* 자료구조

1. 대표적인 자료구조 4가지에 대해 설명하시오

* 자료구조에는 대표적으로 리스트 , 트리 , 맵 , 집합이 있다

1. 자료구조 리스트에 대해 설명하시오

* **순서**를 가지고 일렬로 나열한 원소들의 모임

[*추상적 자료형*](https://namu.wiki/w/%EC%B6%94%EC%83%81%EC%A0%81%20%EC%9E%90%EB%A3%8C%ED%98%95)*,*[*자료구조*](https://namu.wiki/w/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0)*의 하나. 순열(Sequence)이라고도 불리며, 순서가 있다는 점에서*[*집합*](https://namu.wiki/w/%EC%A7%91%ED%95%A9)*과는 구별되며, 갈림길 없이 일렬로 나열되어 처음과 끝이 각각 하나씩만 있다는 점에서*[*그래프*](https://namu.wiki/w/%EA%B7%B8%EB%9E%98%ED%94%84)*와도 구별된다*

1. 배열과 연결 리스트 간의 차이점에 대해 설명하시오

* 배열은 생성과 동시에 크기가 정해진다,

이에 반해 리스트는 원소간에 연결로 구성되어 있어서 크기의 제한에서 자유롭다

1. 배열과 연결 리스트의 각각 장단점에 대해서 설명하시오

* 배열의 장점은

첫번째로 검색 성능이 좋다 ,

인덱스를 활용해서 무작위 접근 (Random Access) 이 가능하고

순차 접근 (Sequential Access) 의 경우에도

배열은 데이터를 하나의 연속된 메모리 공간에 할당하므로

연결리스트 보다 빠르다

둘째로는 구현이 쉽다

셋째로는 참조를 위한 추가적인 메모리 할당이 필요 없다

반면 배열의 단점으로는

첫번째로 삽입과 삭제시 비효율적이다 ,

배열은 삽입과 삭제시에 다음 항목에 해당하는 모든 요소를 모두 이동 해야 하므로 많은 연산이 수행된다

둘째로는 크기를 바꿀수 없다 ,

생성시 지정된 크기를 바꿀 수 없기 때문에 메모리 낭비 가능성이 크다

셋째로는 메모리 재사용이 불가능하다 ,

생성시에 할당 받은 메모리 만큼 사용하는데 해당 메모리에 데이터 존재 유무에 상관없이 계속해서 메모를 점유하고 있는다

* 리스트의 장점은

첫번째로는 노드의 삽입과 삭제시에 효율적이다 ,

삭제하고자 하는 노드나 삽입하고자 하는 노드 의 전후 노드의 전후관계를 수정하면 되기 때문이다

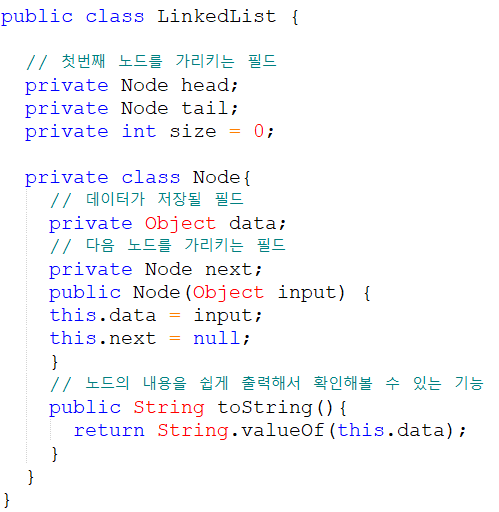
두번째로는 크기가 고정적이지 않다 ,

노드가 하나씩 삽입될때마다 독립적인 공간의 메모리를 차지하기 때문에 크기에 대한 제한이 없다

세번째로는 메모리 재사용이 가능하다 ,

노드가 삭제되면 참조 정보가 사라지므로 나중에 가비지 컬렉터에 의해서 가용될수 있는 메모리로 재활용 할수 있다

1. 자료구조 리스트를 자바로 간략히 구현하시오



1. 큐(Queue) 와 덱 (Deque) 에 대해 설명하고 자바로 구현 하시오

* 큐(Queue)는 자료구조 리스트의 구현체중 하나로서 삽입된 순서대로 삭제되는 **선입선출**(FIFO, First-In-First-Out)의 구조

**QueueNode.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package Queue;    public class QueueNode {      char item;      QueueNode next;        public QueueNode() {          this.next = null;      }        public QueueNode(char item){          this.item = item;          next = null;      }        public char getItem(){          return this.item;      }  } |

**LinkedQueue.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80 | package Queue;    public class LinkedQueue implements Queue{      private QueueNode front;      private QueueNode rear;        public LinkedQueue() {          front = null;          rear = null;      }        @Override      public boolean isEmpty() {          return (front==null);      }        @Override      public void enQueue(char item) {          QueueNode node = new QueueNode(item);            if(isEmpty()){              front = node;              rear = node;          }else{              rear.next = node;              rear = node;          }      }        @Override      public char deQueue() {          if(isEmpty()){              System.out.println("큐의 내용이 존재하지 않습니다.");              return 0;          }else{              char item = front.item;              front = front.next;              if(front==null){                  rear=null;              }              return item;          }      }        @Override      public void delete() {          if(isEmpty()){              System.out.println("큐의 내용이 존재하지 않습니다.");          }else{              front = front.next;              if(front==null){                  rear=null;              }          }        }        @Override      public char peek() {          if(isEmpty()){              System.out.println("큐의 내용이 존재하지 않습니다.");              return 0;          }else{              return front.item;          }      }        public void print() {          if(isEmpty()){              System.out.println("큐의 내용이 존재하지 않습니다.");          }else{              QueueNode node = front;              while(node!=null){                  System.out.print(node.item + " ");                  node = node.next;              }              System.out.println();          }      }  } |

* 큐의 양쪽 끝에서 삽입과 삭제가 모두 발생할 수 있는 큐로서, 큐와 스택의 성질을 모두 가지고 있는 자료구조이다

**DequeNode.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package Deque;    public class DequeNode {      char data;      DequeNode llink;      DequeNode rlink;        public DequeNode() {          this.llink = null;          this.rlink = null;      }        public DequeNode(char data) {          this.data = data;          this.llink = null;          this.rlink = null;      }  } |

**Deque.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138 | package Deque;    public class Deque {      DequeNode front;      DequeNode rear;        public Deque() {          front = null;          rear = null;      }        public boolean isEmpty(){          return (front==null);      }        public void insertFront(char item){          DequeNode node = new DequeNode(item);          if(isEmpty()){              front = node;              rear = node;              node.rlink = null;              node.llink = null;          }else{              node.rlink = front;              node.llink = null;              front.llink = node;              front = node;          }      }        public char deleteFront(){          if(isEmpty()){              System.out.println("덱이 비어있습니다.");              return 0;          }else{              char data = front.data;              if(front.rlink==null){                  front = null;                  rear = null;              }else{                  front = front.rlink;                  front.llink = null;              }              return data;          }      }        public void insertRear(char item){          DequeNode node =  new DequeNode(item);          if(isEmpty()){              front = node;              rear = node;              node.rlink = null;              node.llink = null;          }else{              node.llink = rear;              node.rlink = null;              rear.rlink = node;              rear = node;          }      }        public char deleteRear(){          if(isEmpty()){              System.out.println("덱이 비어있습니다.");              return 0;          }else{              char data = rear.data;              if(rear.llink==null){                  front = null;                  rear = null;              }else{                  rear = rear.llink;                  rear.rlink = null;              }              return data;          }      }        public void removeFront(){          if(isEmpty()){              System.out.println("덱이 비어있습니다.");          }else{              if(front.rlink==null){                  front = null;                  rear = null;              }else{                  front = front.rlink;                  front.llink = null;              }          }      }        public void removeRear(){          if(isEmpty()){              System.out.println("덱이 비어있습니다.");          }else{              if(rear.llink==null){                  front = null;                  rear = null;              }else{                  rear = rear.llink;                  rear.rlink = null;              }          }      }        public char peekRear(){          if(isEmpty()){              System.out.println("덱이 비어있습니다.");              return 0;          }else{              return rear.data;          }      }        public char peekFront(){          if(isEmpty()){              System.out.println("덱이 비어있습니다.");              return 0;          }else{              return front.data;          }      }        public void print(){          if(isEmpty()){              System.out.println("덱이 비어있습니다.");          }else{              DequeNode node = front;              while(node!=null){                  System.out.print(node.data + " ");                  node = node.rlink;              }              System.out.println();          }      }  } |

1. 스택 (Stack) 에 대해 설명하고 자바로 구현 하시오

* 스택은 시간순서에 따라 자료가 쌓이고, 삭제할 때는 가장 마지막에 삽입된 자료가 가장 먼저 삭제되는 **후입선출** (LIFO)의 구조

|  |
| --- |
| **StackNode.java**    package Stack;    public class StackNode {      char item;      StackNode next;  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | **LinkedStack.java**    package Stack;    public class LinkedStack implements Stack{      StackNode top;        public LinkedStack() {          this.top = null;      }        @Override      public boolean isEmpty() {          return (top==null);      }        @Override      public void push(char item) {          StackNode node = new StackNode();          node.item = item;          node.next = top;          top = node;      }        @Override      public char pop() {          if(isEmpty()){              System.out.println("스택이 비어있습니다.");              return 0;          }else{              StackNode node = top;              top = node.next;              return node.item;          }      }        @Override      public void delete() {          if(isEmpty()){              System.out.println("스택이 비어있습니다.");          }else{              top = top.next;          }      }        @Override      public char peek() {          if(isEmpty()){              System.out.println("스택이 비어있습니다.");              return 0;          }else{              return top.item;          }      }        public void printStack() {          if(isEmpty()){              System.out.println("스택이 비어있습니다.");          }else{              StackNode node = top;              System.out.println("<<Stack>>");              while(node.next!=null){                  System.out.println(node.item);                  node = node.next;              }              System.out.println(node.item);              System.out.println();          }      }  } |

1. 자료구조 트리 에 대해 설명하시오

* 비선형 자료구조 중에서 자료들 간에 계층관계를 가진 계층형 자료구조(Hierarchical Data Structure)이다.

1. 자료구조 트리 를 자바로 구현하시오

**TreeNode.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | package Tree;    public class TreeNode {      Object data;      TreeNode left;      TreeNode right;        public TreeNode(Object data) {          this.data = data;          this.left = null;          this.right = null;      }        public Object getData(){          return this.data;      }  } |

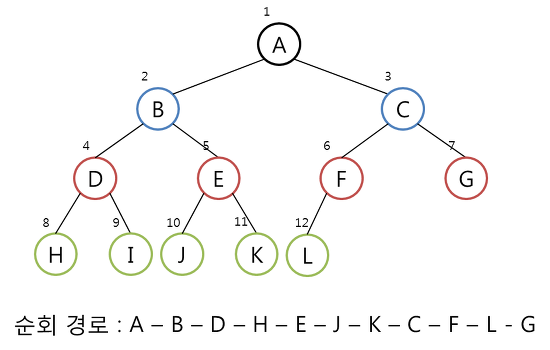
**LinkedTree.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package Tree;    public class LinkedTree {      private TreeNode root;        public TreeNode makeBT(TreeNode bt1, Object data, TreeNode bt2){          TreeNode root = new TreeNode(data);          root.left = bt1;          root.right = bt2;          return root;      }        public void preorder(TreeNode root){          if(root!=null){              System.out.print(root.data + " ");              preorder(root.left);              preorder(root.right);          }      }        public void inorder(TreeNode root){          if(root!=null){              inorder(root.left);              System.out.print(root.data + " ");              inorder(root.right);          }      }        public void postorder(TreeNode root){          if(root!=null){              postorder(root.left);              postorder(root.right);              System.out.print(root.data + " ");          }      }  } |

1. 이진 검색 트리에서 순회에 대해 설명 하시오

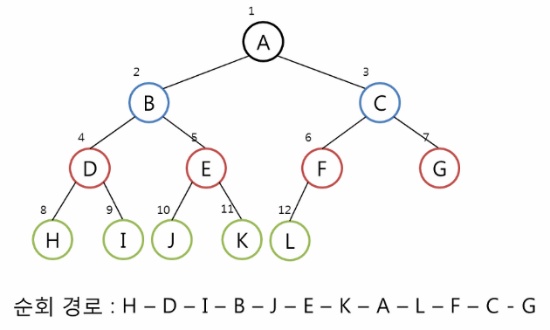
* **"순회(traversal)"**  이진 트리에 있는 모든 노드를 한번씩 모두 방문하여 노드가 가지고 있는 데이터를 처리하는 것
* **전위 순회 (Preorder Traversal)**

**"**현재 노드를 방문하여 **데이터**를 읽는 작업 **D"** 을 가장 먼저 수행하여 **DLR** 의 순서로 순회하는 방법



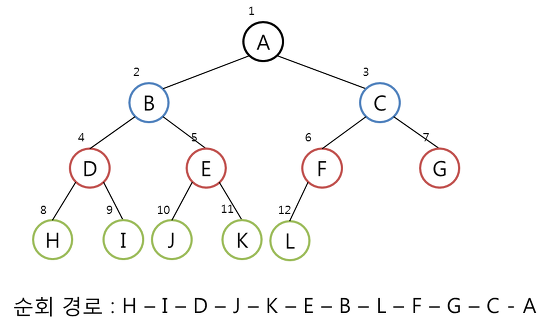
* **중위 순회 (Inorder Traversal)**

**"**현재 노드를 방문하여 **데이터**를 읽는 작업 **D"** 을 L과 R의 사이에 수행하여 **LDR** 의 순서로 순회하는 방법

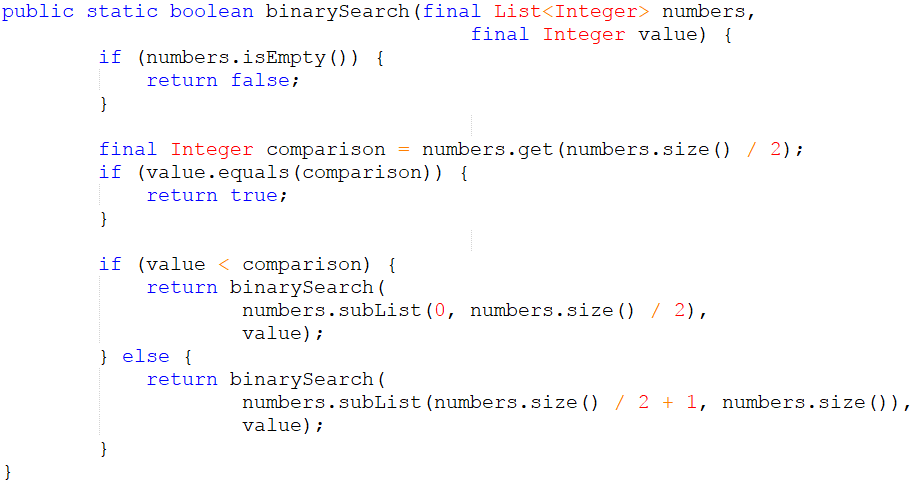


* **후위 순회 (Postorder Traversal)**

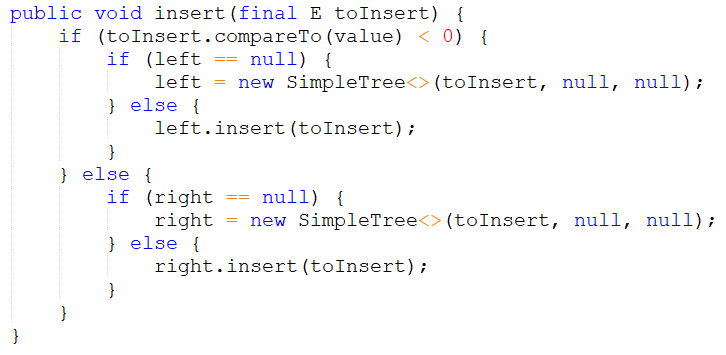
**"**현재 노드를 방문하여 **데이터**를 읽는 작업 **D"** 을 가장 나중에 수행하여 **LRD** 의 순서로 순회하는 방법



1. 이진 검색 트리에서 검색 알고리즘을 구현하시오



1. 이진 검색 트리에서 삽입 알고리즘을 구현하시오



1. 힙 에대해 설명하시오

* **완전 이진 트리**에 있는 노드 중에서

**키값이 가장 큰 노드나 키값이 가장 작은 노드**를 찾기 위해

정렬되어 있는 자료구조

최대 힙 (Max Heap) 키값이 가장 큰 노드를 찾기 위한 힙 (부모노드의 키값 >= 자식노드의 키값)

최소 힙 (Min Heap) 키값이 가장 작은 노드를 찾기 위한 힙 (부모노드의 키값 <= 자식노드의 키값)

1. AVL 트리 에 대해 설명하시오

* 모든 노드의 왼쪽과 오른쪽 서브트리의 높이 차이가 1 이하인 완전 이진 트리

1. 자료구조 맵 에 대해 설명하시오

* 데이터를 저장할 때 , Key 와 Value 가 한쌍으로 저장되는 자료구조
* 순서는 유지되지 않고 ,

키는 중복을 허용하지 않으면서 값은 중복을 허용

1. 자료구조 맵을 자바로 구현 하시오

* **public interface** Map<K,V> {**int** size();  
  **boolean** isEmpty();  
  **boolean** containsKey(Object key);  
  **boolean** containsValue(Object value);  
  V get(Object key);  
  V put(K key, V value);  
  V remove(Object key);  
  **void** putAll(Map<? **extends** K, ? **extends** V> m);

}

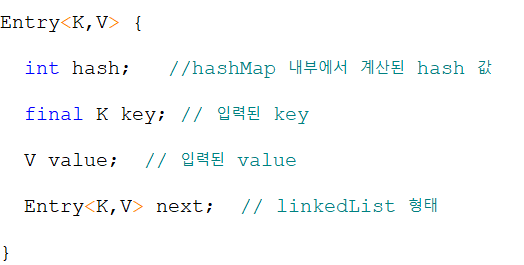
1. 자료구조 맵을 구현한 자바 클래스 종류를 나열하시오

* HashMap(HashTable), TreeMap ,

LinkedHashMap , CocurrentHashMap

1. 자료구조 맵을 구현한 HashMap 에 대해 설명하시오

* 키(Key) 와 값(Value) 을 하나의 쌍으로 가지고 있는 LinkedList Entry 객체 들을 담고 있는 Associate Array



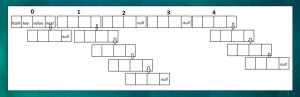
1. Entry 로 구성되어 있는 Associate Array

Entry<K,V> [] table;

1. HashMap 의 주요 알고리즘을 설명하라

* **1. put**

put() 메소드를 통해 key, value 값이 저장되면 내부적으로 아래와 같은 모습으로 저장된다.

[](https://minchangdev.files.wordpress.com/2014/01/ec8aa4ed81aceba6b0ec83b7-2013-10-09-ec98a4ed9b84-9-47-40.png)

이미지 출처 : <https://mkbansal.wordpress.com/2010/06/24/hashmap-how-it-works/>

저장되는 순서는 아래와 같다.

1) key 객체의 hashCode() 값을 이용하여 내부적으로 새로운 hash 값을 구한다.

2) 새로운 hash 값을 이용하여 table 배열에서 몇번째 index 에 저장할지 index 를 찾는다. ( indexFor 함수 )

3-1) 구해진 index 에 entry 가 이미 존재하면 linked-list 형태로 연결한다.

3-2) 구해진 index 에 entry 가 없으면 그냥 assign 한다.

4) 만약 key 가 기존에 존재하던 값이면 value 만 replace 하고 이전 value 값을 리턴한다.

**2. doubling**

위에서도 볼 수 있듯이 hashMap 에서는 entry 를 저장할 수 있는 table 배열의 크기가 정해져 있기 때문에 많은 수의 entry 가 입력되면 collision 이 자주 발생하게 되고 이는 성능저하를 가져오게 된다. 이를 방지하기 위해서 HashMap 은 table 배열의 사이즈를 doubling 하게 된다. ( 배열의 크기는 항상 2의 제곱수를 유지한다 )

1) entry 저장 시 entry 개수가 threshold ( table.size \* loadFactor ) 를 넘어서면 table 을 doubling

2) table 사이즈가 커졌기 때문에 기존에 저장된 모든 entry 에 대해서  rehash 를 수행하여 entry 분산

3) table.size 가 maximum\_capacity 를 넘어가면 doubling 하지 않음.

doubling 은 기존 배열의 두배 크기의 배열을 새롭게 만들고 기존의 entry 를 rehash 하며 옮겨오는 방식으로 진행된다.

**3. indexFor**

앞서 설명했듯이 내부적으로 새롭게 계산된 hash 값을 이용하여 table 에 저장된 index 값을 구하게 되는데 이를 위해 indexFor 함수를 호출한다. 일반적으로 table size 만큼 entry 를 분산해야 하기 때문에 modular 방식을 생각할 수 있겠으나 좀 더 효율적인 분산을 위해 아래와 같은 방법으로 구현되어 있다.

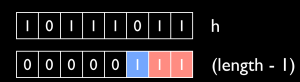
static int indexFor ( int h, int length ){

return h & ( length -1 );

}

h 는 hash 값, length 는 table 배열의 size 이다.

table 의 length 는 2의 제곱수이기 때문에 length – 1 은 모든 비트가 1로 채워진 수가 된다. 그 값과 h 값을 & 연산을 통해서 index 값을 구하게 된다. 이런 방식을 취하면 length 값이 두배로 증가하였을때는 상위 하나의 비트가 더 추가된 값과 h 값이 & 연산을 하게 된다. 그 결과는 반드시 기존 index 값과 같거나 doubling 으로 새롭게 추가된 영역의 index 만을 가지게 되기 때문에 좀 더 효율적인 entry 의 분산을 기대할 수 있다.

[](https://minchangdev.files.wordpress.com/2014/01/ec8aa4ed81aceba6b0ec83b7-2013-10-30-ec98a4eca084-12-29-01.png)

**4. ConcurrentModificationException**

HashMap 은 thread-safe 한 자료구조가 아니다. 즉, 특정 thread 가 자료를 읽는 중에도 다른 thread 가 map 내부의 자료를 삭제할 수 있다. 이는 multi-thread 환경에서 분명 문제가 된다. 이같은 문제 때문에 HashMap 에서는 entry 들을 iteration 하는 도중 자료구조에 변경이 일어나면 ConcurrentModificationException 을 발생시켜 해당 문제를 알려준다.

이를 위해서 HashMap 내부에 modCount 라는 변수를 두고 HashMap 의 모든 업데이트의 발생시 modCount 를 증가시켜 준다. 그리고 entry 의 iteration 을 시작하기 전에 modCount 값을 다른 임시 변수에 복사 해 두고 매번 next 호출때 마다 두 변수 값을 비교하여 차이가 발생하면 ConcurrentModificationException 을 발생시킨다

**5. get**

인자로 받은 key 에 매핑되는 value 값을 찾는 방식은 put 의 방식과 유사하다.

1) key 객체의 hashCode() 값을 이용하여 내부적으로 새로운 hash 값을 구한다.

2) 새로운 hash 값을 이용하여 table 배열에서 몇번째 index 에 저장되어 있는지 찾는다.  ( indexFor 함수 )

3-1) 구해진 index 에 entry 의 key 와 동일한 객체인지 equals 를 이용하여 비교하여 동일하면 entry 의 value 를 리턴한다.

3-2) equals 결과가 false 이면 entry 의 next 링크를 계속 따라가며 동일한 key 를 가진 entry 를 찾는다.

출처 : <https://mkbansal.wordpress.com/2010/06/24/hashmap-how-it-works/>

1. HashTable 과 HashMap 의 차이점은 무엇인가

* HashMap은 보조해시함수(Additional Hash Function)를 사용하기 때문에 그렇지 않은 HashTable에 비해 충돌 (hash collision)이 덜 발생하여 성능상의 이점이 있다.
* HashTable은 synchronized되어 있다.

(여기에 또한 성능상의 패널티 있다)

* HashTable은 key와 value에 null을 허용하지 않는다.

1. 자료구조 맵을 구현한 TreeMap 에 대해 설명하시오

* 이진 트리 구조를 이용하여 Map 인터페이스를 구현한 클래스
* TreeMap 은 크게 이진 트리로 구성되어 있고

각각의 노드는 비교가 가능한 Comparable 혹은 Comparator 인터페이스 구현체 이다

1. HashMap 과 TreeMap 간의 차이점은 무엇인가

* HashMap 과 다르게 TreeMap 이진트리로 구성되어 있기 때문에 각각의 엔트리 마다 순서가 있다

1. 자료구조 맵을 구현한 LinkedHashMap 에 대해 설명하시오

* HashMap 을 상속받은 클래스 이면서 동시에 각각의 Entry 가

쌍방향 리스트로서 연결된 자료구조

1. 자료구조 맵을 구현한 ConcurrentHashMap 에 대해 설명하시오

* HashMap 이 갖고 있는 associate array 의 요소마다 동기화를 설정하는 맵

HashTable 에서 ThreadSafe 를 위해 Associate Array 전체를 동기화(Synchronized) 를 설정 함 으로서 유발되는 성능저하를 수정하기 위해 , array 전체를 여러 개의 Segmenant 를 나누어서 각각에 Segment 에 동기화를 설정하는 맵

1. **What is the need of ConcurrentHashMap when there is HashMap and Hashtable already present?**

**Performance and Thread safety are 2 parameter on which ConcurrentHashMap is focused.**

**Imagine a scenario where we have frequent reads(get) and less writes(put) and need thread safety**

**Can we use Hashtable in this scenario?**

* No. Hashtableis thread safe but give poor performance in case of multiple thread reading from hashtable because all methods of Hashtable including get() method is synchronized and due to which invokation to any method has to wait until any other thread working on hashtable complete its operation(get, put etc).

1. **Can we use HashMap in this scenario?**

* No. Hashmap will solve performance issue by giving parallel access to multiple threads reading hashmap simultaneously.  
  But Hashmap is not thread safe, so what will happen if one thread tries to put data and requires Rehashing and at same time other thread tries to read data from hashmap, It will go in infinite loop.  
  **Infinite loop problem discussed in detail:**[**Infinite loop in HashMap**](http://javabypatel.blogspot.in/2016/01/infinite-loop-in-hashmap.html)

출처 :

<http://javabypatel.blogspot.kr/2016/09/concurrenthashmap-interview-questions.html>

1. 자료구조 집합 에 대해 설명하시오

* 중복을 허용하지 않는 순서없는 객체들의 모음

1. 자료구조 집합 을 구현한 자바 클래스 종류를 나열하시오

* HashSet , TreeSet , LinkedHashSet

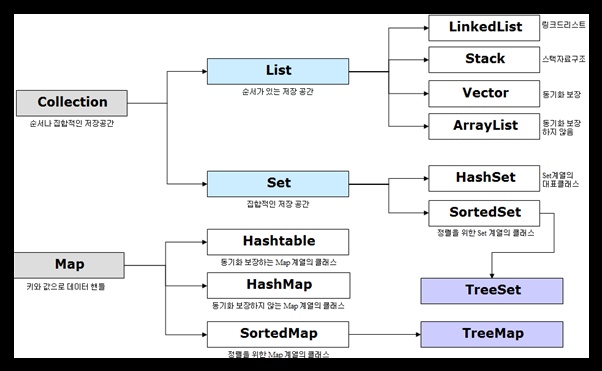
HashSet : 빠른 접근속도 , 중복 허용 X , 순서 X

LinkedHashSet : 중복 허용 X , 삽입순서에 의해 반복

TreeSet : 중복 허용 X , 정렬된 순서에 의해 반복

1. JCF 란 무엇인가 ?

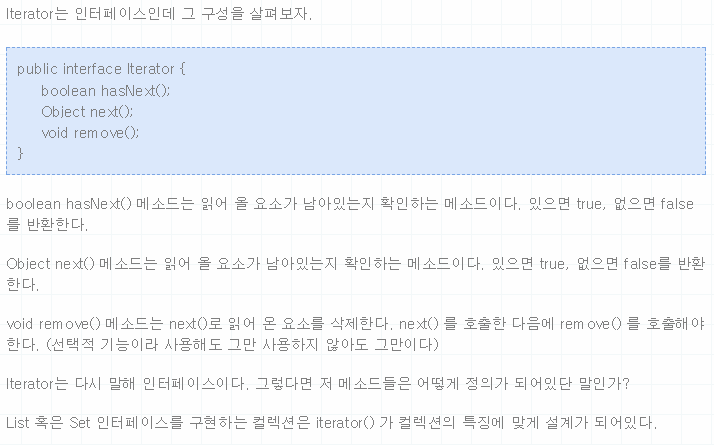
* Java에서 데이터를 저장하는 기본적인 자료구조들을 한 곳에 모아 관리하고 편하게 사용하기 위해서 제공하는 것을 의미



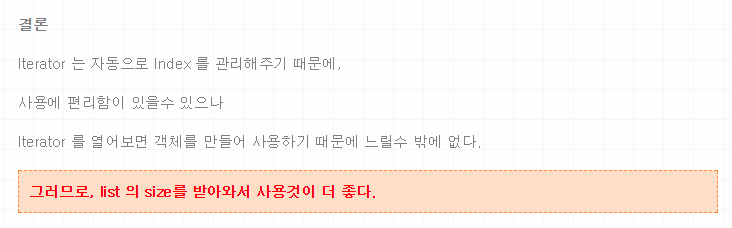
참고자료 : <http://withwani.tistory.com/150>

1. Iterator 란 ?

* Iterator는 자바의 컬렉션 프레임웍에서 컬렉션에 저장되어 있는 요소들을 읽어오는 방법을 표중화 하였는데 그 중 하나가 Iterator이다







* 알고리즘

1. 빅오표기법 (Big O) 에 대해서 설명 하시오

* Big O  표기법은 컴퓨터 공학에서 알고리즘의 복잡도 또는 성능을 표현하기 위해 사용된다.

Big O는 특히 최악의 경우를 표현하며, 알고리즘의 실행시간이나, 사용 메모리 (메모리, 또는 디스크) 공간을 표현하기도 한다

1. 빅요표기법을 주로 사용하는 이유는 뭔가

* 아무리 최악의 상황이라도 이정도의 성능을 보장할 수 있다'라는 것을 보여주기 위해 O표기법을 사용하는 것이다

1. 빅오분석법 적용 방법은 무엇인가

* 1. 입력값 기준을 정한다

2. 수행해야할 연산의 횟수를 n 의 식으로 표현한다

3. 차수가 제일 높은 항 (최고차항)만 남기고 , 모든 상수 인수를 없앤다

1. 정렬을 위한 인터페이스 Comparable 과 Comparator 의 차이점은 ?

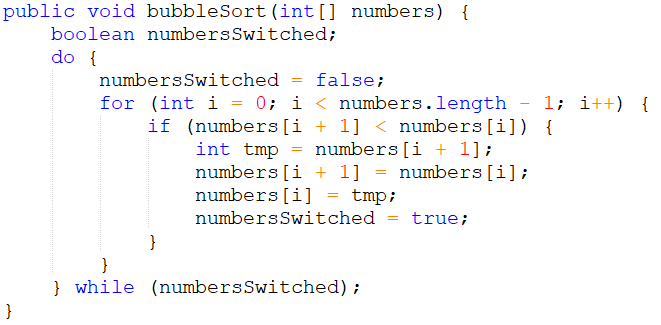
* Comparable 은 비교 가능한 객체로서 기능을 확장하고자 할 때 쓰임

Comparator 은 서로 다른 두 개의 객체를 비교하는 주체로서의 객체로서 쓰임

1. 버블 정렬 알고리즘의 핵심은 무엇인가

* 리스트의 인접하는 항목까지 비교후에 Swap 연산을 전체로 확대하는 연산

1. 버블 정렬 알고리즘을 자바로 구현하시오



1. 버블 정렬 알고리즘의 최악의 경우와 최선의 경우에서의 성능은 ?

* 역순으로 정렬되어 있는 리스트를 정렬할 때 최악 : O(n^2)

정순으로 정렬되어 있는 리스트를 정렬할 때 최선 : O(n)

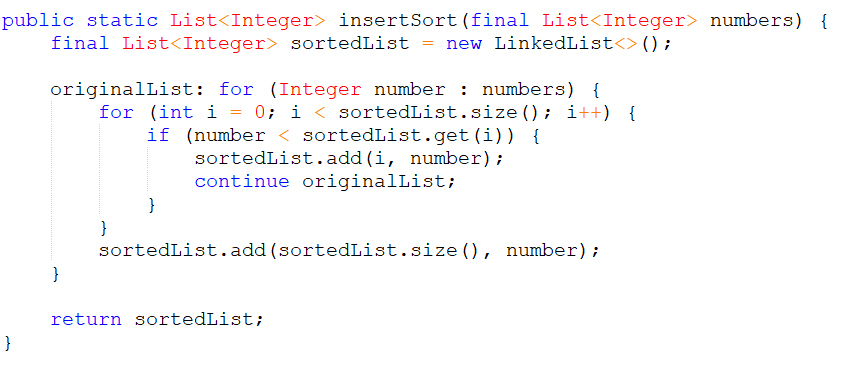
1. 삽입 정렬 알고리즘의 핵심은 무엇인가

* 새로운 정렬 리스트를 생성하고

새로운 리스트의 항목들과 비교해가며

새로운 리스트에 알맞은 위치에 삽입하는 정렬

1. 삽입 정렬 알고리즘을 자바로 구현하시오



1. 삽입 정렬 알고리즘의 최악의 경우와 최선의 경우에서의 성능은 ?

* 역순으로 정렬되어 있는 리스트를 정렬할 때 최악 : O(n)

정순으로 정렬되어 있는 리스트를 정렬할 때 최선 : O(n^2)

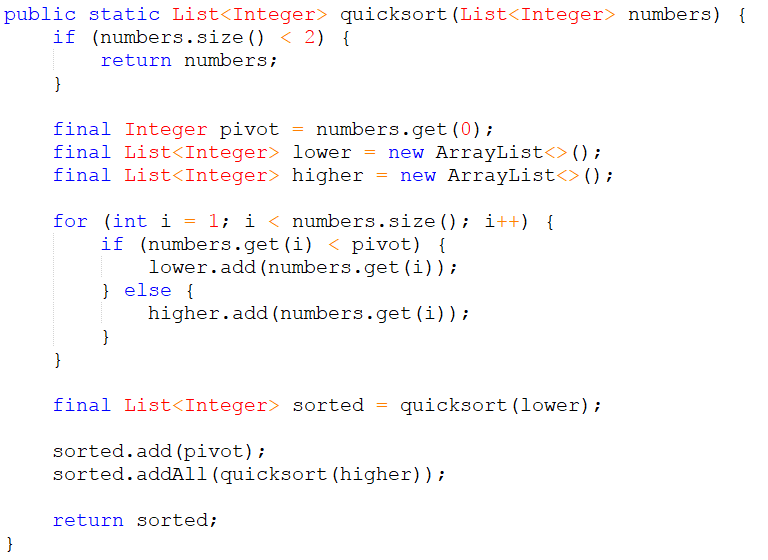
* 삽입정렬알고리즘은 새로운 리스트를 반환하므로 , 정렬하려는 리스트의 두배정도 되는 저장공간이 필요하다

1. 퀵 정렬 알고리즘의 핵심은 무엇인가

* 리스트내의 임의의 원소 (Pivot) 을 기준으로 리스트를 나누고

나누어진 리스트에 대해 재귀연산 을 시행

1. 퀵 정렬 알고리즘을 자바로 구현하시오



1. 퀵 정렬 알고리즘의 최악의 경우와 최선의 경우에서의 성능은 ?

* 역순으로 정렬되어 있는 리스트를 정렬할 때 최악 : O(n^2)

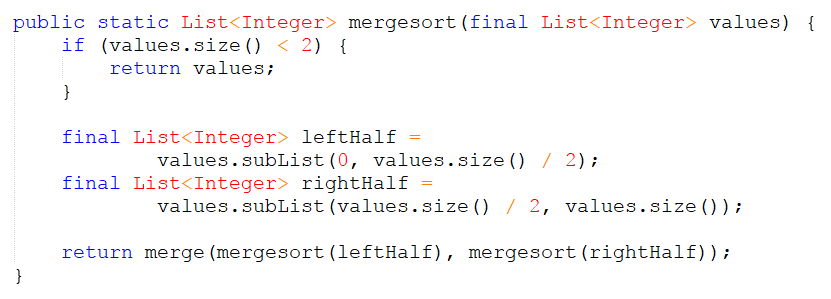
정순으로 정렬되어 있는 리스트를 정렬할 때 최선 : O(n \* logn)

1. 병합 정렬 알고리즘의 핵심은 무엇인가

* 재귀호출을 이용하여 ,

비교정렬이 가능한 최소단위의 리스트 까지 쪼갠 후에 비교정렬

1. 병합 정렬 알고리즘을 자바로 구현하시오





1. 병합 정렬 알고리즘의 최악의 경우와 최선의 경우에서의 성능은 ?

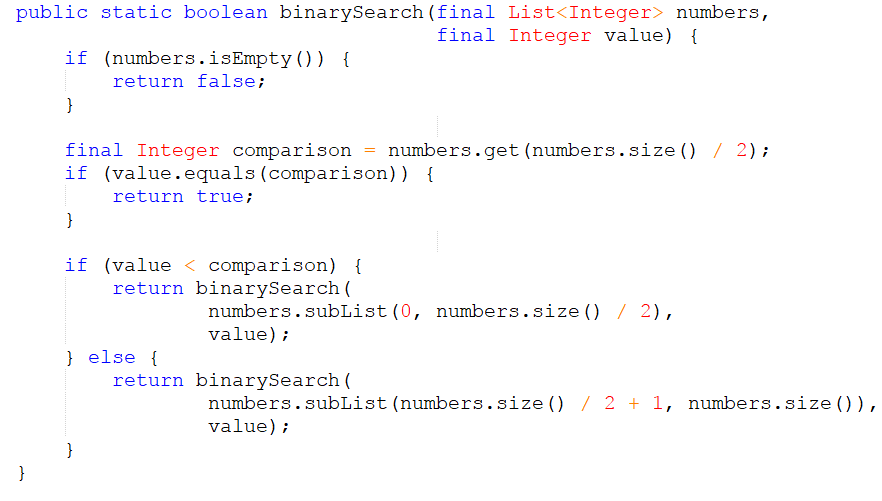
* O ( n \* log n ) , 각각의 병합시간은 O (n) 이며

각각의 재귀호출은 주어진 리스트의 절반만큼 발생한다

1. 이진 검색 알고리즘의 핵심은 무엇인가 ?

* 정렬된 리스트에서 중간값을 비교하여 검색범위를 절반으로 줄이면서 검색

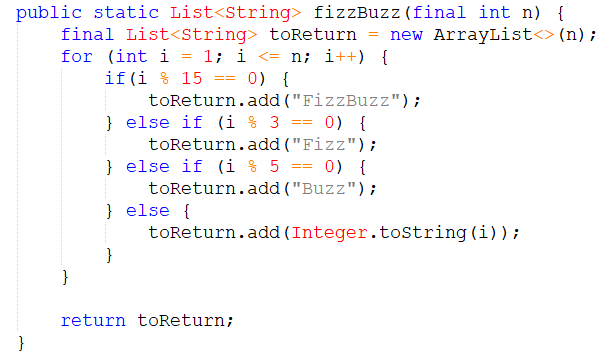
1. 이진 검색 알고리즘을 자바로 구현하시오

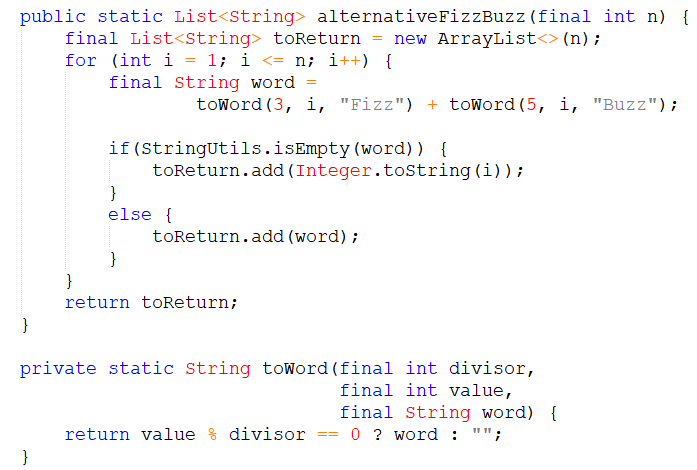


1. 이진 검색 알고리즘의 성능은 ?

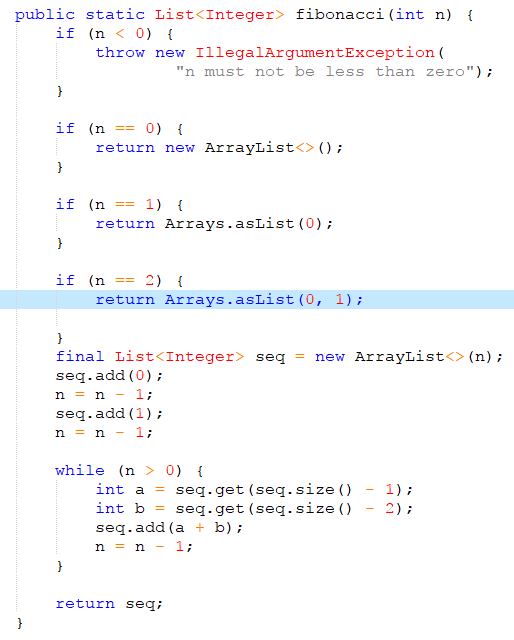
* O(n)

1. 1 에서 n 까지의 숫자를 출력 하되 , 3의 배수는 Fizz 라는 문자열을 출력하고 5의 배수는 Buzz 라는 문자열을 출력하고 , 15의 배수는 FizzBuzz 라는 문자열을 출력하는 알고리즘을 자바로 구현하라

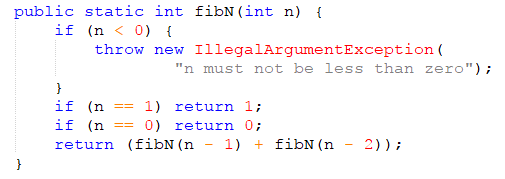




1. 1 에서 n 까지의 피보나치 수열을 반환하는 알고리즘을 구현하라

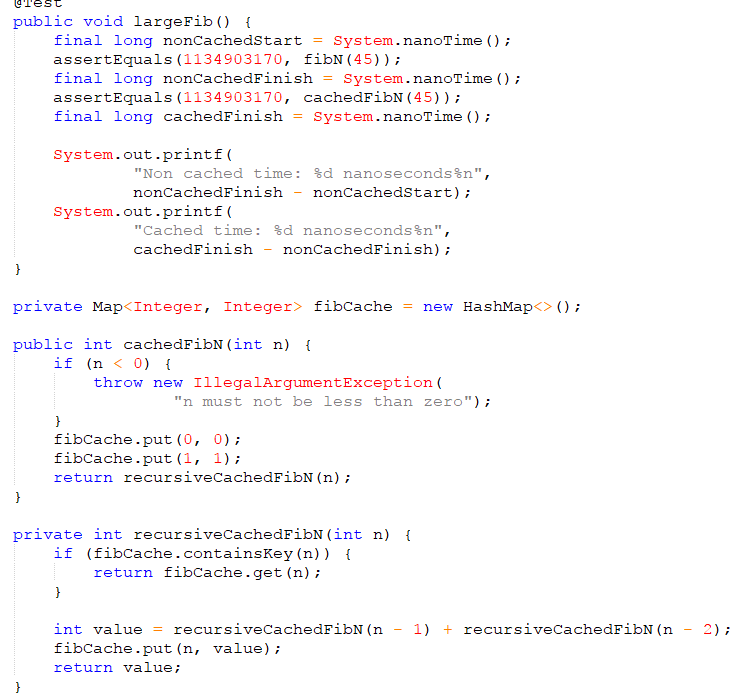


1. 피보나치 수열의 n번째 값을 반환하는 알고리즘을 구현하라

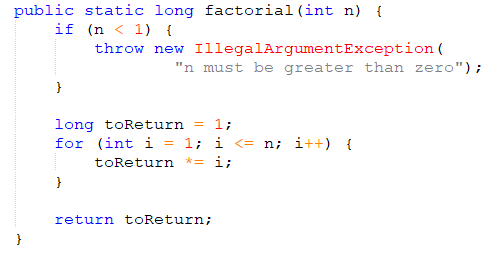


1. 피보나치 수열의 n번째 값을 반환하는 알고리즘을 구현하라

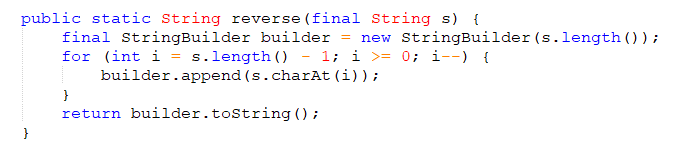
(메모제이션 을 이용)

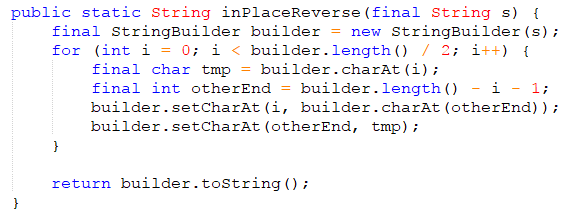


1. 팩토리얼 을 자바로 구현하라

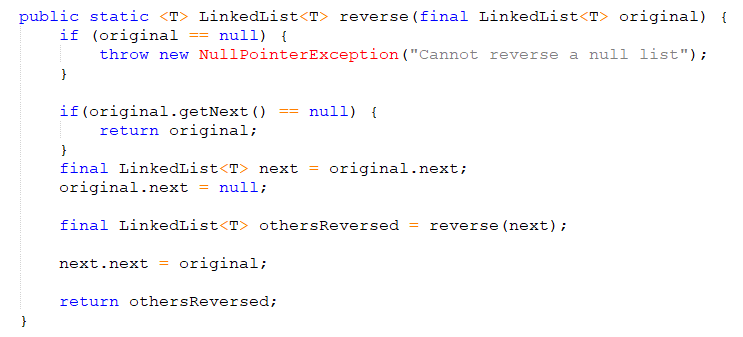


1. 문자열을 뒤집는 알고리즘을 자바로 구현하라

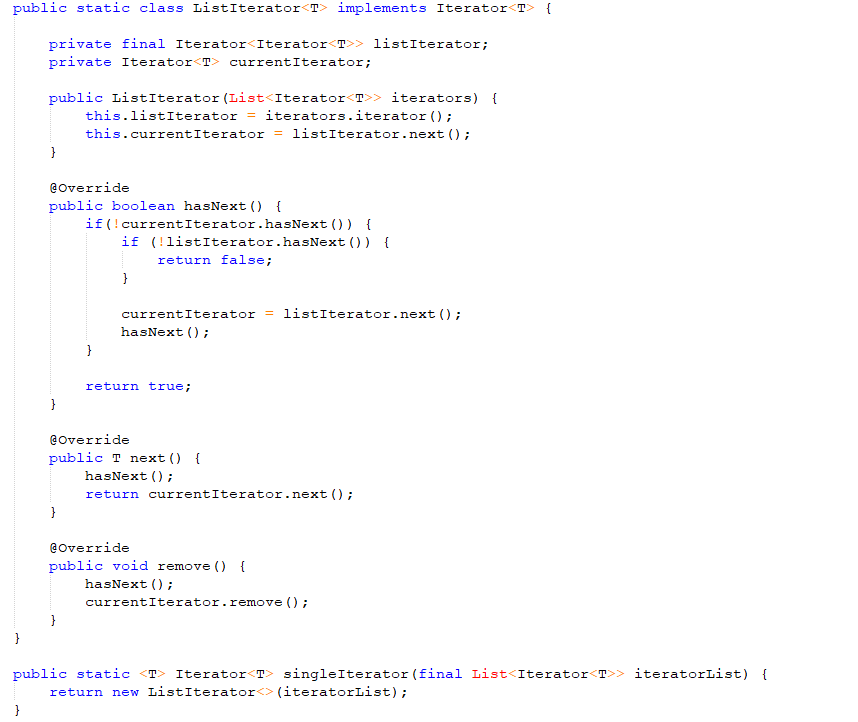




1. 연결 리스트를 뒤집는 알고리즘을 자바로 구현하라



1. 여러 개의 Iterator 인터페이스의 리스트가 있을 때 , 하나의 Iterator 인터페이스로 관리 하는 알고리즘을 작성하라



* 개발방법론

1. TDD 는 무엇인가 ?
2. ABTEST 는 무엇인가 ?
3. Dynamic 프로그래밍은 무엇인가 ?

* 자바 기본

1. 자바에서 객체란 무엇인가 ?

* 객체는 참조타입
* 개체 (Entity) 를 나타내는 변수와 메소드 로 구성되어 있는 컬렉션

1. 자바에서 참조타입 이란 무엇인가 ?

* 힙 에 할당되어 있는 메모리 주소를 값으로 가지고 있는 변수
* 자바에서는 원시타입을 제외한 모든 변수를 일컫는다

1. 인터프리터 와 컴파일러는 무엇인가 ?

* 인터프리터 :

* 컴파일러 :



1. JVM 은 무엇인가 ?

* 자바로 작성된 코드를 메모리에 적재하고 실행하는 주체

1. JVM 은 어떻게 구성되어 있는가 ?

|  |  |
| --- | --- |
| class loader | JVM내로 class를 로드하고 Link를 통해 적절히 배치하는 일련의 작업을 수행하는 모듈.  Runtime 시 동적으로 class load |
| Execution Engine | class loader를 통해 JVM 내의 런타임 데이터 영역에 배치된 바이트 코드는 실행 엔진에 의해 실행, 실행 엔진은 자바 바이트 코드를 명령어 단위로 읽어서 실행 |
| Garbage Collector | JVM은 Garbage Collector를 통해 자동화된 메모리리 관리 기능을 제공.  Garbage Collector는 Garbage Collection을 수행하는 모듈 쓰레드를 말한다. Application이 생성한 객체의 생존 여부를 판단해서 더 이상 사용되지 않는 객체를 해제함으로써 메모리 관리가 자동화되도록 한다.  줄여서 GC라고 부른다. |
| Runtime Data Areas | JVM이라는 프로그램이 운영체제 위에서 실행되면서 할당받는 메모리 영역 |

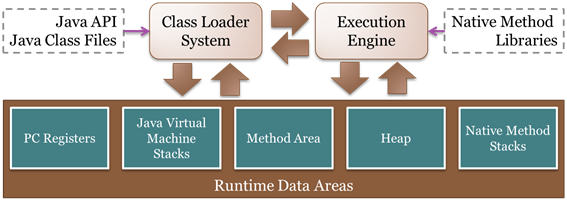
1. JVM 수행 과정은 어떻게 되는가

1. Class Loader System을 통해 Class파일들을 JVM으로 로딩한다.

2. 로딩된 Class 파일들을 Execution Engine을 통해 해석한다.

3. 해석된 프로그램은 Runtime Data Areas에 배치되어 실질적인 수행이 이루어지게 된다. 이러한 실행 과정 속에서 JVM은 필요에 따라 Thread Synchronization과 Garbage Collection 같은 관리작업을 수행하게 된다.

1. Runtime Data Area (JVM 메모리 구조) 는 어떻게 되는가



1. 클래스 영역(Method Area , Static Area)에 대해서 설명하시오

* 모든 스레드가 공유하는 메모리 영역
* JVM 의 클래스 로더에 의해 클래스가 로드되면서 메모리에 가장 먼저 적재되는 영역

적재되는 메모리는 아래와 같다

1. 런타임 상수 풀 (Runtime constant pool)

클래스 파일마다 사용된 상수를 저장하는 곳

문자 상수, 타입, 필드, Method의 symbolic reference (static method)

1. 클래스 변수 (Static 변수)

클래스 에서 선언된 static 변수

1. 필드 정보 (Field Information)

멤버변수의 이름, 데이터 타입, 접근 제어자에 대한 정보

1. 메소드 정보 (Method Information)

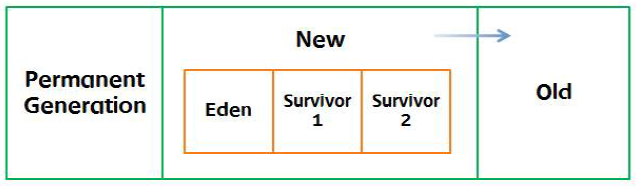
메서드의 이름, 리턴타입, 매개변수, 접근제어자에 대한 정보

1. 타입 정보 (Type Information)

클래스 파일의 전체 경로 (패키지 + 클래스 파일명)

클래스 파일의 Super Class 정보

1. 힙 영역 에 대해 설명하시오



1. Permanent Generation (PermGen)

* 클래스에 대한 정의
* Immutable 변수 (ex, String Pool)

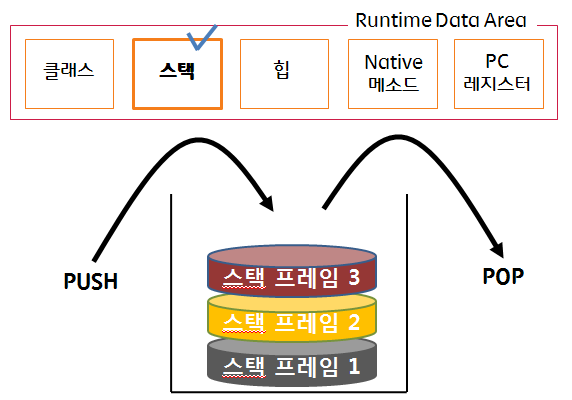
1. New Generation

* Eden : 최초로 생성된 객체가 할당되는 공간
* Survivor : Eden 에 있다가 GC 대상에서 제외된 객체 공간

1. Old Generation

GC 에 자주 수집 당하지 않는 (강한 참조)의 객체들의 공간

1. 스택 영역에 대해 설명하시오



* 각각의 메소드 마다 호출시 생성 되어지는 메모리 프레임
* 호출된 메소드 안에서 사용되어지는 값들 저장
* 호출된 메소드의 매개변수 , 지역변수 및 연산에서 사용하는 임시변수
* 호출된 메소드가 종료되면 메모리에서 해제되며 , 호출된 순서가 빠른 순서로 진행 (LIFO)

1. **Native method Stack 영역에 대해 설명하시오**

* 자바 외의 다른 언어에서 제공되는 메서드들이 저장되는 공간

1. **PC Register 영역에 대해 설명하시오**

* Thread 가 생성 될 때마다 생성되는 공간
* Thread 가 어떤 명령으로 실행할 지에 대한 기록
* 현재 실행되는 부분의 명령과 주소를 저장

1. 자바에서 원시타입 이란 무엇인가 ?

* 참조타입과 다르게 ,

실제 값을 전달하고 스택 영역에서 생성되고 해제된다

1. 자바의 원시타입의 종류가 각각이 차지하는 크기를 서술하라

|  |  |  |  |
| --- | --- | --- | --- |
| 데이터형 | 예약어 | 비트수 | 범위 |
| ①논리형 | boolean | 8bit  (1바이트) | true, false |
| ②문자형 | char | 16bit 유니코드 (2바이트) | 수치로는 0 ~ 65535('\u0000'~'\uFFFF')  유니코드:영어,숫자-1바이트, 그외 다국어-2바이트 |
| ③수치형 (정수) | byte | 8bit (1바이트) | -128 ~ 127(-2의7제곱~2의7제곱-1) 개수:2의8제곱 |
| ④수치형 (정수) | short | 16bit (2바이트) | -32,768 ~ 32,767(-2의15제곱~2의15제곱-1) 개수:2의16제곱 |
| ⑤수치형 (정수) | int | 32bit (4바이트) | -2,147,483,648 ~ 2,147,483,647(-2의31제곱~2의31제곱-1) 개수:2의32제곱 |
| ⑥수치형 (정수) | long | 64bit (8바이트) | -9,223,372,036,854,775,808~9,223,372,036,854,775,807 (-2의63제곱~2의63제곱-1) 개수:2의64제곱 |
| ⑦수치형 (실수형) | float | 32bit (4바이트) | ±3.40282347E+38, ±1.40239846E-45, IEEE 754-1985표준 |
| ⑧수치형 (실수형) | double | 64bit (8바이트) | -1.79769313486231570E308~+1.79769313486231570E308 |

1. 자바의 원시타입이 JVM 에서 어떻게 처리되는가

* 원시타입은 비객체형 타입이다
* 원시타입은 항상 값이 존재한다

1. 보수 란 무엇인가

* '두 수의 합이 진법의 밑수(N)가 되게 하는 수'

Cf ,

10 진수 4의 10의 보수 (-4)는 = 6

10 진수 6의 10의 보수 (-6)는 = 4

1. 보수 를 사용 하는 이유는 무엇인가

* 덧셈만 가능한 컴퓨터 연산에서 뺄셈이 가능하게 하기 위해 사용
* 1의보수 , 2의보수 모두 동일하게 음수를 나타낼수 있다

1의 보수 : 모든 비트를 NOT 연산으로 반전

2의 보수 : 1의 보수로 표현된 비트에 1을 더함

 - 1의 보수 뺄셈 : 1의 보수를 이용한 뺄셈은 빼는 수의 1의 보수를 구한 다음 더합니다. 덧셈한 결과가 최상위 비트에서 자리올림이 생겼다면 최하위 비트에 1을 더하고, 자리올림이 생기지 않았다면 연산 결과에 대해 1의 보수를 구한 후 - 부호를 붙입니다.

​

  예시 ) ​111(2) - 110(2) = 111(2) + 001(2) = 1000(2)에서 최상위 비트를 지우고 1을 더하므로 답은 1(2)입니다.

  예시 ) 100(2) - 110(2) = ​100(2) + 001(2) = 101(2)에서 다시 1의 보수를 취하므로 답은 10(2)입니다.

​

​ - 2의 보수 뺼셈 : 2의 보수를 이용한 뺄셈은 빼는 수의 2의 보수를 구한 다음 더합니다. 덧셈한 결과가 최상위 비트에서 자리 올림이 생겼다면 자리올림을 제외한 나머지 부분이 연산 결과이고, 자리 올림이 생기지 않았다면 연산 결과의 2의 보수를 구한 후 - 부호를 붙입니다.

​

  예시 ) 111(2) - 110(2) = 111(2) + 010(2) = 1001(2)에서 최상위 비트를 지우므로 답은 1(2)입니다.

  예시 ) 1​00(2) - 110(2) = 100(2) + 010(2) = 110(2)에서 답은 010(2)입니다.

1. 왜 Short , Integer , Long 는 2의보수를 사용하는가

* 1의보수를 사용하면 0 으로서 표현 가능한 가지수가 2개가 생김

0000 0000 = 0

1111 1111 = 0

따라서 , 2의보수를 사용하면 이 경우를 방지할 수 있음

1. 왜 Integer.MIN\_VALUE 에 해당하는 양수가 없는가

* Integer 는 8비트로 숫자를 표현하는데 , 최하로 작은숫자는

Singend Bit 를 제외하고는 모두 0 이기 때문에

양수로 표현할 수가 없다

양수로 표현하고자 한다면 ,

BigInteger 또는 Long 으로 표현 해야 한다

1. Integer.valueOf() 메소드가 효율적인 이유는 ?

* Integer 값을 재사용하기 위해 ,

플라이웨이트패턴으로 구현된 캐시참조 메소드 이기 때문이다

1. Immutable 변수와 mutable 변수 에 대해서 설명하시오

* Immutable 변수란

생성 후 값이 변경 불가능(=heap 영역에서 변경불가) 한 객체

대표적으로 String, Boolean, Integer, Float, Long

* Mutable 변수란

생성후 값이 변경 가능한 객체

1. Immutable 변수의 장단점 ?

* 멀티스레드 환경에서 동기화(Lock , Synchronized)를 설정할 필요가 없다
* 안정된 공유 , 재사용이 가능하기 때문에 Cache 를 사용하여

성능 향상 및 메모리 효율을 도모한다

* 복잡도가 감소한다
* 업데이트를 해야 하는경우에는(불가피하게 수정 해야하는 경우)

업데이트된 값을 가진 객체의 복사본을 생성 해야한다는 단점이 있다

1. “자바에 진짜 Immutable 은 없다” 는 무엇을 의미하는가

* Java 의 Reflection 을 이용하면 언제든지 변경 가능하다

1. Final 키워드는 객체 참조에 어떤 영향을 미치는가

* Final 변수

값을 수정할 수 없는 변수

* Final 메소드

오버라이딩이 불가능한 메소드

상속 받은 그대로 사용 해야함

* Final 클래스

상속이 불가능함

1. 객체의 가시성 수정자는 무엇이며 종류는 무엇인가

* Private : 같은 클래스내에서 접근가능 , 하위클래스 접근 불가능
* <none> : 같은 패키지의 모든 클래스에서 접근 가능
* Protected : 모든 하위 클래스에서 접근가능
* Public : 어디서든 접근가능

1. hashCode 메소드가 가지는 의미는 무엇인가

* 객체를 구별하기 위해 고유한 정수값으로 출력시켜주는 메소드

객체란 참조타입 변수를 의미하는데 , 이때의 참조값(주소)를 반환

* 두개의 객체가 같은 객체라면 반드시 같은 hashCode 값을 반환

하지만 ,

같은 hashCode 값을 가진 객체라고 해서 같은 객체라고 볼수는 없다

이는 동시에 , 객체가 같지 않더라도 hashCode 는 같을 수 있다

* 서로 다른 객체에 대해 항상 서로 다른 hashCode 값을 보장이 되면 ,

hashCode 값을 기반으로 한 Collection 리스트 객체 (ex, HashMap , HashTable , HashSet) 의 성능을 향상 할 수 있다

1. Equal 메소드가 가지는 의미는 무엇인가

* 두 객체의 참조가 논리적으로 같은지 확인하는데 사용
* String 클래스에서는 실제로 가지고 있는 값을 비교한다
* 일반적으로 두객체의 동일성을 확인하는데 보다는 , 메소드 오버라이딩의 동일성을 확인할필요가 있을때만 사용해야 함

1. 왜 “equals 메소드를 오버라이드 할 때는 hashCode 메소드도 항상 같이 오버라이드” 해야 하는가 ?

* “equls 메소드 값이 true 이면 두객체에 대한 hashCode 메소드값이 항상 동일 해야함”

이라는 보편적 계약사항을 위배하지 않기 위해 , hashCode 메소드 또한 오버라이드 (재정의) 해야한다

1. Static 키워드의 역할은 무엇인가

* 모든 인스턴스가 공통적으로 접근 및 사용 가능하게 해주는 키워드
* 클래스에 의해서 접근 가능하게 함

1. Volatile 키워드의 역할은 무엇인가

* 변수의 가시성을 보장하는 키워드

다수의 스레드가 같은 값의 변수로서 읽고,쓸수 있도록 보장

주의 !

volatile 변수 에 대한 쓰기/읽기 에 대해 발생할수 있는 변동사항에 대해 재배치는 보장해주지 않는다

따라서 ,

Synchronized 를 통해 , 연산의 원자성을 보장하는 것이 필요하다

출처 :

<http://thswave.github.io/java/2015/03/08/java-volatile.html>

1. Synchronized 키워드는 무엇인가

* 다수의 스레드에 의한 접근에서 동기화를 보장

1. Transient 키워드는 무엇인가

* 해당 변수는 직렬화 대상에서 제외하고자 할 때 쓰임

Serializable 로서 직렬화 되어 전송될 때

해당 변수는 null 로서 전달되어짐

1. 자바에서 다형성과 상속은 무엇을 의미하는가

* 다형성은 행동의 특정 타입에 대한 정의를 만들수 있게 하고 , 행동을 구현하는 수많은 다른 클래스를 갖게 한다
* 상속은 부모클래스에서 클래스의 행동과 정의를 가져다 사용할수 있게 해준다

1. 매소드 오버라이드 와 오버로딩의 차이점은 ?

* 오버로딩 : 같은 이름의 메소드를 여러 개 가지면서 매개변수의 유형과 개수를 다르도록 하는 기술
* 오버라이딩 : 상위 클래스가 가지고 있는 메소드를 하위 클래스가 재정의

1. @Override 어노테이션의 역할은 ?

* 컴파일러 에게 슈퍼 클래스의 메소드가 오버라이드 되었다는 것을 알려주는 역할

1. 자바에서 배열은 어떻게 표현하는가

* 자바에서 배열은 하나의 객체 (참조타입) 으로 다루어진다

1. String 은 메모리에 어떻게 저장되는가

* 쌍따옴표를 이용하여 문자열을 String 에 할당하면 String pool 에 있는 값을 참조하게 끔 한다

(String pool 에 값이 없으면 , 이값을 상수로서 추가)

* 생성자 함수로서 String 을 생성하면 Heap 영역에 객체(참조값) 추가

1. String Pool 이란 무엇인가 ?

* 문자열을 재사용 해서 메모리 사용량을 최소화 할수 있도록 만들어진 , 플라이 웨이트 패턴으로 구성된 저장공간

1. String intern() 메소드 (인터닝) 는 무엇인가 ?

* 새롭게 만들어지는 String 객체를 상수화 (String pool 에 추가)

1. String intern() 메소드 (인터닝) 의 장단점은 무엇인가 ?

* 같은 값을 가지는 문자열을 새로 중복해서 Heap 생성하지 않게 되므로 메모리를 절약 할수 있다
* String 객체를 하나 생성해서 String pool 에 해당하는 값이 있는지 없는지 여부를 비교하는 로직이 있으므로 시간 복잡도 증가

1. String 객체의 값은 변경할 수 있는가

* String 은 Immutable 변수이다 ,
* String 객체 에 다시 새로운 문자열을 주입하는 것은 값을 변경하는 것이 아닌 새로운 객체를 생성한 것과 같다

1. String 은 왜 Immutable 변수 인가 ?

* String Pool 에서 변경 가능한 변수로서 남아있으면 , 문자열 값이 변경되면 다른 참조에 잘못된 참조가 연결될수 있기 때문
* hashCode 값 무결성을 보장해주기 때문에 캐쉬화 할수 있다
* 문자열은 광범위하게 사용되는 변수로서 , 참조값을 통해 값을 변경이 가능하게 되면 여러가지 형태 조작이 가능하게 되므로 , 보안성을 유지 해줘야한다

출처:

<https://dzone.com/articles/why-string-immutable-java>

1. StringBuffer 와 StringBuilder 와의 차이점 ?

* StringBuffer 와 StringBuilder 둘다 mutable 변수이다
* StringBuffer 는 멀티스레드 환경에서 ThreadSafe 한 변수이지만

StringBuilder 는 그렇지 않다

따라서 성능은 StringBuilder 가 더 우수하지만

ThreadSafe 하지 않다는 단점이 있다

참고.

사실 JDK 1.5 버전 이전에서는 이전에서는 문자열 연산(+, concat)을 할 때에는 조합된 문자열을 새로운 메모리에 할당하여 참조함으로 인해서 성능상의 이슈가 있었습니다(위에서 설명한 String 클래스 동작 방식에 의한 이유). 그러나 JDK 1.5 버전 이후에는 컴파일 단계에서 String 객체를 사용하더라도 StringBuilder로 컴파일 되도록 변경되었습니다. 그리하여 JDK 1.5 이후 버전에서는 String 클래스를 활용해도 StringBuilder와 성능상으로 차이가 없어졌습니다.

참고:

<http://javacan.tistory.com/entry/39>

1. 제네릭이란 무엇인가

* 매개변수화된 타입

클래스 내부에서 사용할 데이터 타입을 나중에 인스턴스를 생성할 때 확정하는 것을 제네릭

* 타입의 안정성을 보장 , 형변환을 생략 하기 위한 기능

Cf , 원시타입은 제네릭으로 표현 할 수 없다

1. 자바에서 와일드카드는 무엇이며 어떠한 의미가 있는가

* 알수 없는 타입 , 모든 객체를 포함

1. 원시타입과 Wrapper 클래스 간의 차이점은 ?
2. 오토박싱/박싱 과 언박싱 은 무엇인가 ?

* 박싱 : JDK 1.5 이전 버전에서 원시타입을 참조타입(Wrapper 클래스) 로 형변환 해주는 것을 의미

참조타입(Wrapper 클래스)의 생성자 함수를 사용하거나 ,

valueOf() 메소드를 사용 혹은 강제 형변환 을 의미

* 오토박싱 : JDK 1.5 이후 버전에서 원시타입을 참조타입(Wrapper 클래스) 로의 형변환을 컴파일러가 자동으로 해주는 것을 의미
* 언박싱 : 참조타입(Wrapper 클래스) 를 원시타입으로 형변환 해주는 것을 의미

1. 어노테이션 이란 무엇인가

* JEE5 부터 새롭게 추가된 문법요소 로서 ,

자바코드에 주석으로 달려서 해당 코드가 컴파일러 타임 혹은 런타임에 의해 해석 될수 있도록 해주는 기능

1. 어노테이션 이 등장하게 된 배경 은 ?

* 프로그래머에게 그들의 코드에 대한 메타데이터를 자신의 코드에 직접적으로 기술할수있는것을 제공하기위함

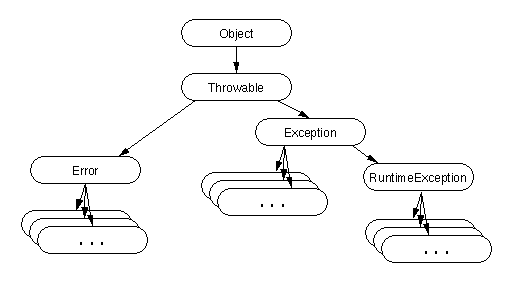
1. 어노테이션의 종류는 ?

* 빌트인 어노테이션 : 자바 SDK 에서 지원하는 어노테이션
* 커스텀 어노테이션 : 개발자가 임의의로 개발한 어노테이션

1. @Override 의 역할은 ?

* 컴파일러에게 슈퍼 클래스의 메소드가 오버라이드 되었다는 것을 알려준다

1. 자바의 예외 처리 구조를 이루는 주요 클래스를 설명하라

* 

1. ‘런타임 예외’ 와 ‘확인해야 하는 예외’ 중 어느것이 좋은가

* Runtime Exception : 컴파일시에는 알수 없는 예외
* Checked Exception : 컴파일시에 알수 있는 예외

1. 연쇄예외 란 무엇인가

* Catch 문 안에 새로운 예외를 추가하는 것

1. Try – with – resource 문은 무엇인가

* Try-with-catch 문과 함께 쓰일수 있으며 ,

try 문에 조건식으로 시작한다

1. 왜 private 필드가 변하지 않도록 final 키워드를 선언 해야하는가

* 리플렉션을 이용해서 값을 변경할수 없도록 하기 위해서

1. 자바8 에서 예상되는 새기능은 무엇인가

* 인터페이스의 기본 구현과 람다

1. 가비지 컬렉션이란 무엇인가

* 힙에 할당된 메모리를 재사용 하기 위한 매커니즘

1. 가비지 컬렉션의 알고리즘은 무엇인가

* 이와 같은 자바의 메모리 누수 현상에 대한 정확한 진단과 처방은 때로는 GC 알고리즘에 대한 보다 정확한 이해를 필요로 한다.  
  자바의 GC 알고리즘은 reference counting을 사용하지 않아서 객체간에 cyclic reference가 생겨도 GC되지 않는 문제가 없는 완벽한 방법이지만 일정 시간의 GC 시간을 필요로 한다는 단점을 가지고 있다.  
  즉, 자바 heap의 모든 객체는 현재 사용중인 객체와 사용되고 있지 않는 객체, 2가지로 나뉘어지며,  
  사용중인 객체중에서도 사실상 사용되지 않는 객체가 있을 수 있으며 이는 메모리 누수에 해당한다는 것이다.

현재 사용중인 객체란 다음과 같은 루트 참조 (객체가 아님)들로부터 직간접적으로 참조가 되는 (reachable한) 모든 객체를 의미하며, 나머지 객체는 모두 쓰레기 객체이고, 요즘 JVM은 이러한 쓰레기 객체를 완벽하게 수거하므로 (옛날 버전의 JVM은 그렇지 않았음) 이 단계에서의 메모리 누수는 없다.

이와 같은 루트 참조는 다음과 같이 크게 3가지가 존재한다.

1. static 변수에 의한 객체 참조  
2. 모든 현재 자바 스레드 스택내의 지역 변수, 매개 변수에 의한 객체 참조  
3. JNI 프로그램에 의해 동적으로 만들어지고 제거되는 JNI global 객체 참조

이를 직관적으로 이해하는 방법은 다음과 같다.  
GC 알고리즘에서 현재 사용중인 객체의 의미는 현재 생성된 객체들중에서 현재 이후에 참조되어 사용될 가능성이 있는 모든 객체를 의미한다.  
객체는 직접 참조되지 않고 항상 변수를 통하여 참조가 가능하다.  
static 변수는 프로그램 어디서든 사용할 수 있으므로 static 변수에 의해 참조되는 객체와 그 객체로부터 직간접적으로 참조되는 모든 객체는 언제든 사용될 가능성이 있는 객체라서 사용중인 객체이다.

자바에서 현재 실행중인 (각 스레드별로) 모든 메소드내에 선언된 지역 변수와 매개변수에 의해 참조되는 객체와  그 객체로부터 직간접적으로 참조되는 모든 객체는 참조되어 사용될 가능성이 있으며, 이 뿐만 아니라 caller 메소드로 return된 후에는 caller 메소드에서 참조하고 있는 지역변수, 매개변수에 의해 참조되는 객체와  그 객체로부터 직간접적으로 참조되는 모든 객체 또한, 참조되어 사용될 가능성이 있다.

따라서, 각 자바 스레드의 스택 프레임내에 있는 모든 지역변수와 매개 변수에 의해 참조되는 객체와 그 객체로부터 직간접적으로 참조되는 모든 객체들이 참조되어 사용될 가능성이 있다는 것이다.

또한, JNI 네이티브 C 함수내에서도 JNI 함수를 사용하여 자바 객체를 생성할 수 있다.

이때 생성된 자바 객체에 대한 참조를 int 값등으로 변환시켜 C 함수내의 지역 변수, 매개 변수, 전역 변수로 참조하더라도 이는 자바 가상 머쉰의 영역을 벗어나는 것으로서, 즉 자바 스레드 스택이 아닌 네이티브 스택이어서 자바 가상 머쉰의 스레기 수거 기능이 동작하지 못한다. 따라서, 자바의 static 변수나 지역 변수, 매개 변수에 의해 참조되지 않으면서 쓰레기 수거되지 않고 C 변수를 통하여 지속적으로 자바 객체를 접근할 수 있도록 JNI C 함수를 호출하여  JNI global reference로 JVM내에 등록시킬 수 있으며, 물론 등록 해제도 가능하다.

따라서, 자바의 사용되는 메모리란 사용될 가능성이 있다는 것일뿐이므로 논리적으로도 정확하게 사용되고 있는 객체가 아닌 사실상의 쓰레기 객체가 있을 수 있으며 이러한 객체들이 자바나 닷넷 프로그램의 메모리 누수 현상을 초래하는 것이다.

이와 같이 사살상의 쓰레기인지 아닌지는 기계적인 검출이 사실상 곤란하여 툴의 도움을 받을 수 있을지라도 프로그래머가 로직을 이해하여 파악해야 한다.  
그렇지 않은 객체들은 어떠한 방법으로도 참조할 수 있는 수단이 없어서 확실하게 쓰레기 객체라는 것을 의미하며, 최근 버전의 자바 가상 머쉰은 이런 확실한 쓰레기 객체는 확실하게 수거해서 재사용되게 해준다.

1. 자바의 메모리 누수에 대해 설명하시오

* 자바에서는 GC에 의해 메모리가 자동 관리되어 memory leak가 없다고 하지만, 사실은 memory leak가 발생할 수 있다.

그 이유는 실제로 사용되지 않는 객체의 reference를 프로그램에서 잡고 있으면 그 객체는 GC에 의해 처리되지 않고 프로그램 내에서도 접근하여 사용될 수 없는 사실상 쓰레기로서 메모리(보다 정확하게는 주소 공간)를 점유하게 된다.

그러한 메모리 누수 현상이 있으면 창을 열고 닫을 때마다 그리고 문서를 열고 닫을 때마다 지속적으로 메모리가 증가되어 성능 저하뿐만 아니라 결국에는 메모리 오류 발생으로 프로그램이 종료되는 심각한 현상이 발생한다.

1. JVM 의 힙크기는 어떻게 지정할수 있는가 ?

* JVM 의 각각 메모리 영역에 할당된 메모리 크기를 지정할수 있는 커멘드라인 메게변수를 수정

1. JVM 에서 동작하는 실제 자바 코드의 생명주기 란 ?

* 컴파일러 에 의해서 자바코드는 .class 확장자의 바이트코드 파일로 변환된다
* 클래스로더 에 의해서 생성된 바이트코드 파일을 JVM 메모리 로 적재
* JVM 은 적재된 바이트코드가 유효한지 검증한다
* 검증이 정상적으로 끝나면 , 코드가 실행하는 아키텍쳐 와 운영체제 맞는 명령어로 바이트코드를 해석한다

Cf. 바이트코드를 명령어로 해석하는 시간이 초기에는 오래걸렸다

이를 보완하기 위해 JIT (Just In Time) 컴파일러가 등장했고

이는 , 직접 동적으로 실행중인 바이트코드를 명령어로 변환하는 기능을 수행했다

1. JVM 에게 가비지 컬렉션을 실행하라고 명령 할수 있는가

* System 클래스의 전역메소드 인 gc 메소드를 이용해서 가비지 컬렉션을 실행하라고 명령할 수 있다
* 하지만 gc 메소드가 항상 가비지 컬렉션을 실행하는점을 보장하진 않는다

Cf. gc 를 실행하면 , 어플리케이션 자체는 stop-the-world 상태가 되어 , 오히려 실행속도를 늦추는 큰 장애가 될 수 있다

1. Finalize 메소드는 어떠한 역할인가

* 가비지 컬렉션에서 수집해야 하는 객체와 연관된 모든 자원을 제거하는데 필요한 작업을 수행한다

(주의사항) Finalize 메소드는 언제 호출되는지 제어 할수 없다

1. WeakReference 는 무엇인가

* 제네릭 컨테이너 클래스 이다 .

들어있는 인스턴스에 강력한 참조가 없으면 가비지 컬렉션의 수집대상이 될 수 있다

1. Native 메소드 란 무엇인가

* C/C++ 에서 정의된 헤더를 이용할수 있게 해주는 메소드

1. 셧다운 훅이란 무엇인가

* 스레드 객체가 종료(=exit())할때 실행되는 스레드

1. 코드를 병렬로 수행하게 하려면 어떻게 하는가

* 병렬로 수행하고자 하는 코드를 스레드 객체로 만들어서 병렬로 실행한다

= 멀티 스레드

1. Thread Safe 하다는 말은 무슨 의미 인가

* 여러 쓰레드에 의해 코드가 실행이 되더라도 ,

실행 결과에 정확성(혹은 무결성)이 보장되는 것을 뜻함

1. Reentrant 하다 라는 말은 무슨 의미 인가

* 여러 쓰레드에 의해 코드가 동시에 수행 될 수 있고 , 그런 경우에도 실행 결과의 정확성 (혹은 무결성) 이 보장되는 것을 뜻함

1. Thread 클래스와 Executor 인터페이스의 차이점은 무엇인가

* 1) First and foremost difference between Thread and Executor is that java.lang.Thread is a [class](http://www.java67.com/2016/08/difference-between-class-and-interface-in-java.html) in Java while java.util.concurrent.Executor is an [interface](http://www.java67.com/2014/02/what-is-actual-use-of-interface-in-java.html).

2) The Executor concept is actually an *abstraction* over parallel computation. It allows concurrent code to be run in managed way. On the other hand, Thread is a *concrete* way to run the code in parallel.  
  
3) The third difference between an Executor and a Thread class is that former **decouples**a task (the code which needs to be executed in parallel) from execution, while in the case of a Thread, both task and execution are **tightly coupled**. You can further read [Java Concurrency in Practice](http://aax-us-east.amazon-adsystem.com/x/c/QsX6W-q8_FXruEmT5aA2Ys0AAAFd8WZq9gEAAAFKARN2OPo/https:/assoc-redirect.amazon.com/g/r/http:/www.amazon.com/dp/0321349601/ref=as_at?creativeASIN=0321349601&linkCode=w61&imprToken=LtupBwIlDgcadzVCf-Po5g&slotNum=0&tag=javamysqlanta-20) by Brian Goetz to learn more about how decoupling a task from execution simplify the design of concurrent applications in Java.  
  
4) The Executor concept allows your task is to be executed by a worker thread from the thread pool, while Thread itself execute your task.

5) Executor provides a execute() method which accepts a Runnable task, while Thread accepts the [Runnable task](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html) on its constructor.  
  
6) One more key difference between a Thread and an Executor is that a Thread can only execute **one Runnable task** but an Executor can execute any number of Runnable task.  
  
7) In the case of Thread, the task is executed by the Thread which accepts Runnable instance, but in the case of Execution the command (a Runnable implementation) may be executed in a new thread, a pooled thread or in the calling thread itself, depending upon the implementation of Executor interface.  
  
8) In the case of a thread, **it's developer's responsibility to create and start the thread**, but in the case of Executor, the framework will create and start threads for you. Though you can control the whole process by giving your implementation of Executor interface. Though, with the improvements in ForkJoinPool in Java 7 and 8, you might want to use that instead of Executor. If ForkJoinPool is a new concept to you, I suggest reading [Java 8 in Action](http://aax-us-east.amazon-adsystem.com/x/c/QsX6W-q8_FXruEmT5aA2Ys0AAAFd8WZq9gEAAAFKARN2OPo/https:/assoc-redirect.amazon.com/g/r/http:/www.amazon.com/Java-Action-Lambdas-functional-style-programming/dp/1617291994/ref=as_at?creativeASIN=1617291994&linkCode=w61&imprToken=LtupBwIlDgcadzVCf-Po5g&slotNum=1&tag=javamysqlanta-20) to learn more about it.

Read more:

<http://javarevisited.blogspot.com/2016/12/difference-between-thread-and-executor.html#ixzz4q2OcO5oi>

1. 스레드 사이의 공유 상태는 어떻게 관리 하는가

* 자원에 대한 동시성확보(Synchronized , Volatile ) 을 통해 멀티스레드 환경에서도 임의의 자원에 대한 상태를 공유 할 수 있도록 한다

1. 스레드 조인이란 무엇인가

* 스레드 간의 동기화

1. 스레드 인터럽트 란 무엇인가

* 스레드 종료

1. Atomic 클래스는 무엇을 제공하는가

* 원자 연산을 보장한다

1. Akka 란 무엇인가

* 동시코드를 작성하고 작업할 때 다른 접근법을 제공해주는 프레임워크

1. Actor 객체 대신 ActorRefs 객체를 사용할때의 장점은 무엇인가

* 어플리케이션을 호출한 다음 응답을 받을 때 까지 기다리게 하는 것 보다 메시지를 처리하는 방식으로 하여 , 빠르게 응답하는 시스템을 만들수 있게 해준다

1. 어떻게 하면 병렬 메시지 처리가 가능하도록 만들 수 있는가

* 여러 개의 액터를 생성하고 관리하는 액터시스템 을 통해 가능하다

1. NIO 란 무엇인가 ?
2. DeadLock , Mutex , Semaphore 설명하시오

* 안드로이드

1. APK 형식은 무엇인가 ?
2. 안드로이드 아키텍쳐의 주요 구성 요소는 무엇인가 ?



1. 안드로이드 에서 자바 바이트 코드를 실행할수 있는가 ?

* 실행 할 수 없다

1. 쓰레드간에 통신방법에 대해 설명하시오

* UI 스레드를 통해서

각각의 스레드(Worker Thread)가 통신할 수 있다

1. 메시지 핸들러 (Handler) 에 대해서 설명하시오

* 쓰레드간 상호작용을 위한 일반적인 목적의 클래스

1. 메시지 핸들러 를 구성하는 Looper 에 대해서 설명하시오

* 메시지 큐에 있는 매세지 객체나 혹은 Runnable 객체를 꺼내서 실행하는 주체

1. 백그라운드에서 UI 업데이트 하는 방법은 ?

* 메인 스레드를 루퍼로 가지고 있는 핸들러에 요청

1. Http 통신이나 리소스 관리 등 시간소요 시 구현방법은 ?

* 일정시간 후에 통신을 종료하는 , 타임아웃 설정

1. Context 에 대해서 설명하시오

* 안드로이드 에서 제공하는 글로벌 정보 혹은 메소드에 접근하기 위한

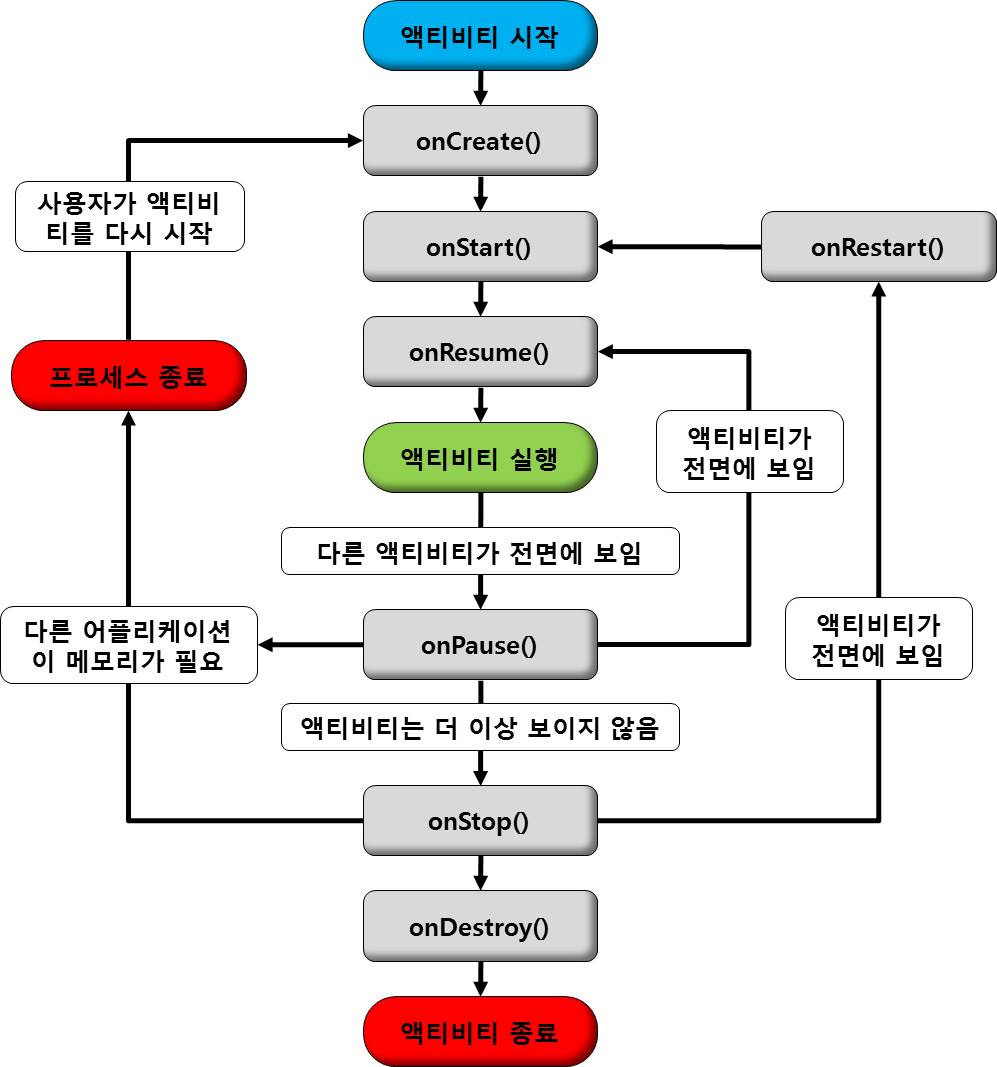
추상 클래스

* 실제 구현은 안드로이드 에 의해서 구현이 되며

\* 어플리케이션에 관하여 시스템이 관리하고 있는 정보에 접근

\* 안드로이드 시스템 서비스에서 제공하는 API를 호출 하는 기능

1. 액티비티의 생명주기에 대해서 얘기하고 각 항목에 대해서 설명하시오



* 액티비티 생명주기는 onCreate() -> onStart() -> onResume() -> onPause() -> onStop() -> onDestory()순으로 실행

1. 액티비티의 onRestart 는 언제 호출되는가 ?

* 액티비티가 멈췄다가 다시 시작되기 바로 전에 호출됨.

1. ANR 은 무엇인가

* ANR (Application Not Responding) 은 어플리케이션이 응답하지 않는다는 에러

에러의 원인은 Main Thread(UI Thread)가 일정 시간 어떤 Task에 잡혀 있으면 발생하게 된다

* ANR은 대체로 아래의 상황에서 발생하게 된다.

애플리케이션이 UI 스레드에 어떠한 I/O 명령(빈번한 네트워크 액세스)으로 인해 막힐 때

너무 많은 시간을 정교한 메모리 구조를 구축하는데 들일 때

구체적으로 설명하자면 [Android Developers](https://developer.android.com/training/articles/perf-anr.html#anr) 에서는 아래와 같이 명시하고 있다.

Input 이벤트(키를 누르거나 화면을 터치하는 등)에 5초안에 반응을 하지 않을 때

BroadcatReceiver 가 10초내로 실행을 끝내지 않을 때 (UI가 없는 브로드캐스트 리시버, 서비스도 실행 주체가 메인스레드이므로 긴 시간을 소모하는 작업인 경우 ANR을 발생시킨다.)

1. AsyncTask 에 대해 설명하시오

* 백그라운드 스레드에서의 작업결과를 UI 스레드에 반영하기 위한 기능을 가진 추상 클래스

1. Parcel – Serialzable 비교

* Serialzable 이란 ? Value Object 를 쉽게 직렬화 하기 위한 인터페이스
* Parcelable 이란 ? IPC (Inter Process Communication) 에 최적화되어 있는 Serialzable 구현체

1. Parcelable is faster than serializable interface
2. Parcelable interface takes more time for implemetation compared to serializable interface
3. serializable interface is easier to implement
4. serializable interface create a lot of temporary objects and cause quite a bit of garbage collection
5. Parcelable array can be pass via Intent in android
6. 인텐트 는 무엇인가 ?

* 어플리케이션 구성요소(컴포넌트) 간에 작업 수행을 위한 정보를

전달하는 역할

1. 안드로이드 매니페스트 파일은 무엇인가

* 어플리케이션에 대한 전반적인 정보를 담고 있는 파일로, 안드로이드 시스템에서 어플리케이션의 정보를 알아내는데 사용합니다

1. 명시적 인텐트 , 암시적 인텐트 에 대해서 설명하시오

* 명시적 인텐트 란 파라미터로 실행할 다음 엑티비티의 클래스이름을 넘겨주어 새 엑티비티를 실행하는 인텐트
* 암시적 인텐트는 다른 기능을 하는 앱을 호출할 수 있는 인텐트

1. 암시적 인텐트 의 사용 예는 ?

* 전화걸기 , 이메일 열기 , 웹브라우저 열기

1. Intent.FLAG\_ACTIVITY\_CLEAR\_TOP , FLAG\_ACTIVITY\_SINGLE\_TOP

에 대해 설명하시오

* Clear Top 은 생성하려는 액티비티와 동일한 종류의의 액티비티가 스택 내부에 있을 경우,

기존 동일한 액티비티 위의 모든 액티비티를 종료 시키고 기존 액티비티를 새로 생성된 액티비티로 교체한다

* Single Top 은 생성 하려는 액티비티가 스택 최상위에 없을 경우 액티비티를 새로 생성한다

1. 데이터 저장방식 5가지에 대해 설명하시오

|  |  |
| --- | --- |
| 방법 | 설명 |
| 공유 프레퍼런스(Shared Preferences) | 키-값 쌍(key-value pair)으로 사적이고 기초적인 데이터를 저장한다. |
| 내부저장 (Internal Storage) | 사적인 데이터를 장치 메모리에 저장한다. |
| 외부저장 (External Storage) | 공유 데이터를 공유 외부 저장소에 저장한다. |
| SQLite 데이터베이스(SQLite Databases) | 구조화된 데이터를 사적인 데이터베이스에 저장한다. |
| 네트워크 연결(Network Connection) | 데이터를 네트워크 서버에 저장한다. |

* **프레퍼런스(Preferences)**

프레퍼런스Preference는 기본적인 데이터 타입에 대한 키와 값의 쌍key-value pair을 저장하고 가져오는 가벼운 메커니즘이다. 이것은 전형적으로 애플리케이션이 시작될 때마다 로드되어야 하는 기본적인 환영 인사말이나 텍스트 폰트와 같은, 애플리케이션의 환경설정 정보를 저장하기 위해 사용된다.

여러분은 (컨텐트 프로바이더를 사용하는 것에 의하지 않고서는) 프레퍼런스를 애플리케 이션의 경계를 넘어서 공유할 수 없다. //두가지 어플리케이션을 동시 실행했을 때의 경우

* **파일(Files)**

여러분은 모바일 디바이스 상에 또는 분리될 수 있는 저장 매체 상에 직접적으로 파일들을 저장할 수 있다. 디폴트로 다른 애플리케이션들은 이러한 파일들에 접근할 수 없다. 여러분은 단지 로컬 파일들에만 접근할 수 있다.   
만약 여러분이 컴파일 시점에 여러분의 애플리케이션과 함께 패키지할 정적static 파일을 가지고 있다면, 여러분 프로젝트의 res/raw/myDataFile로 그 파일을 저장할 수 있으며, 그리고 나서 Resources.openRawResource(R.raw.myDataFile)를 사용해서 그것을 오픈할 수 있다.

* **데이터베이스(Databases)**

안드로이드 API는 SQLite 데이터베이스를 생성하고 사용하는 것에 대한 지원을 포함한다. 각각의 데이터베이스는 그것을 생성한 애플리케이션에게 프라이빗 private하다.   
SQLiteDatabase 오브젝트는 데이터베이스를 나타내고, 그것과 상호작용하는 - 쿼리query를 하고 데이터를 관리하는 - 메쏘드들을 가지고 있다. 데이터베이스를 생성하기 위해서는 SQLiteDatabase.create()를 호출하고, 또한 서브클래스 SQLiteOpenHelper를 호출하라.

모든 데이터베이스(SQLite와 그 밖의 것들)는 디바이스 상의 /data/data/package\_name/databases 안에 저장된다

1. Data Binding 에 대해 설명 하시오

* Xml 상의 뷰와 코드간의 연결을 간결하게 도와주는 라이브러리

1. AIDL 에 대해 설명하시오

* AIDL (안드로이드 인터페이스 정의 언어)는 하나의 안드로이드 장치 위에서

잠김 프로세스 통신 규약(IPC)에 의해 두 프로세스들 사이에 대화가 가능하도록 하기 위한 목적으로 생성 되어진 코드를 사용할 수 있게 하는 IDL 언어

1. 액티비티 는 무엇인가

* 일종의 애플리케이션 구성 요소로서, 사용자와 상호작용할 수 있는 화면을 제공하는 안드로이드 기본 뷰 단위

1. Fragment 에 대해 설명하시오

* 액티비티(사용자 와의 상호작용 할수 있는 화면을 제공하는)를 부분적으로 구성 하는 뷰

1. Android 3.0 미만 에서의 Fragment 구현 설명

(= support library , activity , FragmentManager 차이)

* Android 3.0 (API Level 11) 미만 에서는 플랫폼에서 제공하지 않은므로 support library 를 이용하여 구현 해야한다

1. 서비스는 무엇인가

* Service는 Activity처럼 사용자와 사용자와 상호작용하는 컴포넌트가 아니고, Background에서 동작하는 컴포넌트
* Activity가 종료되어 있는 상태에서도 동작하기 위해서 만들어진 컴포넌트입니다

만약 Service가 실행되고 있는 상태라면 안드로이드 OS에서는 해당 Process를 죽이지 않도록 방지하고 관리합니다.

그렇게 때문에 메모리 부족이나 특별한 경우를 제외하고는 Background 동작을 수행하도록 설계되어있습니다

1. 안드로이드 어플리케이션 과 프로세스 에 대해 설명

* 안드로이드 4대 컴포넌트 (액티비티, 서비스, 리시버, 컨텐츠프로바이더) 로

구성된 응용 프로그램 , 이때의 프로그램은 보조기억장치에 존재하며 실행되기를 기다리는 명령어 와 정적인 데이터의 묶음을 의미함

* 프로세스란 , 메모리 (CPU) 에 적재된 응용 프로그램을 의미 함
* 어플리케이션을 구성하는 컴포넌트가 실행하고자 할 때 , 메모리에 구동중인 프로세스가 없을 때 OS 로부터 하나의 스레드로 구성된 프로세스를 할당 받아 실행되어진다. 이때의 스레드를 메인스레드 (혹은 UI 스레드) 라고 일컫는다

기본적으로 이후에 실행되는 , 같은 어플리케이션 내의 모든 컴포넌트는 이 메인스레드에 귀속되어 실행되지만 , 서로 다른 프로세스에서 동작할수 있도록 재배치 할수 있다

메니페스트 파일에서 컴포넌트를 선언할 때, android:process 필드를 지정하면, 이 컴포넌트는 앱의 프로세스와는 별개로, 다른 독립된 프로세스에서 실행된다

1. 프로세스 라이프사이클 이란 ?

* 안드로이드 시스템이 제한적인 메모리 환경에서 어떠한 프로세스를 종료시킬지에 대한 5가지 기준

#1 포그라운드 프로세스: 현재 사용자와 상호작용 하고 있는 액티비티

#2 비주얼 프로세스 : 상호작용 하고 있는 액티비티는 아니지만 , 여전히 화면에 노출되고 있는 화면을 가지고 있는 프로세스

#3 서비스 프로세스 : 화면으로 노출되고 있지 않은 서비스를 가지고 있는 프로세스를 의미

#4 백그라운드 프로세스

#5 Empty 프로세스

1. 안드로이드 프로세스 와 서비스 , 그리고 스레드 간 차이점 ?

* 프로세스는 운영 체제로부터 할당 받은 작업의 단위 이고

스레드는 프로세스 로부터 할당받은 자원을 이용하여 실행되는 단위

* 하나의 프로세스에서 가지고 있는 자원을 여러 개의 스레드가 공유하며 동작함

1. startService 와 bindService 설명

* 바인드 서비스는 서비스를 실행한 객체와 라이프사이클이 동일하게 동작하게끔 서비스를 실행
* 스타트 서비스는 서비스를 실행한 객체와 상관없이 동작하도록 실행하는 메소드

1. Android Service 와 Intent Service 의 차이

* 서비스는 , 노출되는 UI 가 없이 동작하는 메인스레드 컴포넌트
* 인텐트 서비스는 , 인텐트로 실행하는 워커 스레드

인텐트 서비스는 onHandleIntent 내에 동작을 모두 처리하면 자동적으로 종료된다

인텐트 서비스는 병렬적으로 수행되지 않으며 , 연속적인 인텐트 호출에 의해 순차적으로 수행 된다

1. 리시버는 무엇인가

* 글로벌 하게 이벤트를 수신 해서 동작하는 컴포넌트

1. 컨텐츠 프로바이더는 무엇인가

* 어플리케이션 간에 데이터를 공유 하는 기능을 제공하는 컴포넌트

1. ROA 란 무엇인가 ?

* ROA(Resource Oriented Architecture)
* 웹의 모든 리소스를 URI로 표현하여 구조적이고 유기적으로 연결하여 비 상태 지향적인 방법으로 정해진 method만을 사용하여 리소스를 사용하는 아키텍처
* 이는 4가지의 고유한 속성과 연관되어 진다. ( REST는 이 속성들을 지향한다 )

- Addressablilty

- Connectedness

- Statelessness

- Homogeneous Interface

\* Addressablilty (주소로 표현 가능함)

- 제공하는 모든 정보를 URI로 표시할 수 있어야 한다.

- 직접 URI로 접근할 수 없고 HyperLink를 따라서만 해당 리소스에 접근할 수 있다면 이는 RESTful하지 않은 웹서비스이다.

+ Connectedness (연결됨)

- 일반 웹 페이지처럼 하나의 리소스들은 서로 주변의 연관 리소스들과 연결되어 표현(Presentation)되어야 한다.

- 예를 들면,

<user>

<name>HJ</name>

</user> 는 연결되지 않은 독립적인 리소스이다.

<user>

<name>HJ</name>

<address>HJ/seoul/</address>

<phone>HJ/010</phone>

</user>

는 관련 리소스(address, phone)가 잘 연결된 리소스의 표현이다.

+ Statelessness (상태 없음)

- 현재 클라이언트의 상태를 절대로 서버에서 관리하지 않아야 한다.

- 모든 요청은 일회성의 성격을 가지며 이전의 요청에 영향을 받지 말아야 한다.

- 다시 또 코리아닷컴의 예를 들면 메일을 확인하기 위해 꼭 '..코리아닷컴../mailView.crd'에 접근하여 해당 세션을 유지한 상태에서 메일 리소스에 접근해야 한다. 이것이 바로 Statelessness가 없는 예이다.

- 세션을 유지 하지 않기 때문에 서버 로드 발란싱이 매우 유리하다.

- URI에 현재 state를 표현할 수 있어야 한다. (권장사항)

+ Homogeneous Interface (동일한 인터페이스)

- HTTP에서 제공하는 기본적인 4가지의 method와 추가적인 2가지의 method를 이용해서 리소스의 모든 동작을 정의한다.

- 리소스 조회 : GET

- 새로운 리소스 생성 : PUT, POST (새로운 리소스의 URI를 생성하는 주체가 서버이면 POST를 사용)

- 존재하는 리소스 변경 : PUT

- 존재하는 리소스 삭제 : DELETE

- 존재하는 리소스 메타데이터 보기 : HEAD

- 존재하는 리소스의 지원 method 체크 : OPTION

- 대부분의 리소스 조작은 위의 method를 이용하여 대부분

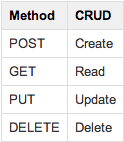
처리 가능하다.

만일 이것들로만 절대로 불가능한 액션이 필요할 경우에는 POST를 이용하여 추가 액션을 정의할 수 있다. (되도록 지양하자)

1. RestAPI 란 무엇인가

* REST는 Representational State Transfer 의 약자이다

ROA(Resource Oriented Architecture) 설계의 중심에 Resource가 있고 HTTP Method를 통해 Resource를 처리하도록 설계된 아키텍쳐



1. RestAPI 의 장점은 무엇인가 ?
2. OPENAPI를 제공하기 쉽다.
3. 멀티플랫폼 지원 및 연동이 용이하다.
4. 원하는 타입으로 데이터를 주고받을수 있따. (XML, JSON, RSS )
5. 기존 웹 인프라를(HTTP)를 그대로 사용가능하다 ( 방화벽, 장비 요건 불필요 )
6. 사용하기 쉽다

6. 세션을 사용하지 않는다. 각각의 요청에 독립적.

1. RestAPI 의 단점은 무엇인가 ?
2. 표준이 없어서 관리가 어렵다
3. 사용할수 있는 메소드가 4가지 밖에 없다
4. 분산환경에는 부적합하다
5. HTTP 통신 모델 에 대해서만 지원한다
6. 안드로이드 에서 RestAPI 는 어떻게 구현하는가

* 기존 ROA 에 기반한 HTTP 통신을 그대로 유지해서 사용해도 무방하나

Retrofit 이라는 라이브러리를 사용하는 것이 보다 더 효율적이고 속도가 빠르다

1. 인텐트 필터란 무엇인가

* 어플리케이션 컴포넌트가 어떠한 인텐트를 받고자 하는지

정할 수 있게 해주는 수단

(37) Doze 모드 에 대해 설명하시오

* 안드로이드 6.0 이후부터 , 베터리를 절약하기 위해 사용자가 휴대폰을 사용하고 있지 않다고 판단되면 , 시스템은 모든 앱의 베터리를 백그라운드 작업과 네트워킹을 일시 중지 시키는 기능

(38) Android MAT 는 무엇인가

* 안드로이드 어플리케이션의 메모리 프로파일링을 위한 툴

(39) GCM 은 무엇인가

* GCM은 구글 클라우드 메시징의 약자로 구글에서 푸쉬 메시지를 보낼 때 사용하는 무료 서비스

1. View 객체 안의 invalidate() 메소드의 의미는 ?

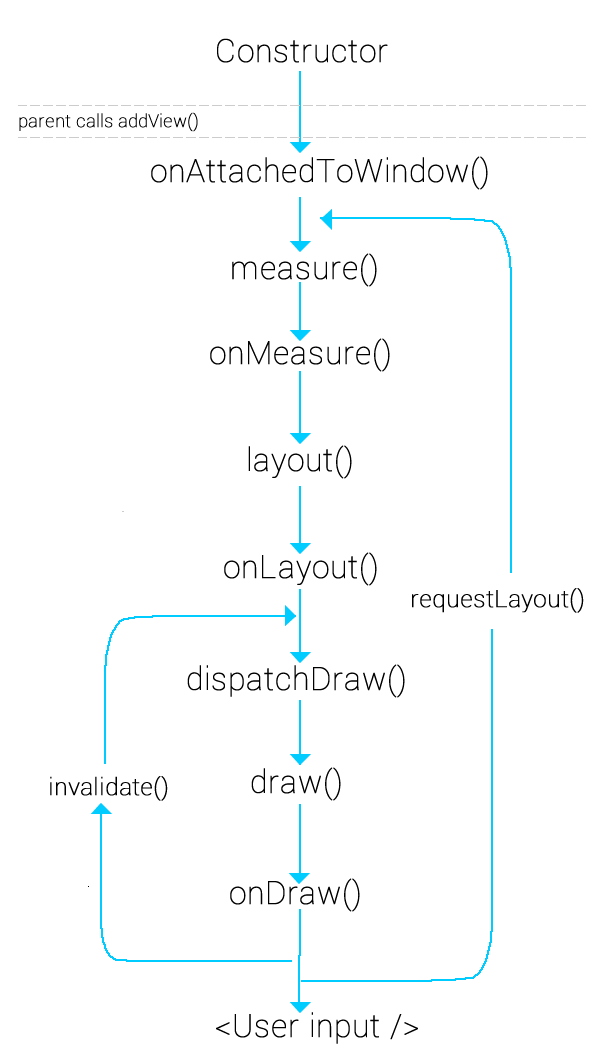
* 뷰를 갱신 하기 위한 메소드

1. Invalidate() 와 postInvalidate() 의 차이점은 ?

* Invalidate 는 메인 스레드 에서만 실행 될수 있는 메소드 이고

postInvalidate 는 워커스레드에서 실행 되어도 , 이벤트 큐에 동작 대기 시킨후에 메인스레드에 의해 실행될수 있게끔 보장해주는 메소드

1. View 객체의 생명주기는 ?

* 

(42) Post() 메소드의 의미는 무엇인가 ?

* 해당 뷰가 윈도우에 attached 되었을때 , 쌓여있던 메시지 큐를 실행 보장 해주는 메소드

(43) 리스트뷰의 뷰홀더 를 쓰는 이유는 무엇인가 ?

* 자원 소모 비용이 상대적으로 큰 findViewById 를

자주 호출 하지 않기 위해 , 한번 메모리에 적재한 뷰를 다시 findViewById 로 찾지 않고 멤버변수로 해당 뷰를 저장하고 사용 하도록 하기 위함

1. 리사이클러뷰 와 리스트뷰 의 차이점은 ?

* 리스트뷰에서 권장하던 뷰홀더 패턴을 리싸이클러뷰 에서는 강제적으로 써야 한다는 점이 가장 큰 차이
* 그외에 리스트뷰 에서는 커스텀으로 제작 해야 했던 UI 동작을 리싸이클러 뷰에서는 API 로서 풍부하게 제공 된다는 점

(예 , ItemDecoration , ItemAnimator .. )