not possible using volatile keyword alone. Volatile keyword can not guarantee atomicity of operation. It's a lighter weight synchronization which can guarantee memory visibility only.
The only way to implement CAS is either using synchronized block (Lock Interface as well) or using java provided hardware level CAS in it's atomic package i.e. using AtomicReference, AtomicInteger, etc

Question : You are writing a multi-threaded software piece for NSE for maintaining the volume of Trades made by its individual brokers (icici direct, reliance). It's highly concurrent scenario and we can not use lock based thread safety due to high demand of throughput. How would handle such scenario ?

SOLUTION

```
private ConcurrentHashMap<String, BigDecimal> sumByAccount;

this hashmap could contain entries like :
'ICICI Direct' -> 10000.00
'Reliance Money' -> 20000.00
```

Since the multiple threads could be simultaneously adding the value to their respective broker, the designed code should be thread safe. Un-Thread safe version looks like :

```
public void addToSum(String account, BigDecimal amount){
    BigDecimal newSum = sumByAccount.get(account).add(amount);
    sumByAccount.put(account, newSum);
}
```

Solution

CAS can be utilized for achieving the high throughput requirement of the underlying system in this case. AtomicReference<BigDecimal> could be used to store the BigDecimal value atomically.

```
ConcurrentHashMap<String, AtomicReference<BigDecimal>> map;

public void addToSum(String account, BigDecimal amount) {
   AtomicReference<BigDecimal> newSum = map.get(account);
   for (;;) {
      BigDecimal oldVal = newSum.get();
      if (newSum.compareAndSet(oldVal, oldVal.add(amount)))
          return;
   }
}
```

AtomicReference uses CAS to atomically compare and assign a single reference. In the above code the compareAndSet(oldVal, oldVal.add(amount)) method checks if the AtomicReference == oldVal (by their memory location instead of actual value), if true then it replaces the value of field stored in AtomicReference with the oldVal.add(amount). All this comparison and swapping happens atomically by the JVM. Afterwards invoking the newSum.get() will return the added amount.

For loop is required here because it is possible that multiple threads are trying to add to the same AtomicReference and doing so just one thread succeeds and other fails. The failed threads must try again the operation to make the addition to BigDecimal.

Please be noted that CAS is recommended for moderate Thread contention scenarios. Synchronized should always be preferred for high contention code blocks.

Question: Calculate the time spread for 10 threads - Suppose T1 started earliest and T5 finished last, then the difference between T5 and T1 will give time spread.

SOLUTION

This is a typical thread synchronization problem which can be solved using various available techniques in Java. We will discuss three main approaches to solve this problem - first one using a synchronized object, second one using non-blocking CAS, third using existing synchronizer CountDownLatch. Algorithm for the both is same - Two times will be recorded, first time for the thread which started earliest, and second time for the thread which finished last. The difference of the two times will give us time window.

Writing custom synchronizer to address this problem
We will write a custom synchronized class which records the first start time and last stop time.

```
public class TimeSpread2 {
    int threads;
    long startTime;
    long stopTime;
    boolean started = false;
    public TimeSpread2(int threads) {this.threads = threads;}
    public synchronized void start(){
        if(!started){
            started = true;
            startTime = System.currentTimeMillis();
        }
    }
    public synchronized void stop(){
      if(--threads<=0){
          stopTime = System.currentTimeMillis();
          notifyAll();
      }
    }
    public synchronized long timeSpread() throws InterruptedException {
        while(threads >0){          wait();   }
        return stopTime-startTime;
    }

    public static void main(String[] args) throws InterruptedException {
        int threads1 = 100;
        final TimeSpread2 timeSpread = new TimeSpread2(threads1);
        Runnable t = new Runnable(){
            public void run() {
                timeSpread.start();
                try {TimeUnit.SECONDS.sleep(5);} catch (InterruptedException e) {}
                timeSpread.stop();
            }
        };

        for(int i=0;i< threads1;i++){
            Thread thread = new Thread(t);thread.start();
        }
        System.out.println("time spread = " + timeSpread.timeSpread());
    }
}
```

Writing Non-Blocking version for same using Atomic package

This version does not require the calling thread to obtain lock on the Object, thus it could be slightly faster.

```
public class TimeSpread {
    final AtomicBoolean started = new AtomicBoolean(false);
    final AtomicInteger stopCounter;
    long startTime;
    long stopTime;
    public TimeSpread(int threads) {
        stopCounter = new AtomicInteger(threads);
    }
    public void start() {
        if (!started.get()) {
            if (started.compareAndSet(false, true)) {
                startTime = System.currentTimeMillis();
            }
        }
    }
    public void stop() {
        if(stopCounter.getAndDecrement()==1){
            stopTime = System.currentTimeMillis();
        }
    }
    public long timeConsumed(){
        return stopTime-startTime;
    }
    public static void main(String[] args) throws InterruptedException {
        int threads = 300;
        final TimeSpread timeSpread = new TimeSpread(threads);
        Runnable t = new Runnable(){
            public void run() {
                timeSpread.start();
                try {TimeUnit.SECONDS.sleep(5);} catch (InterruptedException e) {}
                timeSpread.stop();
            }
        };
        List<Thread> list= new ArrayList<>(threads);
        for(int i=0;i< threads;i++){
            Thread thread = new Thread(t);
            thread.start();
            list.add(thread);
        }
        for (Thread thread : list) {
            thread.join();
        }
        System.out.println("time spread = " + timeSpread.timeConsumed());
    }
```

Using existing Synchronizer - CountDownLatch for Calculating the time window

We can use two CountDownLatch (with permit 1 and 10) and countdown the first latch at the first line of run method, second latch just before the exit of run method.

Here is the sample Java code illustrating the same,[1]

```
public static long time(Executor executor, int concurrency, final Runnable action) throws InterruptedException {
    final CountDownLatch ready = new CountDownLatch(concurrency);
    final CountDownLatch start = new CountDownLatch(1);
    final CountDownLatch done = new CountDownLatch(concurrency);
    for (int i = 0; i < concurrency; i++) {
        executor.execute(new Runnable() {
            public void run() {
                ready.countDown(); // Tell timer we're ready
                try {
                    start.await(); // Wait till peers are ready
                    action.run();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                } finally {
                    done.countDown();  // Tell timer we're done
                }
            }
        });
    }
    ready.await();    // Wait for all workers to be ready
    long startNanos = System.nanoTime();
    start.countDown(); // And they're off!
    done.await();     // Wait for all workers to finish
    return System.nanoTime() - startNanos;
}
```

Notes

We can evaluate any of these three approaches for our requirement and pick one. But definitely, using CountDownLatch seems a cleaner approach where the latch hides the boiler-plate code of the synchronization.

1 Effective Java 2nd Edition : Item 69

Question : There is a stream of words which contains Anagrams. How would you print anagrams in a single bucket from that stream ?

SOLUTION

Sort each word and then see if the two words are equal ? abba, baab, abab should go to the same bucket.

Simple method to check if two Strings are anagrams

```
public boolean isAnagram(String s1, String s2){
    char[] a1 = s1.toCharArray();
    char[] a2 = s2.toCharArray();

    Arrays.sort(a1);
    Arrays.sort(a2);

    if (Arrays.toString(a1).equals(Arrays.toString(a2))){
        return true;
    }
    return false;
}
```

Algorithm

1) Use a hashmap with string as key and list<string> as value where list of strings contain all anagrams of a given key string.
2) For each word in the input array, create a key by sorting the word and put this word to that list whose key is the sorted word. for example [aakk -> akka, akak] If it does not exist then create a new list with the sorted word as key in map.
3) Print all strings from the list whose key is the input word(sorted string).

Source Code

```
import java.util.*;

public class Anagrams {
   private static Map<String, List<String>> anagramsMap = new HashMap<>(100);

   public static void main(String[] args) {
      String [] input = {"akka","akak","baab","baba","bbaa"};
      for (String s : input) {
            char[] word = s.toCharArray();
         Arrays.sort(word);
         String key = String.valueOf(word);
         if(!anagramsMap.containsKey(key)){
            anagramsMap.put(key, new ArrayList<String>());
         }
         anagramsMap.get(key).add(s);
      }
      System.out.println("anagramsMap = " + anagramsMap);
   }
}
```

Time Complexity

If we ignore the time consumed by sorting an individual string then we can say that the above approach takes Big O(n) time complexity. Otherwise the actual time complexity would be N log N (sorting) + N (compare)

Question : What are fail-fast Iterator ? what is fail safe ?

SOLUTION

fail-fast Iterator

Iterator fails as soon as it realizes that the structure of the underlying data structure has been modified since the iteration has begun.
Structural changes means adding, removing any element from the collection, merely updating some value in the data structure does not count for the structural modifications. It is implemented by keeping a modification count and if iterating thread realizes the changes in modification count, it throws ConcurrentModificationException.

fail-safe Iterator

This iterator does not throw any exception if collection is modified structurally during the iteration. Such Iterator may work on clone of collection instead of original collection - such as in CopyOnWriteArrayList
While ConcurrentHashMap's iterator returns the state of the hashtable at some point at or since the creation of iterator.

Question : Tell something about CopyOnWriteArrayList ? Where is it used ?

SOLUTION

Functionality wise this collection is very much similar to ArrayList except the fact that CopyOnWriteArrayList is thread-safe. It maintains thread-safety using Immutability approach, where any modification operations results in creating a fresh copy of the underlying array.

This is ordinarily too costly, but may be more efficient than alternatives when traversal operations vastly outnumber mutations, and is useful when you cannot or don't want to synchronize traversals, yet need to preclude interference among concurrent threads.

The "snapshot" style iterator method uses a reference to the state of the array at the point that the iterator was created. This array never changes during the lifetime of the iterator, so interference is impossible and the iterator is guaranteed not to throw ConcurrentModificationException. The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove, set, and add) are not supported.

These methods throw UnsupportedOperationException.

Question : Is it safe to iterate over an ArrayList and remove its elements at the same time ? When do we get ConcurrentModificationException & hidden Iterator ?

SOLUTION

Iterator returned by the ArrayList (and many other collection types) is fail-fast. If the list is structurally modified at anytime after the iterator is created, in any way except through the Iterator's own remove() method, the iterator will throw ConcurrentModificationException and thus fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Structural Modification

"A structural modification is any operation that adds or deletes one or more elements, or explicitly resizes the backing array; merely setting the value of an element is not a structural modification." - Java Docs

Further, the structural modification could happen either from single thread or from multiple threads. The behavior of ArrayList would be different in both the cases as mentioned below.

Single Threaded Scenario

Never call list.remove(element) to remove a item from list while traversing it. Rather use Iterator.remove() mehtod.

```
private List<String> list = new ArrayList<>(asList("first", "second", "third", "fourth"));

    public void unsafeMethod() {
        for (String item : list) {              // Will throw ConcurrentModificationException
            list.remove(item);
        }
    }

    public void safeMethod() {
        Iterator<String> iterator = list.iterator();
        while (iterator.hasNext()) {           // safe to call iterator.remove()
            String item = iterator.next();
            iterator.remove();
         }
    }
```

Multi-Threading Scenario

ArrayList implementation is not thread-safe because it provides no synchronization mechanism for protecting the shared state of its fields. If multiple threads access an ArrayList instance concurrently, and at least one of the threads modifies the list structurally, it **must** be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the list.

If no such object exists, the list should be "wrapped" using the Collections.synchronizedList() method. This is best done at creation time, to prevent accidental unsynchronized access to the list :

```
List list = Collections.synchronizedList(new ArrayList(...));
```

Hidden Iterators

There are certain ArrayList methods which uses Iterators in a hidden form the API user. size() and toString() are few of them. So care must be taken to call these methods from synchronized block in case of multi-threaded scenario.

Question : What is ThreadLocal class, how does it help writing multi-threading code ? any usage with example ?

SOLUTION

ThreadLocal class provides a simple mechanism for thread safety by creating only one object instance per thread. These variables differ from their normal counterparts in that each thread that accesses one (via its get or set method) has its own, independently initialized copy of the variable. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread (e.g., a user ID or Transaction ID).

Each thread holds an implicit reference to its copy of a thread-local variable as long as the thread is alive and the ThreadLocal instance is accessible; after a thread goes away, all of its copies of thread-local instances are subject to garbage collection (unless other references to these copies exist)

ThreadLocal is used to achieve thread-safety in our multi-threaded code by creating a copy local to a thread and thus no more sharing of state.

When get() method is invoked the first time, ThreadLocal calls initialValue() and returns the newly created Object, the same Object is returned on subsequent invocations by the same thread untill we clear the Object.

ThreadLocal Usage Scenarios

- EntityManager in JPA - EntityManager instance is not thread safe but creating too many EntityManagers could be expensive, thus in a servlet environment ThreadLocal copy could be created using a servlet Filter and re-used the same EntityManager for the whole request life cycle, committing the changes in the end. In this case, when the thread calls get() for the first time, a new EntityManager instance is created and the same is re used on subsequent calls to get() method by the same thread.

```
public class ThreadLocalSession implements SessionCache{
    @Override
    public EntityManager getUnderlyingEntityManager() {
        return threadLocalJPASession.get();
    }
    private static class ThreadLocalJPASession extends ThreadLocal<EntityManager>{
        @Override
        protected EntityManager initialValue() {
            return JPASessionFactory.getInstance().getSession();
        }
    }
    public static final ThreadLocalJPASession threadLocalJPASession = new ThreadLocalJPASession();

    public void set(EntityManager em){
        threadLocalJPASession.set(em);
    }
    public void clear() {
        threadLocalJPASession.get().close();
        threadLocalJPASession.remove();
    }
}
```

- Using Calendar class in multi-threading environment : Calendar.getInstance() is not safe from multi-threading perspective and a copy of it could be created per thread and stored in ThreadLocal.
- Random Number Generator, ByteBuffers, XML parsers can utilize ThreadLocal for optimization purpose.

Three main Criteria for choosing ThreadLocal's applicability

- Object is non-trivial to construct
- Instance of object is frequently needed
- Application is multi-threaded

Notes

ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread (e.g., a user ID or Transaction ID).

Each thread holds an implicit reference to its copy of a thread-local variable as long as the thread is alive and the ThreadLocal instance is accessible; after a thread goes away, all of its copies of thread-local instances are subject to garbage collection (unless other references to these copies exist).

Question : There are M number of Threads who work on N number of shared synchronized resources. How would you make sure that deadlock does not happen ?

SOLUTION

If a single thread uses more than one protected shared resource, then we should make sure that we acquire shared resources in particular order and release them in **reverse** order, otherwise we might end up into a deadlock scenario.

Question : Are there concurrent version for TreeMap and TreeSet in Java Collections Framework ?

SOLUTION

Java Collection Framework have ConcurrentSkipListMap and ConcurrentSkipListSet which are concurrent replacement for TreeMap and TreeSet. These classes implement SortedMap and SortedSet interface respectively. So if our application demands fair concurrency then instead of wrapping TreeMap and TreeSet inside synchronized collection, we can prefer these concurrent utilities. These also implement NavigableMap and NavigableSet interface with methods like lowerKey, floorKey, ceilingKey, higherKey, headMap and tailMap.

Question : You are writing a server application which converts microsoft word documents into pdf format. Under the hood you are launching a binary executable which does the actual conversion of document. How would you restrict the parallel launch of such binaries to 5 in Java, so as to limit the total load on the server.

SOLUTION

This is a typical problem of controlling the parallel access to the shared scarce resource so as to avoid the thread starvation.

JDK 1.5 provides a class specifically designed to address this kind of problem - Semaphore

Semaphore

Counting semaphores are used to control the number of activities that can access a certain resource or perform a given action at the same time, and it could be used for

- Resource pooling for e.g. Database connection pooling.
- To turn any collection into a blocking bounded collection.

Java Source

```
public class PDFConverter {
    private final Semaphore semaphore;

    public PDFConverter(int concurrencyLevel) {
        this.semaphore = new Semaphore(concurrencyLevel);
    }

    public void convertToPdf(File input, File output) throws InterruptedException {
        try{
            semaphore.acquire();            <== Excess threads have to wait here till a permit becomes available
            // Convert the input file to PDF and then write it to the output file.
        }
        finally {
            semaphore.release();
        }
    }
}
```

Notes

Before Java 1.5, we had to write the semaphore functionality from scratch using synchronization (along with wait and notify for inter thread communication)

Semaphores can be used to convert a standard Java Collection into Bounded Collection after which the collection would hold only certain number of elements. Once the allowed elements are In, the thread has to wait till some other thread removes from that collection.

BoundedHashSet Example

```
package org.shunya.power.interview;

import java.util.Collections;
import java.util.HashSet;
import java.util.Set;
import java.util.concurrent.Semaphore;

public class BoundedHashSet<T> {
    private final Set<T> set;
    private final Semaphore sem;

    public BoundedHashSet(int bound) {
        this.set = Collections.synchronizedSet(new HashSet<T>());
        sem = new Semaphore(bound);
    }

    public boolean add(T o) throws InterruptedException {
        sem.acquire();
        boolean wasAdded = false;
        try {
            wasAdded = set.add(o);
            return wasAdded;
        } finally {
            if (!wasAdded)
                sem.release();
        }
    }

    public boolean remove(Object o) {
        boolean wasRemoved = set.remove(o);
        if (wasRemoved)
            sem.release();
        return wasRemoved;
    }
}
```

Chapter 4

Algorithms & Data Structures

Question : Given a collection of 1 million integers ranging from 1 to 9, how would you sort them in Big O(1) time ?

SOLUTION

This is a typical Integer Sorting problem with a constraint that the number range to sort is very limited in spite 1 million total entries. Integer Sorting with limited range is achieved efficiently with Bucket Sorting.

TIP What does Wiki Says about Sorting ?
Bucket sort, counting sort, radix sort, and van Emde Boas tree sorting all work best when the key size is small; for large enough keys, they become slower than comparison sorting algorithms...
Integer Sorting Techniques : http://en.wikipedia.org/wiki/Integer_sorting#Algorithms_for_few_items
Sorting Algorithms : http://en.wikipedia.org/wiki/Sorting_algorithm

Algorithm

Create a array of size 9 and at each index store the occurrence count of the respective integers. Doing this will achieve this sorting with time complexity of Big O(1) and even the memory requirements are minimized. In Order to print the output just traverse the above created array.

Source Class

```
public class BucketSort {
    public int[] sort(int[] array, int min, int max) {
        int range = max - min +1;
        int[] result = new int[range];
        for (int i : array) {
            result[i]++;
        }
        return result;
    }
}
```

Test Class

```
public class BucketSortTest {
    @Test
    public void testBucketSortFor1To9() {
        int[] array = {2, 1, 5, 1, 2, 3, 4, 3, 5, 6, 7, 8, 5, 6, 7, 0};
        int[] sort = new BucketSort().sort(array, 0, 8);
        for (int i = 0; i < sort.length; i++) {
            for(int j=0;j<sort[i];j++){
                System.out.println(i);
            }
        }
    }
}
```

Program output : 0,1,1,2,2,3,3,4,5,5,5,6,6,7,7,8

Notes

Bloom Filter[1] could help us achieve something similar.

1 http://en.wikipedia.org/wiki/Bloom%5Ffilter

Question : Given 1 million trades objects, you need to write a method that searches if the specified trade is contained in the collection or not. Which collection would you choose for storing these 1 million trades and why ?

SOLUTION

HashSet is a good choice for storing this collection because it will offer Big O(1) time complexity. In order to use HashSet we must override equals() and hashcode() method for the Trade Object. If that's not possible then we should created a Trade Wrapper class which overrides these methods.

```
public class Trade{
...
@Override
public boolean equals(Object o) {...}
@Override
public int hashCode() {...}
}
```

Question : I have an Integer array where every number appears even number of time except one. Find that number.

SOLUTION

Approach

This problem can be solved by utilizing bitwise operators in O(1) space and O(n) time complexity.
XOR all the number together and the final result would the odd number.

How does XOR works ?

Here is the complete solution using XORing

```
public class OddNumberProblem {
    private int[] array = {1,1,2,3,4,5,2,3,4};
    public int findSingleOdd(){
        int result =0;
        for (int i : array) {
          result=result^i;
        }
        return result;
    }

    public static void main(String[] args) {
        OddNumberProblem test = new OddNumberProblem();
        int singleOdd = test.findSingleOdd();
        System.out.println("singleOdd = " + singleOdd);
    }
}
```

Output:

```
singleOdd = 5
```

Question : how would you check if a number is even or odd using bitwise operator ?

SOLUTION

Least significant bit (rightmost) can be used to check if the number is even or odd.
For all Odd numbers, rightmost bit is always 1 in binary representation.

```
public static boolean checkOdd(long number){
    return ((number & 0x1) == 1);
}
```

Notes

We prefer bitwise operator for checking even odd because the traditional way of checking even by n % 2 ==0 is compassionately expensive compared to bitwise & operator (Big O(1) time complexity)

Question : How would you check if the given number is power of 2 ?

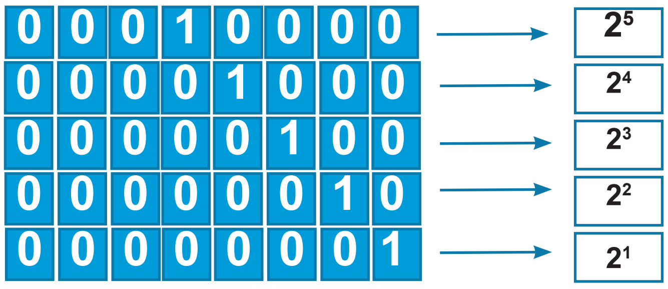
SOLUTION

This can be easily checked using bitwise operators in Java. Firstly let's see how a number looks in binary when it is power of two.

From the figure, we can see that only 1 bit is set for all numbers which are exponent of 2.

Let's now write a method to check if only nth bit of a number is set to true.

```
int isPowerOfTwo (int x)
{
  return ((x != 0) && !(x & (x - 1)));
}
```



Binary presentation of 2's exponent

The first condition (x!=0) checks if the number is positive, because the second conition works only for positive numbers.

The second condition (x & (x-1)) will return zero for a number which is exponent of two (provided the number is positive). For example, in case of number 32,

```
          00010000 (2⁵)
&         00001111  (2⁵-1)
---------------------------------
          00000000 (0)
```

Thus through the above code, we are checking if the number is positive and is power of two.

Question : What is a PriorityQueue ? How is it implemented in Java ? What are its usages ?

SOLUTION

What is a PriorityQueue ?

It is an abstract data type similar to queue but the elements are stored in a sorted order according to some priority associated with them, and the element with the higher priority is served before the element with lower priority. Priority is decided using the Comparator provided at the time of its construction. If no comparator is provided, then the natural ordering of elements is used to prioritize them.

For example, if all elements are of type Integer and no comparator is provided, then the natural order is used resulting in highest priority to the smallest Integer value.

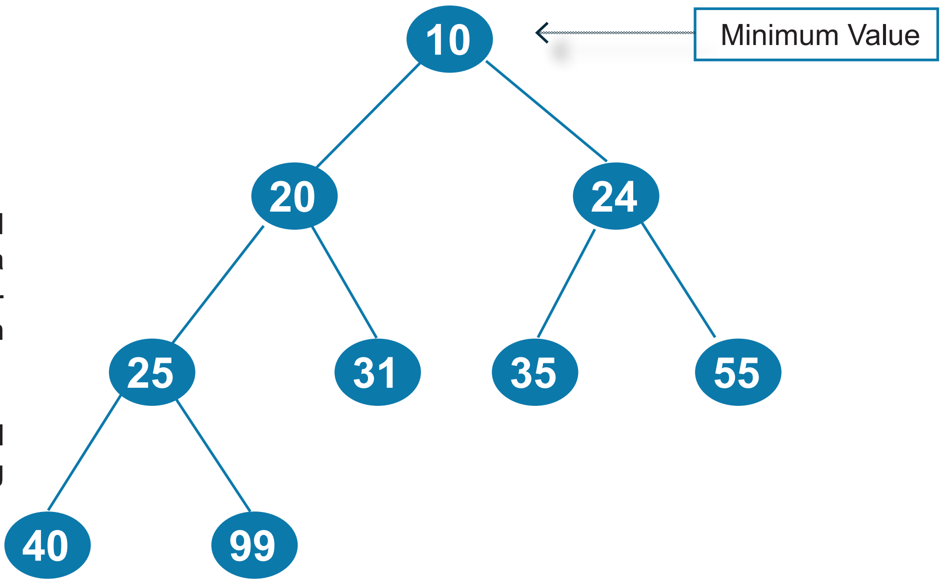


Figure : **Binary Min-Heap (minimum on top)**

Implementation - Binary Heap

Binary Heap[1] data structure is used as the underlying for implementing a PriorityQueue in Java. A Binary Min-Heap is a complete binary tree such that -

- Each node is less than or equal to each of its children according to the comparison predicate (Comparator) provided at the time of construction.
- The tree is a complete binary tree - all levels of the tree are full except, at the level farthest from root. And if the last level of the tree is not complete, then the nodes of that level are filled starting from left.

A complete binary trees can be implemented with an random access array, which makes the implementation fast for retrieval.

Given the index of an element, element's children can be accessed in constant time using an random access array. Children of the element at index i are at indexes (i << 1)+1 and (i << 1)+2. And the parent of an element at index j is at (j-1) >>1

How is it different from Binary Search Tree ?

Please note that a Binary heap is not a binary search tree.
- The ordering in binary heap is top to bottom compared to left to right in case of binary search tree.
- Duplicate elements are allowed in Binary Heap which is not the case with Binary Search Tree (a duplicate key are overwritten by the new key).
- A binary heap is a complete binary tree which may not be true for a Binary Search Tree.

1 http://en.wikipedia.org/wiki/Binary_heap

Time Complexity for PriorityQueue

- Big O(1) time for retrieval methods - peek(), element() and size().
- Big O(log n) time for enqueing and dequeing method - offer, poll, remove() and add.
- Big O(n) time for remove(Object) and contains(Object) methods.

Usage

- A network printer where multiple people submit print jobs at the same time, While one big print job is executing, PriorityQueue could re-arrange other jobs so that the small print jobs (with very less number of pages) execute on priority compared to big ones.
- Emergency department in any hospital handles patients based on their severity, thus priority queue could used to implement such logic.

Notes

Binary Heap can be used for solving algorithmic problems, like the following -

- Finding top 10 most frequently used words in a very large file in O(n)
- Finding top 1 million numbers from a given large file containing 5 billion numbers in O(n)
- You have a file with 1 trillion numbers, but memory can hold only 1 million, How would you find the lowest 1 million out of all these numbers ?
  Hint - Create a binary-max-heap with 1 million capacity, so the largest number will be at the top. Now go over each number in the file, compare it with the peek(), if it is smaller then poll() the element and add the new one. The total complexity should be less than O (n log n). Selection Rank algorithm could also be used to solve this problem, provided there exists no duplicate number.

Question : What is a Binary Search Tree ? Does Java provide implementation for BST ? How do you do in-order, pre-order and post-order Traversal of its elements ?

SOLUTION

A Binary Search Tree[1]

(also known as sorted binary tree) is a node based binary tree data structure which has the following properties,

- All elements in the left subtree are less than the root element.
- All elements in the right subtree are greater than the root element.
- Both, left and right subtree must also be binary search trees.
- There can not be any duplicate element in the entire tree.

TreeMap in Java 6

Java provides its Binary Search Tree implementation in TreeMap class. TreeMap is special kind of BST which is height balanced and known as red-black-tree.

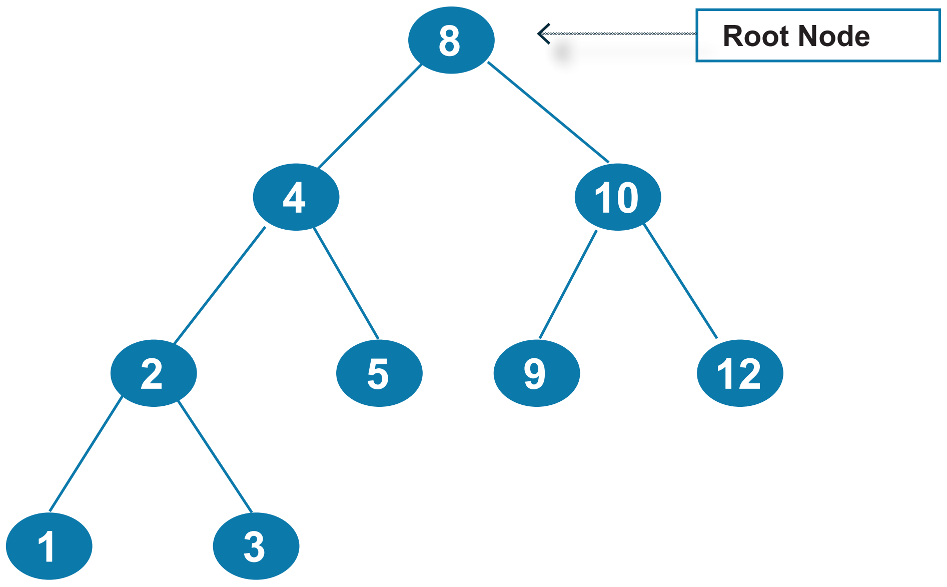


Figure : **Binary Search Tree Example**

Tree Traversal

There are three types of depth-first traversal, namely pre-order, in-order and post-order.

Pre-order Traversal

- Visit the root node
- Traverse the left subtree
- Traverse the right subtree

In-order Traversal

- Traverse the left subtree
- Visit the root node
- Traverse the right subtree

Post-order Traversal

- Traverse the left subtree
- Traverse the right subtree
- Visit the node

Pseudo Code for Traversal

Given a Node with definition, `Node{data, Node left, Node right}`

Pre-order

```
   preOrder(node){
         if(node == null)
             return
         visit(node)
         preOrder(node.left)
         preOrder(node.right)
   }
```

1 http://en.wikipedia.org/wiki/Binary_search_tree

In-order

```
inOrder(node){
        if(node==null)
           return;
        inOrder(node.left)
        visit(node)
        inOrder(node.right)
}
```

Post-order

```
postOrder(node){
        if(node==null)
          return;
        postOrder(node.left)
        postOrder(node.right)
        visit(node)
}
```

Question : check if a binary tree is a Binary Search Tree or not ?

SOLUTION

Source Code

```
public boolean isValid(Node root) {
     return isValidBST(root, Integer.MIN_VALUE,
          Integer.MAX_VALUE);
}

private boolean isValidBST(Node node, int MIN, int MAX) {
     if(node == null)
         return true;
     if(node.value > MIN
         && node.value < MAX
         && isValidBST(node.left, MIN, node.value)
         && isValidBST(node.right, node.value, MAX))
         return true;
     else
         return false;
}
```

The recursive call makes sure that subtree nodes are within the range of its ancestors. The time complexity will be O(n) since every node will be examined once.

Reference

http://cslibrary.stanford.edu/110/BinaryTrees.html#java

Question : How would you convert a sorted integer array to height balanced Binary Search Tree ?

Input: Array {1, 2, 3}
Output: A Balanced BST

```
    2
  /  \
 1    3
```

SOLUTION

Algorithm

1. Get the middle element of the sorted array (start+(end-start)/2) and put this element at the root
2. Recursively repeat the same for the for the left and right child
   i. Get the middle of the left half and make it left child of the root created in step 1.
   ii. Get the middle of right half and make it right child of the root created in step 1.

Time Complexity : Big O(n)

Java Source

```java
public class SortedArrayToBST {
    static class Node {
        Node left;
        Node right;
        int data;
    }

    Node sortedArrayToBST(int arr[], int n) {
        return sortedArrayToBST(arr, 0, n - 1);
    }
    Node sortedArrayToBST(int arr[], int start, int end) {
        if (start > end) return null;
        // same as (start+end)/2, but it avoids overflow.
        int mid = start + (end - start) / 2;
        Node node = new Node();
        node.data = arr[mid];
        node.left = sortedArrayToBST(arr, start, mid - 1);
        node.right = sortedArrayToBST(arr, mid + 1, end);
        return node;
    }

    public static void main(String[] args) {
        int[] sortedArray = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
        SortedArrayToBST test = new SortedArrayToBST();
        Node node = test.sortedArrayToBST(sortedArray, 11);
        printTree(node);
    }
}
```

References

http://cslibrary.stanford.edu/110/BinaryTrees.html#java
http://www.geeksforgeeks.org/sorted-array-to-balanced-bst/
http://leetcode.com/2010/11/convert-sorted-array-into-balanced.html

Question : How would you calculate depth of a binary tree ?

SOLUTION

Height of a binary tree can be easily calculated using the following recursive algorithm.

Algorithm

1. Base condition - if nod is null then return 0.
2. Calculate height of left sub tree, height of right sub tree, and then return the height of the tree as the max of two heights+1.

```
static class Node {
        Node left;
        Node right;
        int data;
    }

public static int height(Node node){
        if(node == null){
            return 0;
        }
        int hLeft = height(node.left);
        int hRight = height(node.right);
        int height = 1+Math.max(hLeft, hRight);
        return height;
    }
```

Question : Calculate factorial using recursive method.

SOLUTION

Recursive Algorithm

1. Base Condition - factorial of 1 is equal to 1.
2. Recursive Condition - factorial of n = n*factorial(n-1).

```
public int factorial(int n){
        if(n==1){
        return n;
        }
        else{
        return n*factorial(n-1);
        }
}
```

Question : You have a mixed pile of N nuts and N bolts and need to quickly find the corresponding pairs of nuts and bolts. Each nut matches exactly one bolt, and each bolt matches exactly one nut. By fitting a nut and bolt together, you can see which is bigger. But it is not possible to directly compare two nuts or two bolts. Given an efficient method for solving the problem.

SOLUTION

This can be solved quickly using a customized Quick Sort algorithm.

A simple modification of Quicksort shows that there are randomized algorithms whose expected number of comparisons (and running time) are O(n log n).

Approach

Pick a random bolt and compare it to all the nuts, find its matching nut and compare it to all the bolts (they match, the nut is larger, the nut is small). This will divide the problem into two problems, one consisting of nuts and bolts smaller than the matched pair and the other consisting of larger pairs. Repeat and divide this until all the nuts and bolts are matched. This is very similar to quick sort in action and achieving the result in O(n log n) time.

References

http://www.wisdom.weizmann.ac.il/~naor/PUZZLES/nuts_solution.html
http://algs4.cs.princeton.edu/23quicksort/

Question : Swap two numbers using XOR ?

SOLUTION

swapping int a and b without any temporary variable can be done using XOR operator, as shown below.

```
a ^= b;
b ^= a;
a ^= b;
```

This will result in the swap in values of a and b. Please note that there is no fear of value overflow in this case.

Question : Reverse the bits of a number and check if the number is palindrome or not ?

SOLUTION

Answer : int numBitsReversed = Integer.reverse(num);

then XOR the number with the reversed number if zero then palindrome

see also Integer.reverseByte(num),

Reversing bits of number in java

```
while(x!=0){
b<<1;
b|=(x&1);
x>>1;
}
```

Question : Your are give a file with millions of numbers in it. Find the 20 biggest numbers ?

SOLUTION

There are multiple ways to solve this problem.

Sorting

Sort all the numbers and pick the first 20. This will take at least O(n log n) time complexity and O(n) memory.

Max Heap (PriorityQueue in Java)

Create a Max heap with size of 20. Now traverse all the elements from the source file using a stream and push it to heap if the minimum element in the heap is less than the current element.
This will take require a constant size of O(k) where k = 20
and O(n log n) for the time complexity.

Selection Rank Algorithm

It is an algorithm to find the ***k***th smallest (or largest) element from a given array in worst case linear time.
This algorithm can be used to find the 20th largest element and then traverse the entire file and compare if the given number is greater than the 20th largest number, if yes then print it.

References

http://en.wikipedia.org/wiki/Selection_algorithm

Question : How would you mirror a Binary Tree ?

SOLUTION

Mirroring will change any given tree into its mirror image. For example,

This tree...

```
      7
     / \
   4   9
  / \
 1  3
```

Will be changed to...

```
      7
     / \
    9   4
       /  \
      3   1
```

Java Source[1]

```
private void mirror(Node node) {
  if (node != null) {
    // do the sub-trees
    mirror(node.left);
    mirror(node.right);

    // swap the left/right pointers
    Node temp = node.left;
    node.left = node.right;
    node.right = temp;
  }
}
```

1 http://cslibrary.stanford.edu/110/BinaryTrees.html#java

Question : How to calculate exponentiation of a number using squaring for performance reason ?

SOLUTION

The following mathematical expression can help us writing a method similar to Math.pow(x, n) using squaring as a technique for achieving exponentiation. This technique greatly reduces the time complexity to O (log n) compared to normal way of multiplying the number n times.

x^n	1 if $n = 0$;
	$1/x^{-n}$ if $n < 0$;
	$x.(x^{n-1/2})^2$, if n is odd
	$(x^{n/2})^2$, if n is even

Simple recursive Algorithm can be written based on above expressions.

```
public class MyMaths {
    public static long expBySquaring(long x, long n) {
        if (n == 1) {
            return x;
        } else if (n % 2 == 0) { // n is even
          return expBySquaring(x*x, n/2);
        }
        else { // n is odd
          return x*expBySquaring(x*x, (n-1)/2);
        }
    }

    public static void main(String[] args) {
        System.out.println(MyMaths.expBySquaring(10, 3));
        System.out.println(MyMaths.expBySquaring(2, 8));

    }
}
```

Result

```
1000
256
```

Notes

Time Complexity of this algorithm is O (log n) where n is the exponentiation. The number of multiplications reduces to half on each interation.

Another method for calculating the pow

Other way to write the same recursive algorithm,

```
int power(int x, int y){
 if( y == 0)
        return 1;
else if (y%2 == 0)
        return power(x, y/2)*power(x, y/2);
 else
        return x*power(x, y/2)*power(x, y/2);
}

int main(){
  int x = 2;
  int y = 3;
  System.out.println(power(10,3));
}
```

But the time complexity in this case would be O (n) and space complexity O (1)

Question : How would you implement a Queue from scratch in Java ?

SOLUTION

Given a class Node, we can easily construct a type safe (Generic) queue which provides two methods - enqueue and dequeue. Internally this queue keeps two references - first one for the Head of the queue and second one for the Tail of the queue. When we add new item to the queue, its added to the head, and when we remove an element then its removed from the Tail (**F**irst **I**n **F**irst **O**ut).

Below is the implementation of the same.

```
public class QueueUsingNode<T> {
    static class Node<T> {
        final T data;
        Node<T> next;

        Node(T data) {this.data = data;}
    }

    Node<T> first, last;

    void enqueue(T item) {
        if (first == null) {
            last = new Node(item);
            first = last;
        } else {
            last.next = new Node<T>(item);
            last = last.next;
        }
    }

    T dequeue(){
        if(first!=null){
            T item = first.data;
            first = first.next;
            return item;
        }
        return null;
    }

    public static void main(String[] args) {
        QueueUsingNode<Integer> test = new QueueUsingNode<>();
        test.enqueue(100);
        System.out.println("test = " + test.dequeue());
    }
}
```

Question : How would you Implement a Stack using the Queue ?

SOLUTION

Here is the Generic implementation of Stack using linked Nodes. Nodes are linked like in a LinkedList and the push and pop operations happen on the head of the Linked Nodes.

```
public class StackUsingNode<T> {
    static class Node<T> {
        final T data;
        Node<T> next;

        Node(T data) {this.data = data;}
    }

    Node<T> top;

    T pop() {
        if (top != null) {
            T item = top.data;
            top = top.next;
            return item;
        }
        return null;
    }

    void push(T item) {
        Node<T> t = new Node<T>(item);
        t.next = top;
        top = t;
    }

    T peek() {
        return top.data;
    }

    public static void main(String[] args) {
        StackUsingNode<Integer> stack = new StackUsingNode<>();
        stack.push(100);
        stack.push(200);
        System.out.println("stack = " + stack.pop());
    }
}
```

Question : How would you implement a simple Math.random() method for a given range say (1-16) ?

SOLUTION

Generating random numbers is not that easy because there are lots of expectations from a perfect random number generator (fair distribution, randomness, fast, etc). The scope of this question is just to write a simple function without worrying about the fairness, speed.
In order to generate a random number, we would require a seed which provides us with the randomness. System.currentTimeMillis() could be a good substitute for providing seed value in our case.

```
public class RandomGenerator {
    public long generate(){
        return System.currentTimeMillis() % 16;
    }

    public static void main(String[] args) {
        RandomGenerator test = new RandomGenerator();
        long generate = test.generate();
        System.out.println("generate = " + generate);
    }
}
```

The value returned by the System.currTimeMillis() is very large and we need to make it fit to our bounds using the modulus operator (x % n) which will bound the upper value to be less than n.

Question : How an elevator decides priority of a given request. Suppose you are in an elevator at 5th floor and one person presses 7th floor and then 2nd presses 8th floor. which data structure will be helpful to prioritize the requests ?

SOLUTION

Generally elevator's software maintains two different queues - one for the upward traversal and another for downward traversal along with the direction flag which holds the current direction of movement. At any given point in time, only single queue is active for the serving the requests, though both the queues can enqueue requests.

PriorityQueue is used for storing the user requests where priority is decided based on following algorithm.

For upward movement PriorityQueue
The floor number with lower value has the higher priority.

For downward movement PriorityQueue
The floor number with higher value has the higher priority.

Requests are removed from the PiorityQueue as soon as they are served. If current floor is 5th and user presses 4th floor with upward moving elevator, then the requests are queued to the downward movement priority queue (which is not yet active)

Question : How would you multiply a number with 7 using bitwise hacks ?

SOLUTION

This can be achieved by multiplying the number with 8 and then subtracting the number from the result.

Multiplying a number with 8 using bit shift operators

If we left shift bits of a number by 2^3 then it would be equivalent to multiplying the number by 8.

```
public class MultiplyBy7 {
    public static void main(String[] args) {
        System.out.println(multiplyBy7(8));
    }

    private static int multiplyBy7(int number) {
        // multiply by 8 using bitwise left shift operator (2^3 = 8)
        int result =number << 3;
        result =  result - number;
        return result;
    }
}
```

Question : What is the best way to search an element from a sorted Integer Array ? What would be it's time complexity ?

SOLUTION

Binary search is best when we want to search from within a sorted collection.
It narrows down the search area to half in each iteration and thus very time efficient.

Binary Search Algorithm

```
int low = 0;
int high = list.size()-1;

while (low <= high) {
    int mid = (low + high) >>> 1;
    Comparable<? super T> midVal = list.get(mid);
    int cmp = midVal.compareTo(key);

    if (cmp < 0)
        low = mid + 1;
    else if (cmp > 0)
        high = mid - 1;
    else
        return mid; // key found
}
return -(low + 1);  // key not found
```

Worst case Time Complexity for Binary Search is Big O (log n)

Question : How would you reverse a Singly linked List ?

SOLUTION

Using Stack to reverse the Linked List[1]

Push all the items to the stack and then re-construct the Linked List by popping the elements from the stack.

Using Recursion to Reverse the Linked List[2]

Breaking down problem statement into recursive method call

1. Base case : if the list is empty or of size one then return list
2. Recursive condition : If the list has n elements, then divide list into two parts - 1st element and rest of the elements. Reverse the position of these two and return.

```
public static Node reverse(Node node) {
        Node firstNode = node;
        if (firstNode.next == null || firstNode == null) {  // Base condition
            return firstNode;
        } else {                                         // Recursive condition
            Node secondNode = firstNode.next;
            firstNode.next = null;
            Node reverseNode = reverse(secondNode);
            secondNode.next = firstNode;
            return reverseNode;
        }
    }
```

Using Iterative approach to reverse a Linked List

Set current Node to the first node of the List
Set its previous node to null
Set next node to the 2nd node
Repeat the process for rest of the List Items.

```
public static Node reverseIterative(Node node) {
        Node prevNode = null;
        Node currNode = node;
        Node nextNode;
        while (currNode != null) {
            nextNode = currNode.next;
            currNode.next = prevNode;
            prevNode = currNode;
            currNode = nextNode;
        }
        return prevNode;
    }
```

Web links for the same

http://goodinterviewquestions.blogspot.in/2012/03/reverse-singly-linked-list.html
http://www.technicalypto.com/2010/01/java-program-to-reverse-singly-linked.html
http://stackoverflow.com/questions/12666178/reverse-singly-linked-list-java-check-whether-circular
http://crackinterviewtoday.wordpress.com/2010/03/24/reverse-a-single-linked-list-iterative-procedure/

1 http://datastructuresblog.wordpress.com/2007/03/30/reversing-a-single-linked-list-using-stack/
2 http://crackinterviewtoday.wordpress.com/2010/03/24/reverse-a-single-linked-list-recursive-procedure/

Question : How would you count word occurrence in a very large file ? How to keep track of top 10 occurring words ?

SOLUTION

There are limited number of natural language words available and all of them can be easily fit into today's computer RAM. For example oxford English dictionary contains total of around 0.6 million words.

Finding the Word Occurrence Count

Stream the words into a HashMap (put operation is Big O(1)) keeping the value as word occurrence count. On every word count, update the word In TopOccurrence so as to maintain top frequently used words.

To Keep Track of Top N occurring Words Using Binary Heap

This can be achieved by maintaining a min heap (using PriorityQueue) of max size N, and then

- Every time a new number arrives, check if the heap size if less than N - then add it. Otherwise
- Check if the peek element is less than the new number, and if it is, then poll the existing number and the new One.
- When we are done traversing the entire word-counts then we will have heap containing the top N frequently occurring words.

Java Source

```
    public class TopOccurrence {
      private final PriorityQueue<Integer> minHeap;
      private final int maxSize;
      public TopOccurrence(int maxSize) {…}
          public void add(int data) {
           if (minHeap.size() < maxSize) {              //  size() is Big O(1)
               minHeap.offer(data);                     // Big O(log(n))
           } else if (minHeap.peek() < data) {          // peek() is Big O(1)
               minHeap.poll();                          // Big O(log(n))
               minHeap.offer(data);                     // Big O(log(n))
           }
       }
       public static void main(String[] args) {
           TopOccurrence test = new TopOccurrence(3);
           test.add(10);  test.add(20); test.add(30); test.add(10);test.add(100);
           test.print();
       }
}
```

Output => 20,30,100

The overall time complexity of the above algorithm should be **O (n log k)** where n is the total number of elements and k is the number of elements that we need. Space complexity would be Big O(k).

Notes

We preferred to choose Binary Heap over TreeSet because TreeSet provide a get method with Big O(log n) time complexity over PriorityQueue's peek() method with Big O(1), so its a big time saver for the given requirement.

Binary Min Heap[1] is a *complete binary tree*[2] data structure in which each Node is less than or equal to each of its children. Heap is very efficient O(1) for finding minima and maxima from a given data set. PriorityQueue in JDK 1.6 is a implementation for *Binary Min Heap*.

Is Heap Abstract Data Type ?

A heap is a specific data structure and PriorityQueue (It's implementation) is the proper term for the abstract data type.
A heap data structure should not be confused with the heap which is a common name for dynamically allocated memory. The term was originally used only for the data structure.

Other Strategies for Scalable Design

If we have very limited memory in our device then we can utilize memory efficient data structures for storing words - TRIE and DAG.

TRIE - memory is shared between multiple words with common prefix and word count can be maintained along with the word termination mark, but it would be more time consuming than the HashMap

http://stackoverflow.com/questions/12190326/parsing-one-terabyte-of-text-and-efficiently-counting-the-number-of-occurrences

Direct Acyclic Graphs[3]

It is a directed graph that contains no directed cycles and hence very less memory consumption.
http://www.dotnetperls.com/directed-acyclic-word-graph

Selection Sort[4] could be used for tracking kth smallest element and then all others can be found by comparing with the kth smallest. This would be memory efficient but not time efficient

1 http://en.wikipedia.org/wiki/Heap_%28data_structure%29
2 http://en.wikipedia.org/wiki/Complete_Binary_Tree
3 http://en.wikipedia.org/wiki/Directed_acyclic_graph
4 http://en.wikipedia.org/wiki/Selection_algorithm

Question : What is difference between synchronized HashMap and a hashtable ?

SOLUTION

Functionally both are same except the single difference that a hash table is one of the legacy collection class which was introduced well before the Java Collection Framework.

Both of these classes implement Map interface and are part of Collections framework as of JDK 2. Hashtable implements one extra interface - Dictionary which Hashmap does not.

HashMap should be preferred if thread-safety is not required, and ConcurrentHashMap should be preferred to Hashtable if highly concurrent implementation is required.

Question : What is difference between Iterator and LisIterator ?

SOLUTION

An Iterator class provides us with 3 methods - next(), hasNext() and remove(). Every Collection in Java is Iterable - posses an iterator to allow traversal and removal its underlying elements.

ListIterator - It is a specialized iterator for lists that allows to traverse the list bidirectionally, modify the list during iteration, and obtain the iterator's current position in the list. It allows complete modification - remove, add and update operations are provided.

Question : What is difference between Collections.sort() and Arrays.sort() ? Which one is better in terms of time efficiency ?

SOLUTION

Collections.sort() internally calls Arrays.sort() and thus the underlying algorithm for both of these methods is same. The only difference is the type of input these methods accept.

Merge Sort algorithm is used by Arrays.sort() method as of JDK 6.

Question : There are 1 billion cell-phone numbers each having 10 digits, all of them stored randomly in a file. How would you check if there exists any duplicate ? Only 10 MB RAM is available to the system.

SOLUTION

Approach 1

Hash all these numbers into 1000 files using hash(num)%1000, then the duplicates should fall into the same file. Each file will contain 1 million numbers roughly, then for each file use HashSet to check for the duplicates.

(If sufficient memory is available)

Approach 2

Use BitSet to represent those 1 billion numbers and then traverse the file and set the appropriate Bit in the BitSet. But check the Bit value before setting it, thus listing the duplicate.

Approach 3

Use bucket Sort to partition the numbers based on some common prefix. Then the duplicate numbers should fall under the same bucket.

Approach 4

Build a TRIE from this huge file (This will load every thing into memory) and then search the number before putting it into TRIE.

References

http://stackoverflow.com/questions/7703049/check-1-billion-cell-phone-numbers-for-duplicates
http://stackoverflow.com/questions/7153659/find-an-integer-not-among-four-billion-given-ones

Chapter 5

Object Oriented Design

Question : How would you design an elevator system for multi story building? Provide with request scheduling algorithm & Class diagram for the design.

SOLUTION

For a single elevator system, normally two different queues are used to store the requests. One queue is used to store upward requests and other queue used to store the downward requests. Queue implementation used is the BlockingPriorityQueue which maintains the priority of its requests based on the floor numbers. For upward motion, the lower floor number has the higher priority and opposite for the downward motion of elevator. A 2 bit flag can be used to store the current direction of the elevator where 00 represents Idle, 01 for upward motion, 11 for the downward motion.

A Bit Vector can be used to map the floor numbers, and if someone presses the floor button then the corresponding bit can be set to true, and a request is pushed to the queue. This will solve the duplicate request problem from outside the elevator at the same floor. As soon as the floor request is served, the corresponding bit is cleared and the request is removed from the queue.

The actual software application for handling elevator requires lot of interaction with the hardware and is out of scope for this book.

References

http://thought-works.blogspot.in/2012/11/object-oriented-design-for-elevator-in.html
http://www.careercup.com/question?id=9842677

Question: Write Object Oriented design for library management system.

SOLUTION

Terminology

Publication - A Publication is a written work meant for distribution to public (it has author, publisher)
Book - A Book is a specific type of publication extending the abstract Publication
Journal - A journal is also a specific type of publication which extends some properties of Publication.
Transaction - Return, Borrow and Renew are three main types of transactions happening inside a library.

Gathering Requirements

1. Authentication for the valid user
2. Managing (add, update, remove) the Inventory of all type of publications (Book, Journal, Magazine, etc)
3. Ability to search for a Publication by various parameters
4. Ability to borrow a Book from library
5. Ability to renew a book
6. Ability to return a book

Design

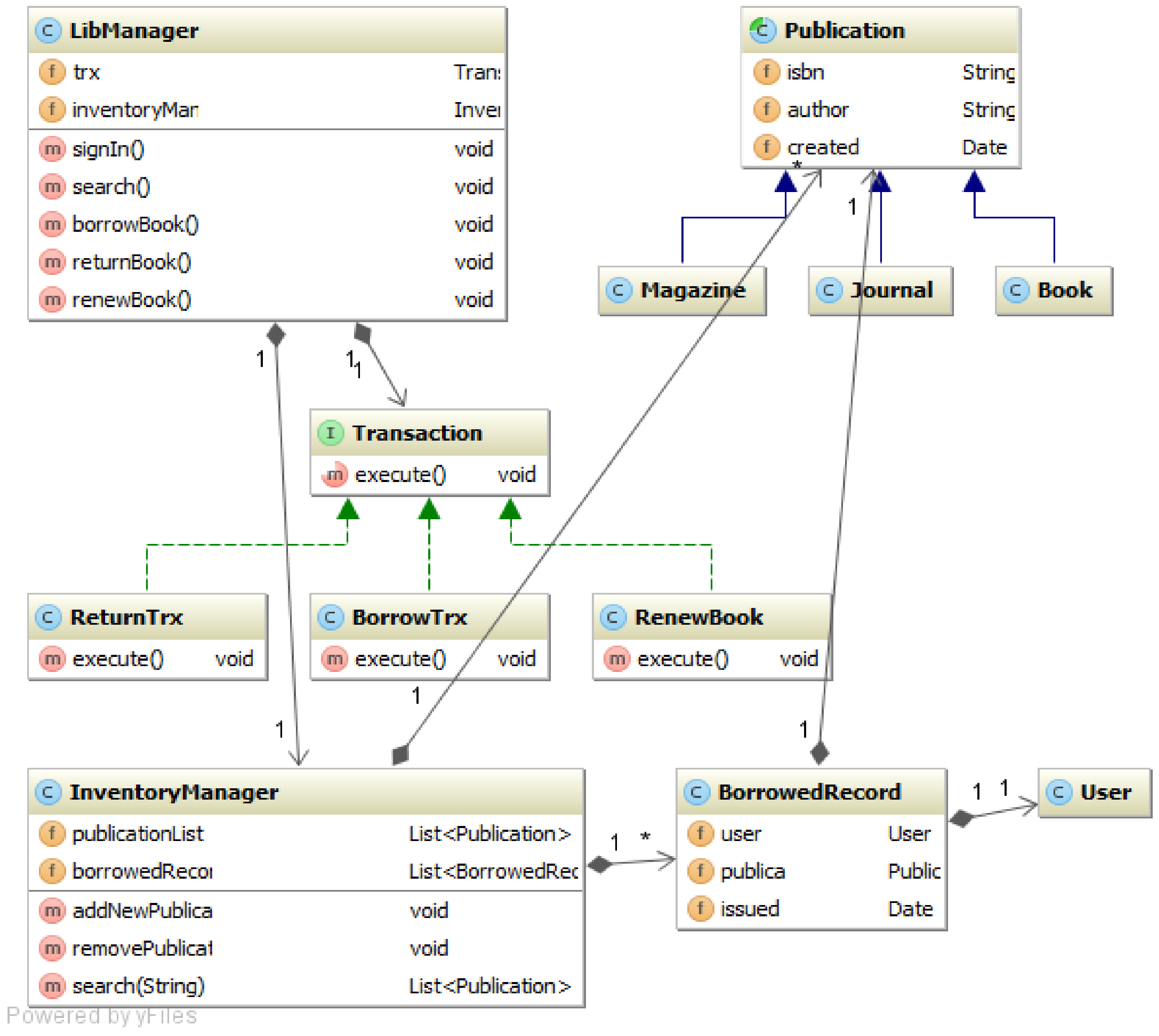
Journal and Book are specific type of Publication so we can map them into Object Oriented World by making Publication Class as Abstract and then Book, Magazine and Journal extending the Publication.

Similarly borrow, return and renew are the type of transaction that a library user will typically be performing. Transaction can be made an interface and Return, Borrow & Renew will implement this interface.

References

Chapter 6. Object oriented design with UML and Java
http://thought-works.blogspot.in/2012/11/object-oriented-design-for-library-to.html

Here is the class diagram for the same.



Question: Design ATM machine.

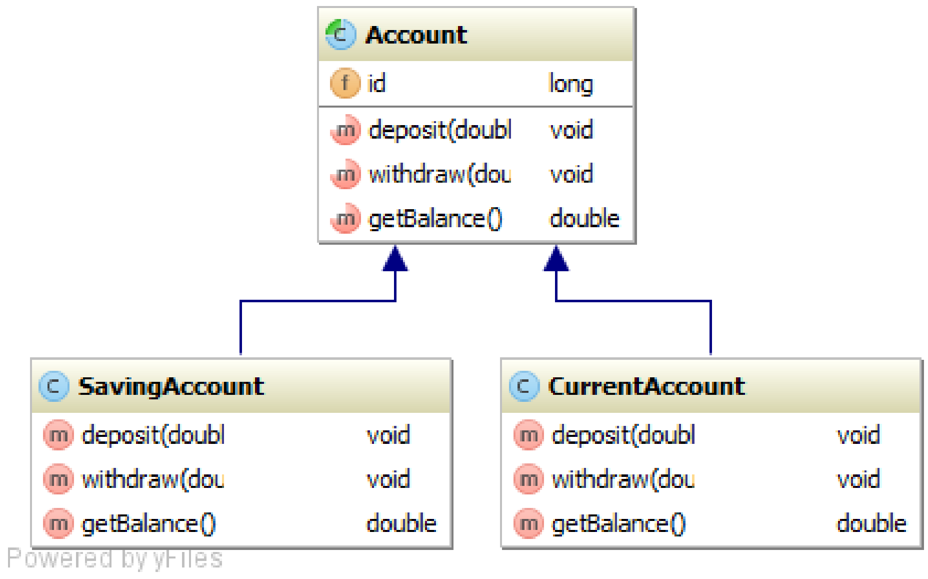
SOLUTION

State Design Pattern along with generalization can be used to solve the problem.

Gathering Requirements
1. Authenticate user with PIN
2. Select account type - Current Account, Saving Account
3. Select Operation : Balance Inquiry, Withdrawal or Deposit
4. Execute any of the above operation after supplying necessary input.
5. Print the transaction if required.

Design
We can utilize state design pattern for 2nd step mentioned above to maintain the state of account selected by the user.

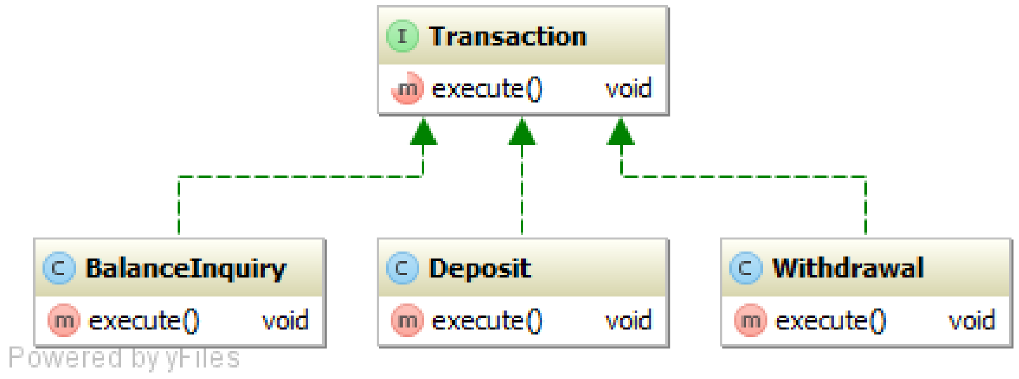
Account - > SavingAccount, CurrentAccount (using state pattern)



Account has two states - Saving Account & Current Account. User will select one of these account for executing a transaction, and the appropriate state will be set at that moment.

3rd step can be solved using polymorphism where Balance Inquiry, Withdrawal & Deposit represents a Transaction which can be executed.

Transaction -> Balance Inquiry, Withdrawal, Deposit (using generalization)



Prompt user if the user requires receipt or not and accordingly execute the action.

References
http://www.careercup.com/question?id=7504685

Question: There is a pricing service which connects to Reuters & Bloomberg and fetches the latest price for the given Instrument Tics. There could be multiple price events for the same Stock and we need to consider the latest one. Design a service to show prices for the Top 10 stocks of the Day ?

SOLUTION

This problems requires us to collect price feeds, remove duplicates based on InstrTic keeping the latest one, and then finally capture the top 10 feeds based on price. Keeping in mind that we need to remove duplicates based on Tic# and then sorting based on price - 2 different fields to act upon.

Lets see how we can solve this problem using Java.

HashSet is a good option for removing duplicates from the collection, so iterate over the entire collection of feeds and then add them to HashSet, rest will be taken care by HashSet. But we need to override equals and hashcode based on Tic# so as to remove the duplicate TIC# entries. So our first requirement of removing duplicates will be done after implementing this.

Now for finding top 10 feeds based on prices, we can use PriorityQueue (min heap) of fixed size 10. While iterating over the HashSet entries we will check if the price of the feed is greater than the peek entry of PriorityQueue, if yes then poll the entry and offer the new one. This way we will get a list of top 10 priced feeds.

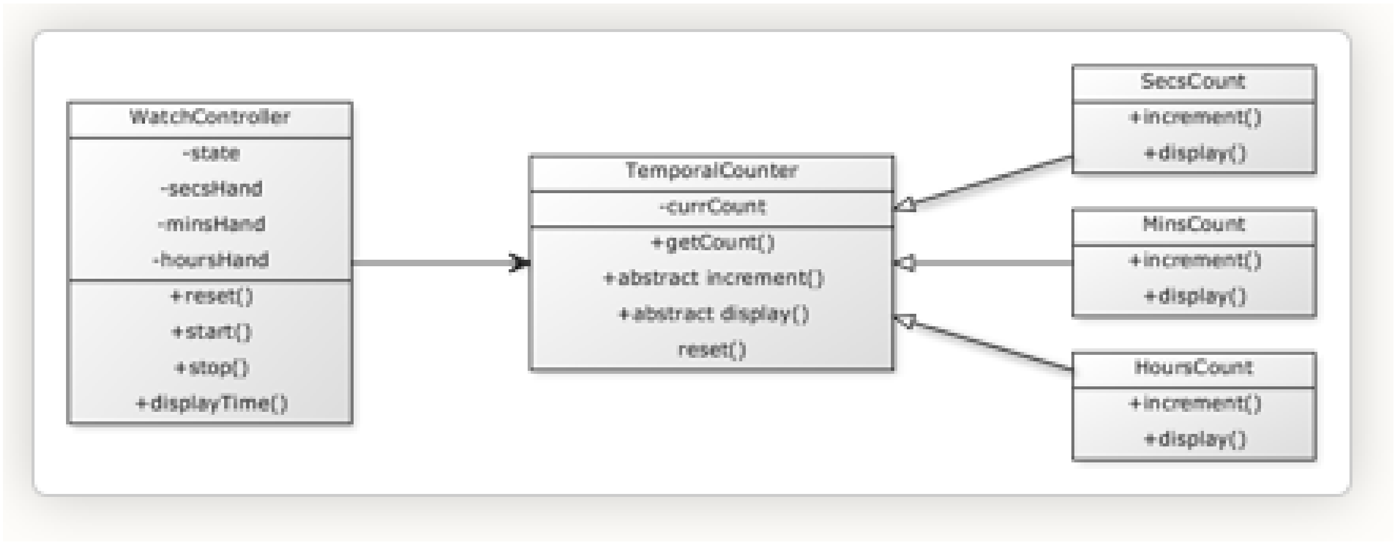
Print device for printing Top N numbers from a collection.

Make the PriorityQueue's size configurable so that it can be adjusted as per the requirement.

Question: How to design a digital watch by using oops??

SOLUTION

Solution :
http://www.careercup.com/question?id=13523672
http://thought-works.blogspot.in/2012/11/object-oriented-design-for-digital.html



Question: Design DVD renting system, database table, class and interface

SOLUTION

Solution :
http://www.careercup.com/question?id=13319670

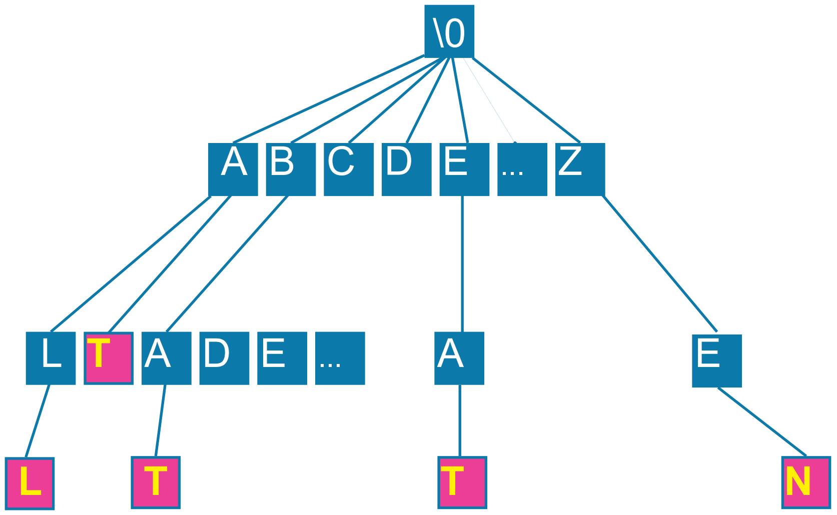
```
RentalImpl{
searchDVD();
rentDVD();
returnDVD();
calculateRent();
};

abstract class DVD{
int charge;
string name;
string id;
}
class movieDVD :DVD
class softwareDVD:DVD
```

Question: Design Phone Book for a mobile using TRIE (or a prefix tree)

SOLUTION

TRIE is an ordered tree data structure which store associative arrays and the keys are usually alphabets. The position in the tree defines the key with which it is associated. The root node is generally empty (\0) and it contains upto 26 children each representing a alphabet character. All descendants of a given node have a common prefix of string associated with that node. Each node contains a flag which tells if the current node is full word or not. In the diagram on right, pink colored boxes shows the full words.

The term trie comes from re**trie**val
TRIE is a generally good data structure for storing dictionary like data. Phone book can be perfectly implemented using TRIE which will save memory as well as time for prefix searching.



Prefix Tree (TRIE) Visualization

A typical Node of a Trie is defined as

```
static class TrieNode {
        char letter;
        TrieNode[] links;
        boolean fullWord;
        TrieNode(char letter) {
            this.letter = letter;
            links = new TrieNode[26];
            this.fullWord = false;
        }
    }
```

Root node is always empty, as shown below

```
static TrieNode createTree() {
        return (new TrieNode('\0'));
    }
```

To insert a word inside Trie, we would this method

```
static void insertWord(TrieNode root, String word) {
        int offset = 97;
        int l = word.length();
        char[] letters = word.toLowerCase().toCharArray();
        TrieNode curNode = root;
        for (int i = 0; i < l; i++) {
            if (curNode.links[letters[i] - offset] == null)
                curNode.links[letters[i] - offset] = new TrieNode(letters[i]);
               curNode = curNode.links[letters[i] - offset];
        }
        curNode.fullWord = true;
    }
```

To find if given word exists in the Tree

```
static boolean find(TrieNode root, String word) {
        char[] letters = word.toCharArray();
        int l = letters.length;
        int offset = 97;
        TrieNode curNode = root;
        int i;
        for (i = 0; i < l; i++) {
            if (curNode == null)
                return false;
            curNode = curNode.links[letters[i] - offset];
        }
        if (i == l && curNode == null)
            return false;
        if (curNode != null && !curNode.fullWord)
            return false;
        return true;
    }
```

And, finally to print the whole Tree

```
static void printTree(TrieNode root, int level, char[] branch) {
        if (root == null)
            return;
        for (int i = 0; i < root.links.length; i++) {
            branch[level] = root.letter;
            printTree(root.links[i], level + 1, branch);
        }
        if (root.fullWord) {
            System.out.println(String.valueOf(branch, 0, level + 1));
        }
    }
```

The main method looks like,

```
public static void main(String[] args) {
        TrieNode tree = createTree();
        String[] words = {"an", "ant", "all", "allot", "alloy", "aloe", "are", "ate", "be", "beat", "beware", "beast", "bed", "bell"}
        for (int i = 0; i < words.length; i++)
            insertWord(tree, words[i]);
}
```

References

http://en.wikipedia.org/wiki/Trie
http://exceptional-code.blogspot.in/2011/07/coding-up-trie-prefix-tree.html
http://amrutbudihal.blogspot.in/2011/07/trie-data-structure-in-java.html#!/2011/07/trie-data-structure-in-java.html
http://jayant7k.blogspot.in/2011/06/data-structures-trie.html
http://www.technicalypto.com/2010/04/trie-data-structure-part-3-inserting.html
http://www.technicalypto.com/2010/04/trie-in-java.html
http://vkroz.wordpress.com/2012/03/23/prefix-table-trie-implementation-in-java/
http://stevedaskam.wordpress.com/2009/05/28/trie-structures/
http://codereview.stackexchange.com/questions/15887/a-trie-implementation-for-left-to-right-wild-card-search
http://stackoverflow.com/questions/2225540/trie-implementation
https://github.com/mikesamuel/html-contextual-autoescaper-java/blob/master/src/main/com/google/autoesc/Trie.java
http://en.literateprograms.org/Suffix_tree_%28Java%29
http://code.google.com/p/google-collections/source/browse/trunk/src/com/google/common/collect/PrefixTrie.java?r=2

Question: Given two log files, each with a billion usernames (each username appended to the log file), find the usernames existing in both documents in the most efficient manner? Use pseudo-code or code. If your code calls pre-existing library functions, create each library function from scratch.

SOLUTION

Solution :
Hashing technique could be used to solve the problem.

```
for 1 file
 read each line,
hash into their abc buckets depending on the start of
 the letter of the word. (26 buckets for A to Z
to form something like a 26 by xxx table)
 then sort each 26 rows in the hash table and delete dups

for 2nd file
 sort and delete dups
 for each line/name, find match in the hashtable.
 if matchfind, output to another file name foundmatch
```

B+ sorting is surely a possible solution we can look for.
http://en.wikipedia.org/wiki/B%2B_tree

http://www.careercup.com/question?id=1938

Question: How would you describe Producer Consumer problem in Java ?

SOLUTION

A producer consumer problem is a typical example of multi-thread synchronization problem. It describes a scenario where some shared resource (like a queue, buffer) is used by two types of threads - Producers & Consumers. Producer populates the task items into the shared queue, and the consumer polls those jobs and execute them. All this happens in parallel. There could be multiple Producer and multiple Consumer, and producer does not even need to know about the consumer. Synchronization is required among the threads so that Producer does not add more items than the size of the queue, and consumer does not take data when there is no item in the shared queue. All these problems are addressed in Producer Consumer Design.

Using Java 1.5 onwards, this problem can be easily solved using BlockingQueue implementation, as demonstrated below.

```
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ProducerConsumerProblem {
    public static Object exit = new Object();
    public static void main(String args[]) {
        BlockingQueue sharedBlockingQueue = new LinkedBlockingQueue();

        Thread producerThread = new Thread(new Producer(sharedBlockingQueue));
        Thread consumerThread = new Thread(new Consumer(sharedBlockingQueue));

        producerThread.start();
        consumerThread.start();
    }
}

class Producer implements Runnable {
    private final BlockingQueue sharedQueue;
    public Producer(BlockingQueue sharedQueue) {
        this.sharedQueue = sharedQueue;
    }
    @Override
    public void run() {
        int i = 0;
        while (i < 10) {
            try {
                System.out.println("Produced: " + i);
                sharedQueue.put(i);
                TimeUnit.MILLISECONDS.sleep(10);
            } catch (InterruptedException ex) {
                Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);
            }
            i++;
        }
        try {
            sharedQueue.put(ProducerConsumerProblem.exit);
        } catch (InterruptedException e) {}
    }
}

class Consumer implements Runnable {
    private final BlockingQueue sharedQueue;
    public Consumer(BlockingQueue sharedQueue) {
        this.sharedQueue = sharedQueue;
    }
    @Override
    public void run() {
        while (true) {
            try {
                Object item = sharedQueue.take();
                System.out.println("Consumed: " + item);
                if(item == ProducerConsumerProblem.exit)
                    break;
            } catch (InterruptedException ex) {
                Logger.getLogger(Consumer.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

References

http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html
http://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html

Question : How would you resolve task's inter dependency, just as in maven/ant.

Let's consider the following task dependencies.

Task	Dependent On
3	1,5
2	5,3
4	3
5	1

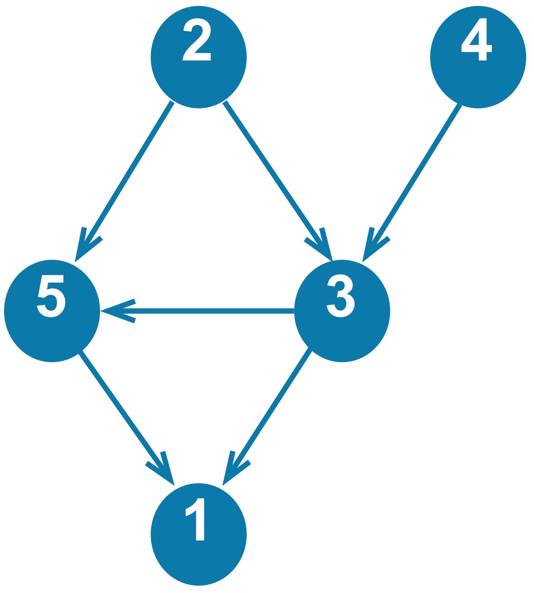
Here first row states that task 3 is dependent on task 1 and task 5, and so on. If the two consecutive tasks have no dependency, then they can be run in any order.

The output should look like - [1, 5, 3, 2 ,4] or [1, 5, 3, 4, 2]

SOLUTION

Approach 1

It is a typical Graph traversal problem, that can be solved using Topological Sorting Algorithm[1] in linear time O(|V| + |E|),
Where
V = number of nodes
E = number of edges



Direct Acyclic Graph for the Task Dependencies

Lets now discuss the algorithm as described by Kahn in 1962.

First, find a list of "start nodes" which have no incoming edges and insert them into a set S; at least one such node must exist in an acyclic graph.

Then we will follow this algorithm,

Pseudo Code For Algorithm

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edges
while S is non-empty do
    remove a node n from S
    insert n into L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
if graph has edges then
    return error (graph has at least one cycle)
```

1 http://en.wikipedia.org/wiki/Topological_sorting

```
else
   return L (a topologically sorted order)
```

Lets see a sample Java Implementation,

Java Source

```
public class Node {
   public int data;
   public final HashSet<Edge> inEdges;
   public final HashSet<Edge> outEdges;

   public Node(int data) {
      this.data = data;
      inEdges = new HashSet<Edge>();
      outEdges = new HashSet<Edge>();
   }
   public Node addEdge(Node node) {
      Edge e = new Edge(this, node);
      outEdges.add(e);
      node.inEdges.add(e);
      return this;
   }
  @Override
   public String toString() {
      return "" + data;
   }
   public static class Edge{
      public final Node from;
      public final Node to;
      public Edge(Node from, Node to) {
         this.from = from;
         this.to = to;
      }
      @Override
      public boolean equals(Object obj) {
         Edge e = (Edge)obj;
         return e.from == from && e.to == to;
      }
   }
}


public class TaskResolver {

   public static void main(String[] args) {
      Node one = new Node(1);
      Node two = new Node(2);
      Node three = new Node(3);
      Node four = new Node(4);
      Node five = new Node(5);

      //Adding dependencies to the nodes.
      three.addEdge(one).addEdge(five);  // task 3 depends on 1 & 5.
      two.addEdge(five).addEdge(three);         // Task 2 depends on 5 & 3.
      four.addEdge(three);
```

```
        five.addEdge(one);

        Node[] allNodes = {one, two, three, four, five,};
        //L <- Empty list that will contain the sorted elements

        ArrayList<Node> L = new ArrayList<Node>();
        //S <- Set of all nodes with no incoming edges
        HashSet<Node> S = new HashSet<Node>();
        for (Node n : allNodes) {
            if (n.inEdges.size() == 0) {
                S.add(n);
            }
        }

        //while S is non-empty do
        while(!S.isEmpty()){
            //remove a node n from S
            Node n = S.iterator().next();
            S.remove(n);

            //insert n into L
            L.add(n);

            //for each node m with an edge e from n to m do
            for(Iterator<Node.Edge> it = n.outEdges.iterator();it.hasNext();){
                //remove edge e from the graph
                Edge e = it.next();
                Node m = e.to;
                it.remove();//Remove edge from n
                m.inEdges.remove(e);//Remove edge from m

                //if m has no other incoming edges then insert m into S
                if(m.inEdges.isEmpty()){
                    S.add(m);
                }
            }
        }

        //Check to see if all edges are removed
        boolean cycle = false;
        for(Node n : allNodes){
            if(!n.inEdges.isEmpty()){
                cycle = true;
                break;
            }
        }

        if(cycle){
            System.out.println("Cycle present, topological sort not possible");
        }else{
            System.out.println("Topological Sort: "+ Arrays.toString(L.toArray()));
        }
    }
}
```

Approach 2

We can use HashMap to solve this problem.

1. Create a List of Tasks which will hold all completed tasks.
1. Find the tasks that are independent, like task 1 in above example and put it into a completed task list.
2. Build a map of task -> List of dependent tasks
3. Traverse the entire entrySet and remove all the tasks whose dependent tasks in completed task basket.
4. Add all tasks to the completed task basket whose dependent tasks are completed.
5. Repeat step 3-4 till the size of the map becomes zero.

References

http://en.wikipedia.org/wiki/Topological_sorting
http://www.electricmonk.nl/log/2008/08/07/dependency-resolving-algorithm/
http://stackoverflow.com/questions/2739392/sample-directed-graph-and-topological-sort-code

Question : How would you sort 900 MB of data using 100 MB of RAM ? What is external sort ?

SOLUTION

Algorithm

External Merge Sort is the answer to the above mentioned problem.

1. Read 100 MB of data in main memory and sort by some conventional method like quicksort.
2. Write the sorted data to the disk.
3. Repeat step 1 & 2 until all the data is in sorted 100 MB chunks (9 chunks) which now need to be merged into single output file.
4. Read first 10 MB of each sorted chunk into input buffer in main memory and allocate remaining 10 MB for the output buffer.
5. Perform 9 way merge and store the result in output buffer.

Lets try to understand this with a concrete example,

Imagine you have numbers 1-9,
9 7 2 6 3 4 8 5 1
And lets suppose that only 3 fit in the main memory.
So break the data into 3 chunks, sort each, store in separate files. The contents of 3 files will now become
2 7 9
3 4 6
1 5 8
Now you would open each of 3 files as streams and read the first value from each.
2 3 1
Output the lowest value 1, and get the next value from that stream, now you have
2 3 5
output the next lowest value , 2 and continue onwards until you have outputted the entire sorted list.

Notes

Similar Questions

Question
There are 2 huge files A and B which contains numbers in sorted order. Make a combined file C which contains the total sorted order.

Solution
Merge Sort technique.

Question
There are k files each containing millions of numbers. How would you create a combined sort file out of these ?

Solution
- Use a binary-min heap (increasing order, smallest at the top) of size k, where k is the number of files.
- Read first record from all the k files into the heap.
- Loop until all k files are empty.
- Poll() the minimum element from the binary heap, and append it to the file.

- Read the next element from the file from which the minimum element came.
- If some file has no more element, then remove it from the loop.
- In this way we will have one big file with all number sorted
- Time complexity will be O (n log k)

References

http://en.wikipedia.org/wiki/External_sorting#External_merge_sort
http://exceptional-code.blogspot.in/2011/07/external-sorting-for-sorting-large.html
http://en.wikipedia.org/wiki/Merge_sort
http://www.careercup.com/question?id=83696

Question: How would you design minimum number of platforms so that the buses can be accommodated as per their schedule ?

BUS	Arrival Time (HRS)	Departure Time (HRS)
A	0900	0930
B	0915	1300
C	1030	1100
D	1045	1145
E	1100	1400

Bus Schedule for a given Platform

SOLUTION

This problem is about finding the peak time when maximum number of buses are waiting to get into platform. Ideally we would not like to stop a bus outside the platform so every single bus would require one platform. So in this problem, the maximum number of buses arriving at the same time during the peak time of day will decide the number of platforms.

Algorithm Skills Required : Sorting, Finding Max from an Array.

Approach

Calculate the peak time of the buses from the given bus schedule, the number of buses at the peak time will give us the number of platforms.

Step 1

Create a single array of bus timing after append A (for Arrival) and D (for departure) to each bus time.
So the above table should now become a one dimensional array like this :
[0900A, 0930D, 0915A, 1300D, 1030A, 1100D, 1045A, 1145D, 1100A, 1400D]

Step 2

Sort the above array in ascending order (Natural Order)
[0900A, 0915A, 0930D, 1030A, 1045A, 1100A, 1100D, 1145D, 1300D, 1400D]

Step 3

Now traverse the entire array and count the maximum number of Buses at peak time.

Pseudo Code

```
var counter, max, peakTime;
for(each time entry){
        if(arrival)
          counter++;
        else
          counter--;
        if(counter > max)
        max= counter
        peakTime= time entry;
}
```

Here the max is the maximum number of platforms that should be built for accommodating all the buses at peak time as per given time table.

Java Source

```
public class PlatformDesigner {

    static class BusSchedule {
        final String name;
        final String arrivalTime;
        final String departureTime;

        BusSchedule(String name, String arrivalTime, String departureTime) {
            this.name = name;
            this.arrivalTime = arrivalTime;
            this.departureTime = departureTime;
        }
    }

    private List<BusSchedule> busScheduleList = new ArrayList<>();

    public void addBusSchedule(BusSchedule schedule) {
        busScheduleList.add(schedule);
    }

  public void calculateNoOfPlatforms(){
        List<String> scheduleTokens = new ArrayList<>();
        for (BusSchedule schedule : busScheduleList) {
            scheduleTokens.add(schedule.arrivalTime+"A");
            scheduleTokens.add(schedule.departureTime+"D");
     }

Collections.sort(scheduleTokens);


  int max=0, counter =0;
  String peakTime ="";

        for (String token : scheduleTokens) {
            if(token.endsWith("A")){
                counter++;
            }else if(token.endsWith("D")){
                counter--;
            }

           if(counter > max){
                max=counter;
                peakTime = token;
            }
        }

        System.out.println("max number of platforms required at peak time ["+peakTime+"] = " + max);
    }
}
```

Question: Design a parking lot where cars and motorcycles can be parked. What data structure to use for finding free parking spot in Parking Lot program? Assume there are million of parking slots.

SOLUTION

Approach

1. Create model classes to OO map Vehicle, vehicleType, Slot, SlotSize, etc.

```
public class Vehicle {
    private String id;
    private String owner;
    private long mobile;
    private String inTime;
    private String outTime;
    private VehicleType vehicleType;
      //create the getters and setters for the above fields
}

public class SlotSize {
    private final int size;
    public SlotSize(int size) {this.size = size;}
}

public enum VehicleType {
    BIKE(new SlotSize(1), "Motor Bike"),
    CAR(new SlotSize(2), "Car"),
    TRUCK(new SlotSize(4), "Truck");

    private final ParkingSlotSize slotSize;
    private final String vehicleName;
    private VehicleType(SlotSize slotSize, String vehicleName) {...}
}

public class Slot {
        int slotid;
        int floorNo;
        boolean occupied;
        SlotSize size;
}
```

2. Create parking manager which will track the free parking slots using a Queue for fast retrieval, and occupied vehicle mapping will be stored using a HashMap for fast O(1) retrieval. Whenever a slot gets free, remove it from the HashMap and add it into Queue, and if new vehicle comes in then pick slot from the head of queue and store the mapping in hashmap. Separate Queue & HashMap could be used for Motor Bike, Truck & Car vehicle type.

```
public class ParkingManager {
private Queue<Slot> slots = new PriorityQueue<>();
private HashMap<Vehicle, Slot> parkDetail = new HashMap<Vehicle, Slot>();
public Slot getVehicle(String id) {...}    //Logic for searching a vehicle using vehicle Id or slot id.
public Slot park(Vehicle vehicle) {...}   //save the <vehicle - slot> mapping inside a hash table
}
```

Notes

A free list is a data structure used in a scheme for dynamic memory allocation. It operates by connecting unallocated regions of memory together in a linked list, using the first word of each unallocated region as a pointer to the next. It's most suitable for allocating from a memory pool, where all objects have the same size.

Free lists make the allocation and deallocation operations very simple. To free a region, one would just link it to the free list. To allocate a region, one would simply remove a single region from the end of the free list and use it. If the regions are variable-sized, one may have to search for a region of large enough size, which can be expensive.

Maintain a PriorityQueue for free parking space, use hashmap for the filled spaces. In this manner it would be easier to find the free space and to find the parked object. Assume that the parking space on the lower floor gets more priority than the parking space on the higher floor, when a new car comes in just pick the top most entry from the parking queue and park the object

References

http://www.careercup.com/question?id=12626681

Question: Design a web crawler that will crawl for links(urls).

SOLUTION

Solution :
http://www.careercup.com/question?id=9996850

Use Queue to add links in that page

```
while(!queue.isEmpty())
{
ParentLink=queue.removeFirstElement(){
Page = Fetch(ParentLink)
For(All Link in the Page)
{

Queue.AddElement(Link)
}

}
```

Question : Implement the classes to model two pieces of furniture (Desk and Chair) that can be constructed of one of two kinds of materials (Steel and Oak). The classes representing every piece of furniture must have a method getIgnitionPoint() that returns the integer temperature at which its material will combust. The design must be extensible to allow other pieces of furniture and other materials to be added later. Do not use multiple inheritance to implement the classes.

SOLUTION

Design

Abstract Factory along with Bridge Pattern can solve this problem.

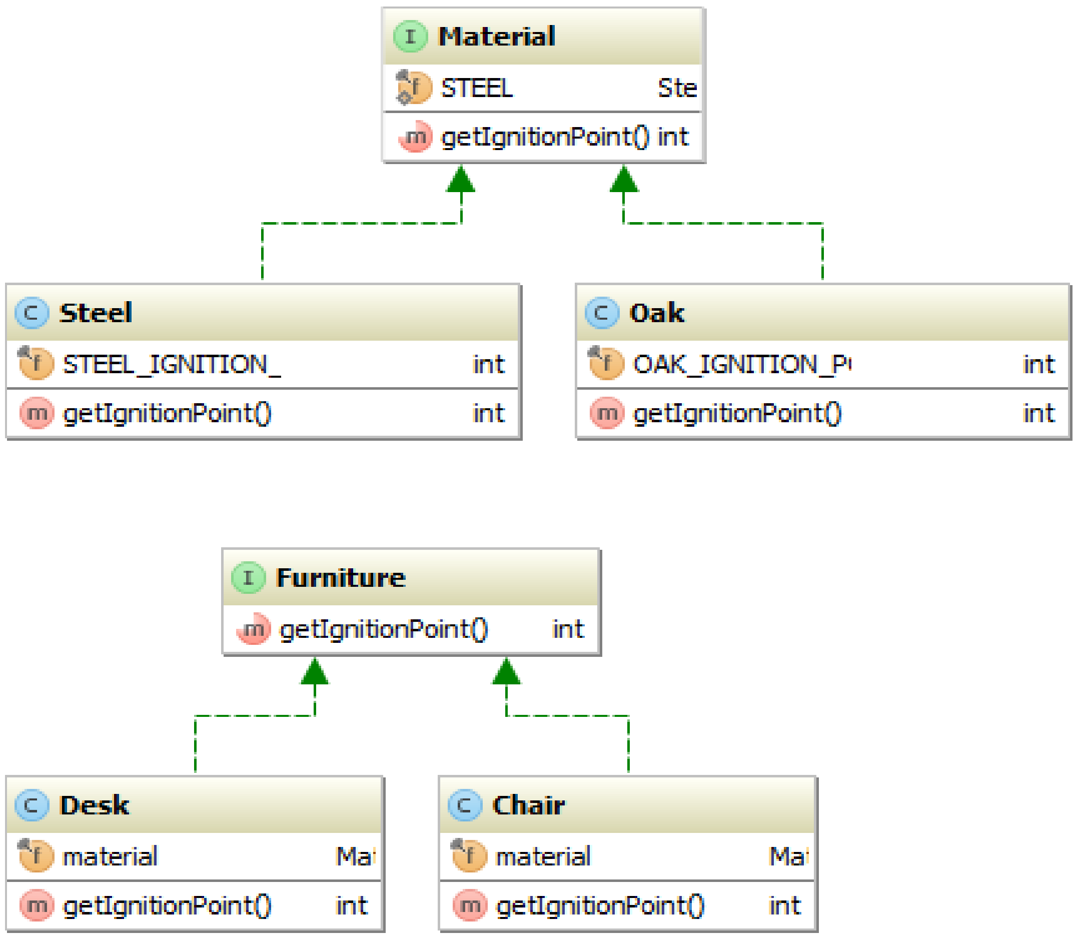
```
public interface Furniture {
    public int getIgnitionPoint();
}

public interface Material {
    final Material STEEL = new Steel();
    final Material OAK = new Oak();
    int getIgnitionPoint();
}

public class Steel implements Material {
    private final int STEEL_IGNITION_POINT = 1500;
    @Override
    public int getIgnitionPoint() {return STEEL_IGNITION_POINT;}
}

class Oak implements Material {
    private final int OAK_IGNITION_POINT = 900;
    @Override
    public int getIgnitionPoint() { return OAK_IGNITION_POINT; }
}

public class Desk implements Furniture {
    private final Material material;
    public Desk(Material material) {
        this.material = material;
    }
    @Override
    public int getIgnitionPoint() {return material.getIgnitionPoint();}
}

public class FurnitureFactory {
    public Furniture createChair(Material material){
       return new Chair(material);
    }

    public Furniture createDesk(Material material){
        return new Desk(material);
    }
    public static void main(String[] args) {
        FurnitureFactory factory = new FurnitureFactory();
        Furniture desk = factory.createDesk(Material.STEEL);
        int ignitionPoint = desk.getIgnitionPoint();
        System.out.println("ignitionPoint = " + ignitionPoint);
    }
}
```

Notes

References
http://www.careercup.com/question?id=14796858

Abstract Factory (interface Furniture), Concrete Factory (class Desk and Chair)
Abstract Product (interface Material) and Concrete Product (class Oak and Steel)

Class Diagram



Chapter 6

Puzzles

Question : Why man holes are round in shape ?

SOLUTION

There are multiple reasons for that

1. Round shape is easy to machine compared to any other shape, thus it reduces the effort and cost.
2. Round objects are easy to move by rolling them, thus labour is reduced.
3. Round shape objects can't fell into the hole, for other shapes its not true.

Question : Solve two egg problem ?

SOLUTION

What we need is a solution that minimizes our maximum regret. The examples above hint towards what we need is a strategy that tries to make solutions to all possible answers the same depth (same number of drops). The way to reduce the worst case is to attempt to make all cases take the same number of drops.

As I hope you can see by now, if the solution lays somewhere in a floor low down, then we have extra-headroom when we need to step by singles, but, as we get higher up the building, we've already used drop chances to get there, so there we have less drops left when we have to switch to going floor-by-floor.

Let's break out some algebra.

Imagine we drop our first egg from floor n, if it breaks, we can step through the previous (n-1) floors one-by-one.

If it doesn't break, rather than jumping up another n floors, instead we should step up just (n-1) floors (because we have one less drop available if we have to switch to one-by-one floors), so the next floor we should try is floor n + (n-1)
Similarly, if this drop does not break, we next need to jump up to floor n + (n-1) + (n-2), then floor n + (n-1) + (n-2) + (n-3) …
We keep reducing the step by one each time we jump up, until that step-up is just one floor, and get the following equation for a 100 floor building:
$n + (n-1) + (n-2) + (n-3) + (n-4) + \ldots + 1 >= 100$

This summation, as many will recognize, is the formula for triangular numbers (which kind of makes sense, since we're reducing the step by one each drop we make) and can be simplified to:
$n (n+1) / 2 >= 100$

This is a quadratic equation, with the positive root of 13.651 (Which we have to round up to 14. This is not a John Malkovich movie!).

Reference
http://www.datagenetics.com/blog/july22012/index.html

Question : There are 100 doors all closed initially. 1st iteration opens all doors (1x multiplier) 2nd iteration opens 2,4,6,8 .. doors (2x multiplier) 3rd iteration opens 3,6,9,12 ... doors (3x multiplier) and so on. In the end of 100 iterations, which all doors will be in open state ?

SOLUTION

A door will be left closed if even number of door toggling iterations for that particular door. For all odd number toggling, the door state will be open.

Now let's see how we can approach towards the solution.
In mathematics the number of factors that can divide a given number can easily decide the solution. The doors will be ultimately closed where the number of factors of that number are even.

For example,
The factors of 1 are 1 and itself. It has one factor - odd factors
The factors of 2, 3, and 5 are 1 and themselves. They have two factors - even factors
The factors of 4 are 1 and itself, as well as 2. 4 has three factors - odd factors

After analysis we can find that every number has even number factors except the number which are perfect squares.
1 = 1x1; (both the factors are single number 1)
4=1x4, 4x4 (both the numbers are single number 4), 4x1;
9=1x9, 3x3 (both the numbers are single number 3), 9x1;

So in case of perfect squares, both the factors are same number, that trick will solve this problem.

So we can conclude that odd number of iterations will happen for all the numbers which are perfect square numbers and those doors will be in Closed state (initially opened)
1,4,9,16,25,36,49,64,81,100

Chapter 7

Glossary

Question : What is financial Instrument, Bond, equity, Asset, future, option, swap and stock with example each ?

SOLUTION

Bond

A bond is a debt security under which the issuer owes the holders a debt and depending on the terms of the bond, is obliged to pay them interest/coupon and to repay the principal at a later date, termed as maturity.

Stock

Constitutes the equity stake of its owners.

Equity

Equity is the residual claim or interest of the most junior class of investors in asset after all liabilities are paid.

Asset

Assets are economic resources. Anything tangible or intangible that is capable of being owned/controlled to produce value and that is held to have positive economic value is considered an asset. In other words, Asset represents value of ownership that can be converted into cash.
Capital = Assets - Liabilities

Coupling

Coupling is the degree to which each program module relies on each one of other module in the software application.

Question : Toolkit & Resources for a Java Developer.

SOLUTION

Essential Tool kit for Java Developer

IntelliJ IDE
Java DB
Twitter Bootstrap CSS library
Jquery
Freemarker
Struts 2
Servlets
Tortoise SVN
Apache Web Server
Jetty Server
HTML 5
Firebug extension for mozilla Firefox
Cygwin for Unix simulation

Books

Design Patterns in Java - Head First
Concurrency In Practice
Algorithms 4th edition : http://algs4.cs.princeton.edu/home/
Cracking the coding interview at CareerCup
Effective Java 2nd Edition

Technology Forums

http://www.geeksforgeeks.org/fundamentals-of-algorithms/
http://www.careercup.com
http://www.stackoverflow.com

Great Tutorials Articles

Few articles on Java 6 at IBM website
http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=5+things+you+did
Concurrency In Practice by Brian Goetz - http://www.briangoetz.com/pubs.html
Java Articles : http://www.oracle.com/technetwork/articles/java/index.html
http://www.oracle.com/technetwork/articles/java/index.html
Java SE Tutorial : http://docs.oracle.com/javase/tutorial/index.html
Java 7 docs: http://docs.oracle.com/javase/7/docs/

Video Tutorials
http://www.youtube.com/user/nptelhrd

Topics To Cover
Bitwise operators, Binary trees, Number System

Chapter 8

Sample Interview Questions

Question : How would you implement a optimistic database locking using various techniques ?How would you avoid non-repeatable reads ?
Hint: https://blogs.oracle.com/carolmcdonald/entry/jpa_2_0_concurrency_and
https://blogs.oracle.com/enterprisetechtips/entry/preventing_non_repeatable_reads_in
Question : Why would you choose a class to be static nested inner class ?
Question : What is the best way to implement PriorityQueue ?
Question : Java serialization using Serializable vs Externizable ?
Question : How does quick sort works ?
Question : How would you print rhyming words from a stream using PipedStreams ?
list of words -> reverse -> sort -> reverse
http://www.cs.nccu.edu.tw/~linw/javadoc/tutorial/java/io/streampairs.html#PIPED
http://www.dca.fee.unicamp.br/projects/sapiens/calm/References/Java/tutorial/essential/io/pipedstreams.html
Question : How would redesign executors framework by reducing the thread contention ?
http://today.java.net/article/2011/06/14/method-reducing-contention-and-overhead-worker-queues-multithreaded-java-applications
Question : How would you find nth highest salary from a table using SQL ?
"Select *
From Employee E1
Where
N = (Select Count(Distinct E2.Salary) From Employee E2 Where E2.Salary >= E1.Salary)"
Question : How would you delete duplicate rows from a table ?
Question : what are Outer Joins and Inner Joins ?
Question : Lowest Common Ancestor in a Binary Search Tree
Question : Your are give a file with milions of numbers in it. Find the duplicate numbers in it.
Question : Quickest way to Find the missing number in an array of 1 to n. slot in array is blank, Find 2 missing number.."Hint : Priority queue with distance from the current floor as the inverse of priority
Question : How would you traverse Binary Search tree using iterations ?
Question : what do you understand by MVC design pattern ?
Question : What is dependency Injection design pattern ?
Question : TreeMap vs ConcurrentSkipListMap ?
Question : Inline sorting of an array where max element in the array is equal to size of the array itself ? and most of the elements are unique.
Question : InOrder traversal of a binary search tree results in Ascending order sorting of elements.
Question : Find the top 10 most frequent words in a file.
Hint: use hashmap to store word occurrence count and for every 10th occurrence update the values in a binary heap (PriorityQueue) with the maximum count. PriorityQueue can just store the top 10 elements
http://stackoverflow.com/questions/742125/how-to-find-high-frequency-words-in-a-book-in-an-environment-low-on-memory
http://stackoverflow.com/questions/358240/space-efficient-data-structure-for-storing-a-word-list
http://en.wikipedia.org/wiki/Bloom%5Ffilter
Question : Provide with the resume template for your resume.
Question : Circular Linked List - take 2 pointers
increment one by +1;
increment other by +2
they will first meet in N iterations
length of stem = n/2;
Question : How would you lower thread contention using Dequeue.
http://today.java.net/article/2011/06/14/method-reducing-contention-and-overhead-worker-queues-multithreaded-java-applications
Question: What is huffman encoding technique ?

Continued on 155

http://en.wikipedia.org/wiki/Huffman_coding
Question : Use Stack to implement a Queue… you can use 2 stacks
Question : Use 2 stacks to implement getMin() function with O(1) complexity.
Question : BFS and DFS of a Tree
Question :3 ants are at 3 vertices of a triangle. They randomly start moving towards another vertex. What is the probability that they don't collide?
Question :You have 2 ropes which burns in 1 hr..measure 45mins.
Question :"given 1000 bottles of juice, one of them contains poison and tastes bitter. Spot the spoiled bottle in minimum sips?"
Question : How would you detect a circular loop inside a linked list ?
Question : How would you calculate size of a linked list having circular loop in it ?
Question : There is a sorted Array of Integer but the array is rotated. How would you fix it using binary search ? why choose binary search ?
http://leetcode.com/2010/04/searching-element-in-rotated-array.html
http://www.careercup.com/question?id=2800
http://xorswap.com/questions/77-implement-binary-search-for-a-sorted-integer-array-that-has-been-rotated
Question: How would you mirror a binary tree ?

Question : I want to implement 2 different display score boards for the IPL cricket match, one specific to IPL another for T20. Which design pattern will rescue you in this case ?
Question : What is contract between equals() and hashcode() method ?
Question : How would you write a hashcode() method for a class having two fields ? Can we multiply hashcode with a random number ?
Question : Fork and Join with example ?
http://www.oracle.com/technetwork/articles/java/fork-join-422606.html
http://www.oracle.com/technetwork/articles/java/trywithresources-401775.html
http://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html
http://fahdshariff.blogspot.in/2012/08/java-7-forkjoin-framework-example.html
http://www.javabeat.net/2012/06/simple-introduction-to-fork-join-framework-in-java-7/
http://www.vogella.com/articles/JavaConcurrency/article.html
Question : http://tech-queries.blogspot.in/2008/11/sort-array-containing-0-and-1.html
Question : Why wait() and notify() are at Object level rather than Thread level ?
Answer : http://javarevisited.blogspot.in/2012/02/why-wait-notify-and-notifyall-is.html

Question : Why not to choose static factory method in place of singleton design pattern ?

Question: There is an JPA entity having lazy loading items. You want to use this entity to render a view page which will display this entity. What all options do you have to overcome the lazy loading in this case ?
http://wiki.eclipse.org/Introduction_to_Mappings_(ELUG)#Indirection_.28Lazy_Loading.29
http://java.dzone.com/articles/jpa-lazy-loading
http://www.javacodegeeks.com/2012/07/four-solutions-to-lazyinitializationexc_05.html

Question : find median of two sorted array
Hint :http://www.geeksforgeeks.org/median-of-two-sorted-arrays/
Question : When no direct method is found, the most specific method is chosen by the JVM for a method call ? example ?
Question : What are various techniques for achieving thread safety in Java : Immutable, ThreadLocal, Synchronized access, non-blocking algo using CAS.
Question : http://www.geeksforgeeks.org/find-the-two-repeating-elements-in-a-given-array/
Question : Write a print method for printing Top N numbers from an array ?

continued on 156

Question : Given a collection of Trades. Write an algorithm to remove duplicates based on Tic# and sorting based on NAV.
Question : Design a vending machine.
Question : How would you implement a BoundedBuffer using Lock and Condition ?
http://www.baptiste-wicht.com/2010/09/java-concurrency-part-5-monitors-locks-and-conditions/
Question : Give an example of timed locking using explicit locking in Java.
Hint : http://codeidol.com/java/java-concurrency/Explicit-Locks/Lock-and-ReentrantLock/
Question : Fibonacci Series using various techniques - recursive, iterative, Big O(1)
Question : Left Outer Join vs Right Outer Join ?
Hint : http://en.wikipedia.org/wiki/Join_(SQL)#Inner_join
Question : Angle between Minute Hand and Hour hand of a clock ?
Question : How is a TreeSet implemented in Java ?
Question : How does google analytics works without causing a extra load on your web server ?
Hint : Javascript is used to hit a google server with the required identifier then the new site is visited.
Question: How would you avoid data corruption by a web page which allows a update a database row, and 2 users try to update the same row simultaneously.
Question : How does batch update works in Java ?
Question : Find the first common ancestor of two given nodes in a binary tree in o log n space complexity and O(n) time complexity.
Hint - 1. do DFS, 2. during DFS if you find one of the nodes store the stack contents (path from the root) repeat the same process for the second node. this requires 2nlogn space. 3. Now compare both of these paths from the root, the last common node in the path is the first common ancestor. this takes logn time avg case and n in worst case
Question : What is a BloomFilter? How is it better than hashmap in certain casess?
http://en.wikipedia.org/wiki/Bloom%5Ffilter
"Sort An Array Containing '0' And '1'
Sort An Array Containing '0','1' And '2'
http://tech-queries.blogspot.in/2008/11/sort-array-containing-0-and-1.html
Dutch flag algo."
http://en.wikipedia.org/wiki/Dutch_national_flag_problem
Discuss the numbering system.
traceroute command and nslookup
Natural ordering in search operation and stable search
TRIE
Sorting and Searching
Atomic package and CAS, non blocking algorithms
Database indexes clustered indexes, query plan etc, Outer and Inner Join
UNIX stuff cut grep ps etc, piping the output
Question : How would you implement a Trie in Java. Suppose you want to implement a Auto-suggest functionality using Java where user presses a letter and all the words starting with that letter are listed in the suggestion box. Which datastructure would you choose for this scenario ?
Question: How would you implement ThreadPool in Java 1.4 ?
Question: How would you implement StringBuffer class, so that it doesn't create un-necessary imtermediate string objects upon modifications ?

Question : Write a method to count the size of a LinkedList given a Node. There could be a circular loop inside the list, hence your method should be smart enough to handle such situation gracefully.
Hint : http://crackinterviewtoday.wordpress.com/2010/03/16/loop-in-a-linked-list/

Continued on 157

Questions from Wallmart.com

How would you design a FileSystem for showing just the hierarchy of files ? File Interface & then File and DIR as the subclasses.
How would you map department and employee table into Java Objects ? What kind of relationship would you put there? Lazy loading ?
Which object construction mechanism you prefer in Spring DI - constructor based or setter based.
We have list of One million numbers on which some mathematical function needs to be applied, How would you make algorithm concurrent ?
There is an un-ordered stack of 5 elements and we have a method nextMinimum() which returns us the subsequent next minimum element in O(1). suppose we have 2,3,1,4,5 as the elements, then first invocation will return us 1, second 2, third 3.
Hint - maintain a queue which maintains the sorted references to the underlying stack.
What is Spring bean lifecycle ?
What is embeddable in JPA
How do you performance tune an application - By GC, by changing algorithm, using different data structure which is more appropriate for a given scenario.
Question: Design multi-player Chess Game using Class Diagrams.
Question: Design a Restaurant Reservation system.
Solution : http://www.careercup.com/question?id=15062886
Question: Design SkyDrive.
http://www.careercup.com/question?id=14692764
http://thought-works.blogspot.in/2012/11/object-oriented-design-for-cloud-based.html
Question: Design Online Auction Site.
Solution : http://thought-works.blogspot.in/2012/11/object-oriented-design-for-online.html

Question: Design a Train & reservation system. Give class structure and design UML
Solution :
http://www.careercup.com/question?id=3220674

Question: Write a 2 Thread application where one thread prints even number and the other thread prints odd numbers and both of them act in a synchronized manner.
Question : Security and Performance Tuning of a REST and Ajax Application
http://www.oracle.com/technetwork/articles/java/securityperf-rest-ajax-177520.html
http://www.oracle.com/technetwork/articles/javaee/jax-rs-159890.html
Question : How does Tree Balancing works ? left and right rotation ?
Question : How does ReentrantReadWriteLock works internally ?
Question : What do you understand by volatile keyword ?
http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html
Question : Why would you Suffix Tree for searching?
Question : Write a chapter on concepts. Big-O notation, Recursion, Bitwise Hacks, performance tuning, JMX
Write a chapter on glossary. mention few keywords used in Java and financial word. jargons
Can you tell me with example the Usage of ThreadLocal class ? Calendar class, JDBC transaction management, etc.
synchronization of getClass() in case of inheritance, will lock the actual class rather than the whole hierarchy.
Question: How would you convert a sorted integer array to height balanced binary tree. ?

Question: Discuss about non-blocking algorithms using CAS ?
https://www.ibm.com/developerworks/java/library/j-jtp04186/
Discuss the numbering system.

Continued on 158

traceroute command and nslookup
TRIE
Sorting and Searching
Atomic package and CAS, non blocking algorithms
Database indexes clustered indexes, query plan etc, Outer and Inner Join
UNIX stuff cut grep ps etc, piping the output
What are the ways to achieve thread-safety in a concurrent program ?
Dealing with ConcurrentModificationException ?
How to expose a method over JMX using MBean ?
Thread.interrupt() puzzle.
Discuss on External Sorting
http://www.careercup.com/question?id=83696
Question : Design a solution to print employee hierarchy in Java given a employee record.
Java Software in Harmony with the Hardware - Mark Thompson
http://mechanical-sympathy.blogspot.in/2012/10/compact-off-heap-structurestuples-in.html
http://mechanical-sympathy.blogspot.in/2011/07/false-sharing.html
http://mechanical-sympathy.blogspot.in/2011/12/java-sequential-io-performance.html

Question : How would you find kth highest number in a list of n Numbers ?
http://en.wikipedia.org/wiki/Selection_algorithm
http://stackoverflow.com/questions/3628718/find-the-2nd-largest-element-in-an-array-with-minimum-of-comparisom
http://stackoverflow.com/questions/251781/how-to-find-the-kth-largest-element-in-an-unsorted-array-of-length-n-in-on
Question: Design Coffee maker (Vending Machine)..provide some class diagram
Solution : http://www.careercup.com/question?id=3171714
http://stackoverflow.com/questions/7067044/java-algorithm-to-solve-vendor-machine-change-giving-problem
Command design pattern with Factory

CoffeeBuilder [Abstract]
Cappucino [Concrete] ....
CoffeeMaker <---- director
Coffee <-- Product

Builder + Factory Method

Question: How do you represent the following expression in "class design": (5*3)+(4/2) ? How would an algorithm that computes the value of this expression work?
http://www.careercup.com/question?id=65911

Question : How would you design Money class, which holds currency as well as amount of money ?
Hint : http://stackoverflow.com/questions/1359817/using-bigdecimal-to-work-with-currencies
http://stackoverflow.com/questions/11434938/display-curreny-value-in-india-as-rs-100-using-java
http://www.javapractices.com/topic/TopicAction.do?Id=13

How would you perform tree rotation to balance a tree.

Question : UNIX Questions.

SOLUTION

How to invert the grep to exclude a given search pattern ?

Assume we have a book-order.txt file and we want to search all orders which do not contain word "gurgaon", then we can use the following command for searching.

grep -v gurgaon book-orders.txt

How to count number of lines in a file in UNIX ?

wc command can be used to count bytes, lines, characters, and words in a given file

```
ls -ltr | wc -l
wc -c abc.txt
wc -l abc.txt
```

How would you find which UNIX operating system you are running ?

Following commands can be used to know the OS details -

```
uname -a
arch
```

How would you find the number of cpu's in your computer ?

```
cat /etc/cpuinfo
```

How would you print the running processes ?

```
ps -elf | grep <identifier>
```