6. 큐

6.1 큐 ADT

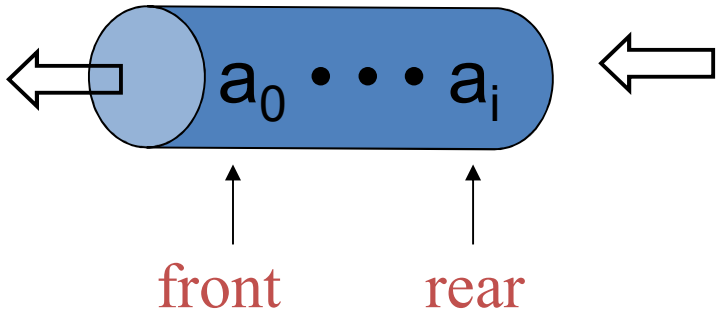
- 선입선출(FIFO: first-in-first-out) 프로토콜을 구현하는 자료 구조
- 원소의 삽입은 큐의 뒤(rear)에서 수행되고, 제거는 앞(front)에서 진행
- 예, 입장권을 구입하기 위해 기다리는 사람들의 줄
- 예, 활주로에서 이륙을 위해 대기중인 비행기의 줄

- ADT: Queue
  - *큐(queue)*란 FIFO 접근 프로토콜을 유지하는 원소들의 컬렉션이다.
  - 연산
    1. Add: 주어진 원소를 큐의 뒤에 삽입한다.
    2. First: 큐가 공백이 아니면, 큐의 앞에 있는 원소를 리턴한다.
    3. Remove: 큐가 공백이 아니면, 큐의 앞에 있는 원소를 삭제해서 리턴
    4. Size: 큐에 있는 원소의 수를 리턴한다.

$Q = (a_0, \bullet\bullet\bullet, a_{n-1})$

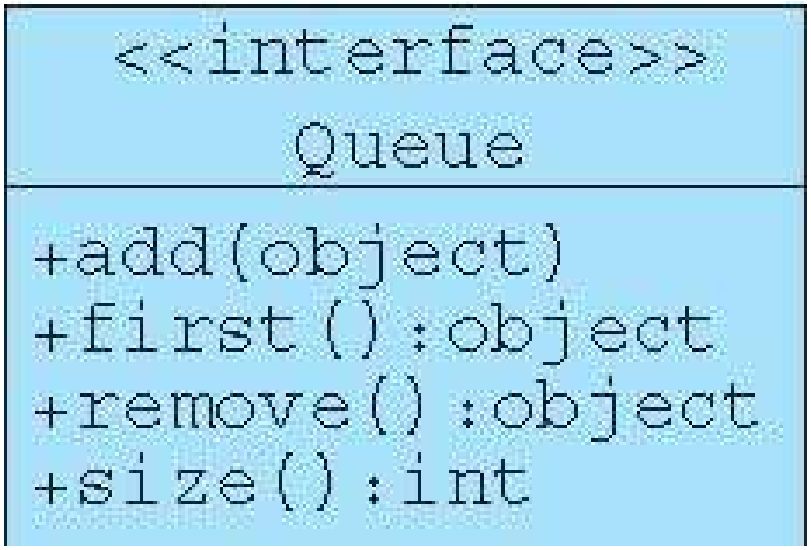
front element

rear element



- FIFO (First-In-First-Out)

Queue ADT에 대한 UML 다이어그램



Queue ADT에 대한 Java 인터페이스

```
public interface Queue {
   /** Adds the specified element to the back of this queue.
   @param object the element to be added to this queue.  */
   public void add(Object object);
   /** Returns the element at the front of this queue.
   @return the element at the front of this queue.
   @throws IllegalStateException if this queue is empty */
  public Object first( );
   /** Removes and returns the element at the front of this queue.
   @return the element at the front of this queue.
   @throws IllegalStateException if this queue is empty  */
  public Object remove();
   /**  Returns the number of elements in this queue.
   @return the number of elements in this queue.  */
  public int size();
}
```

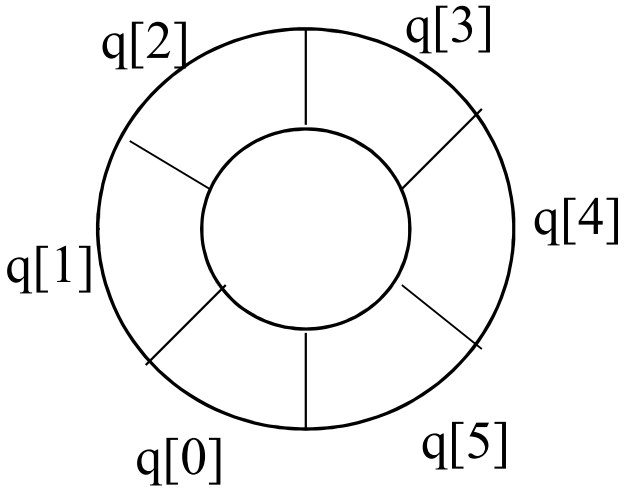
배열 구현

```
public class ArrayQueue implements Queue {
    private Object[ ] a;
    private int front, rear;
    public ArrayQueue(int capacity) { a = new Object[capacity];  }
    public void add(Object object) {    /*  구현  */      }
    public Object first( ) {      /*  구현  */      }
    public boolean isEmpty( ) {      /* 구현 */     }
    public Object remove( ) {    /* 구현 */   }
    public int size( ) {   /* 구현 */    }
}
```

- 규칙: front는 처음 삽입한 원소보다 하나 작은 index라고 하자
- 규칙: rear는 마지막에 삽입한 원소의 index라고 하자
- front와 rear의 초기값은?

배열 구현 – 환상큐(circular queue)

- 배열을 원형이라 가정한다

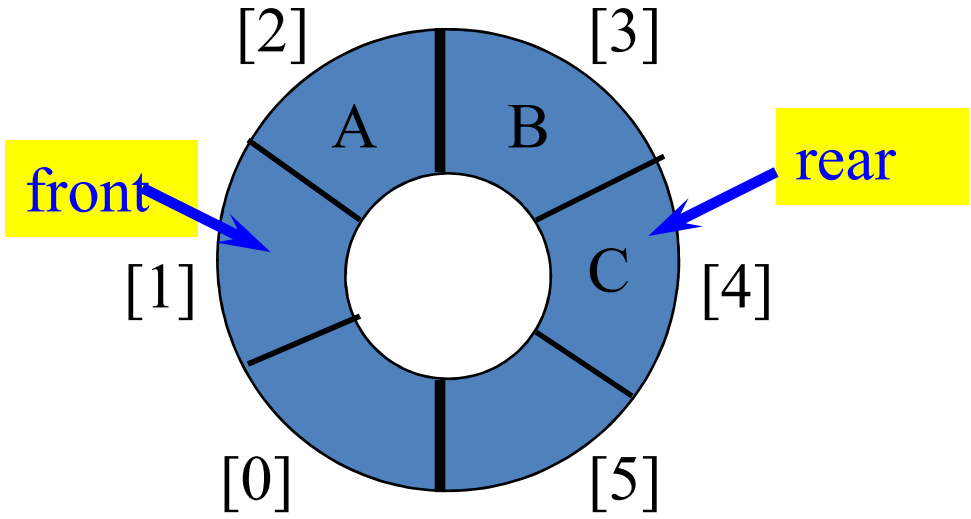


MAX_QUEUE_SIZE =6

초기 front = rear = 0

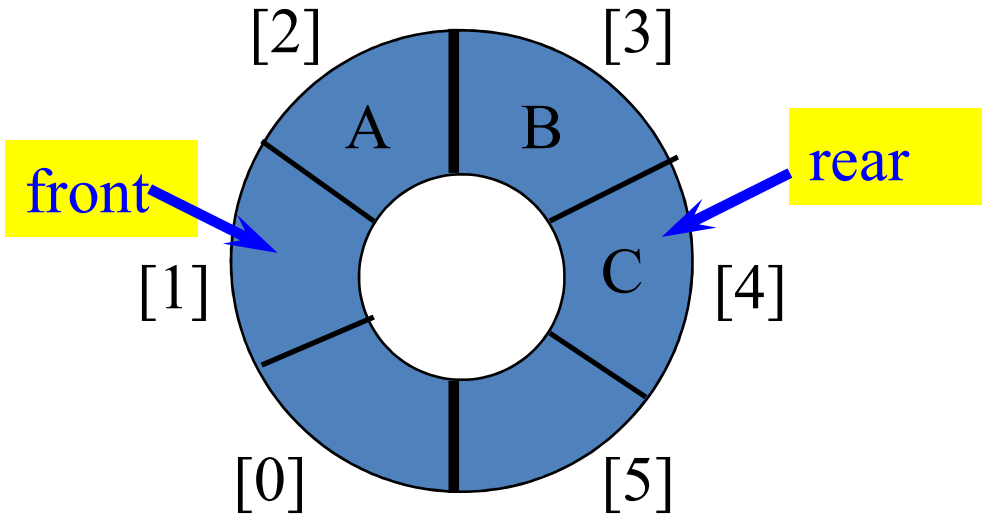
Circular Array

- front is one position counterclockwise from first element
- rear gives the position of last element
- initially, front = rear = 0



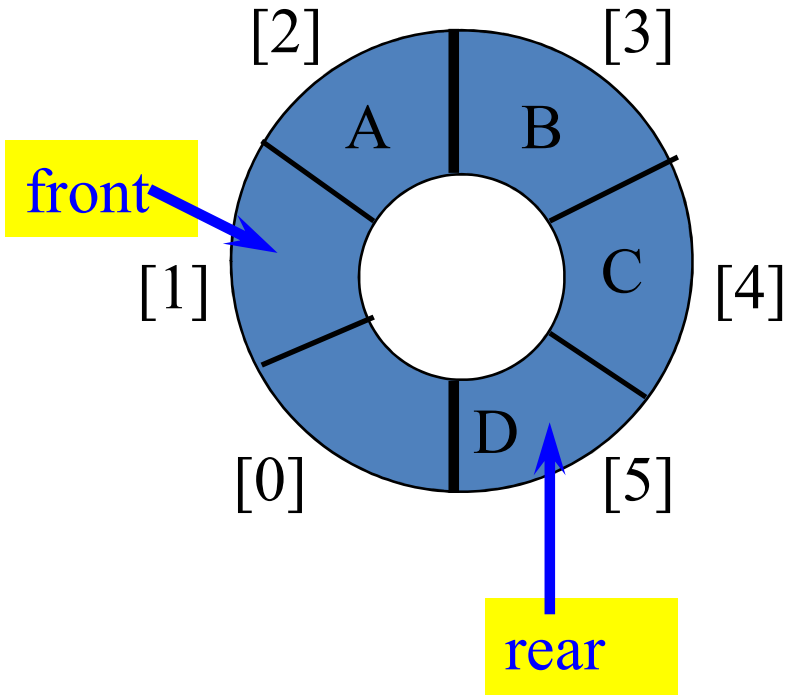
Add An Element

- Move rear one clockwise.



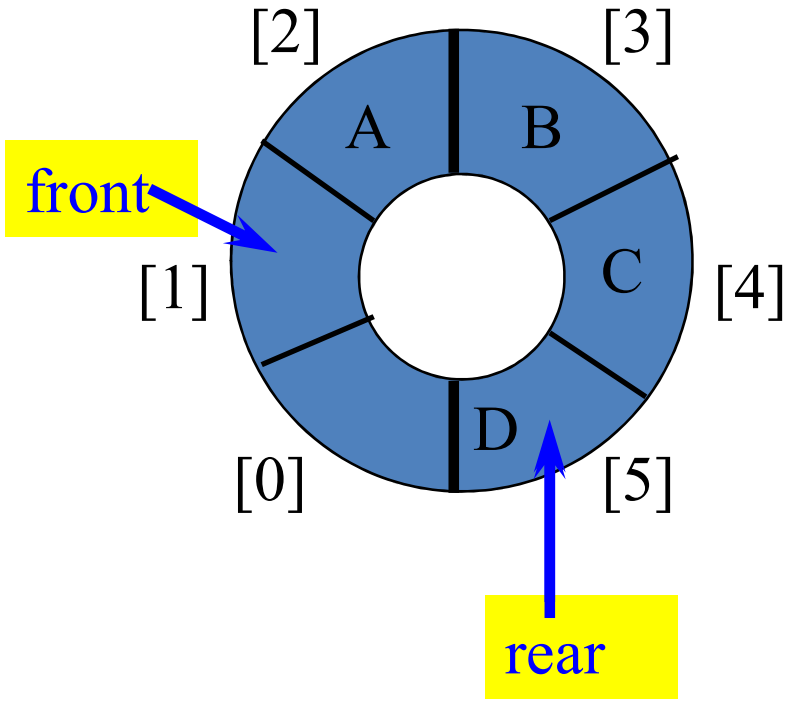
Add An Element

- Move rear one clockwise.
- Then put into queue[rear].



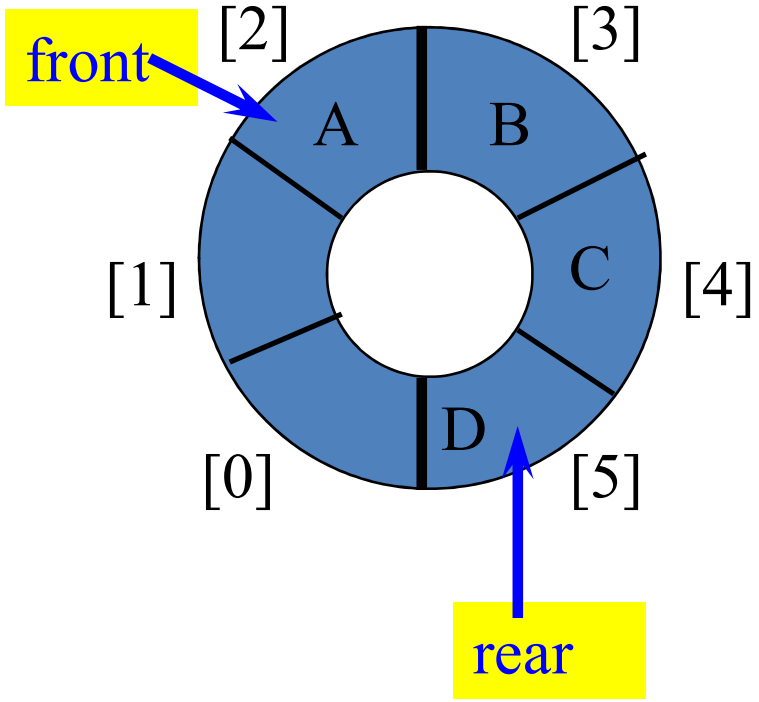
Delete An Element

- Move front one clockwise.



Delete An Element

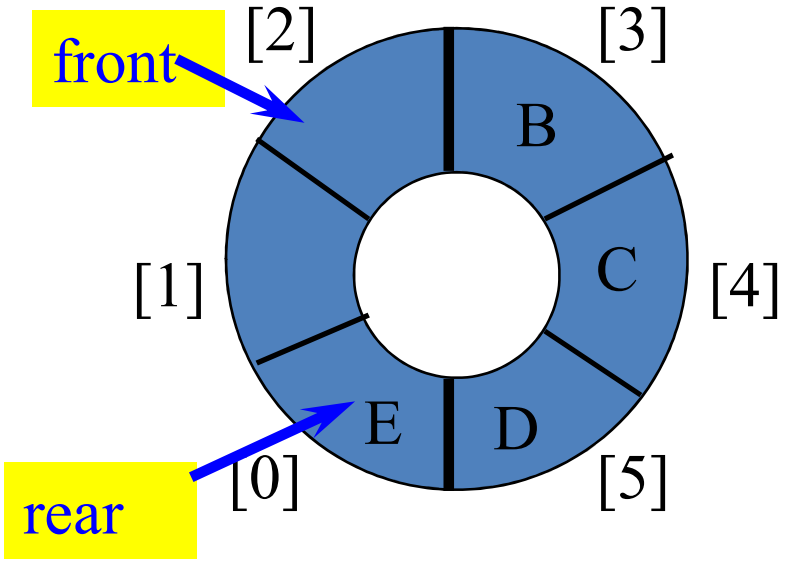
- Move front one clockwise.
- Then extract from queue[front].



Moving rear clockwise

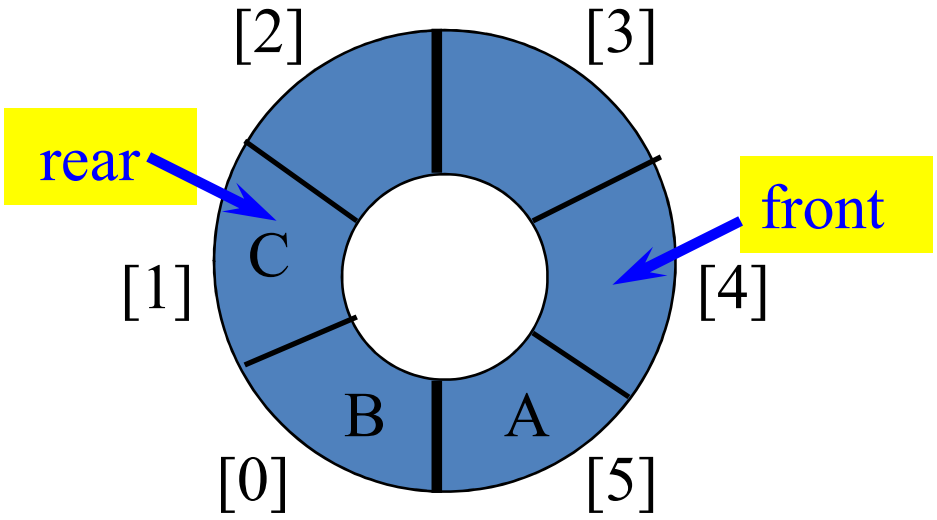
- rear++;

  if (rear = = MAX_QUEUE_SIZE) rear = 0;

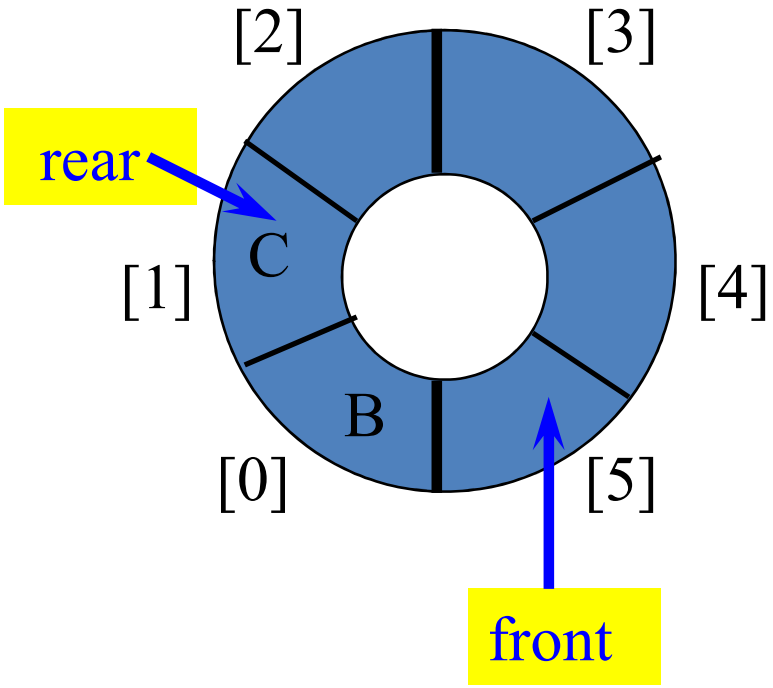


- rear = (rear + 1) % MAX_QUEUE_SIZE ;

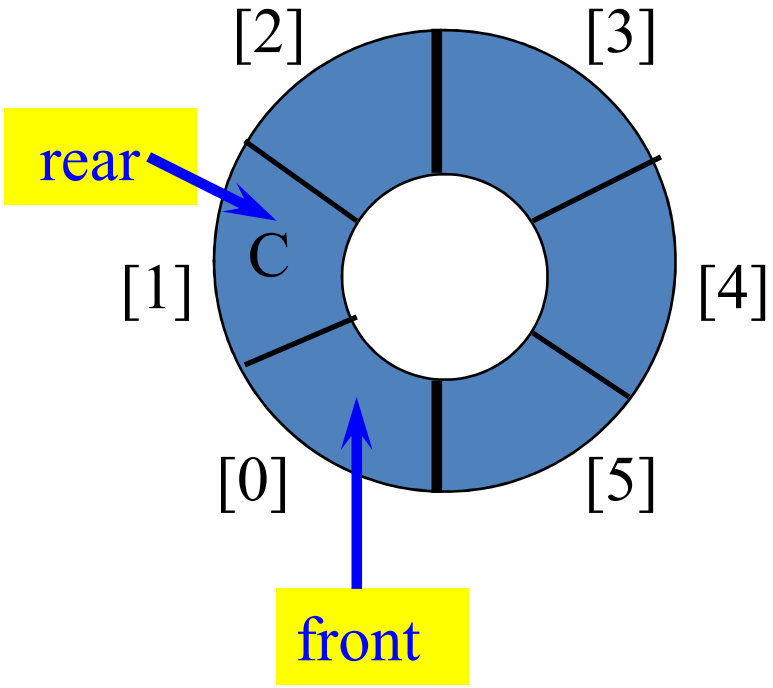
Empty The Queue



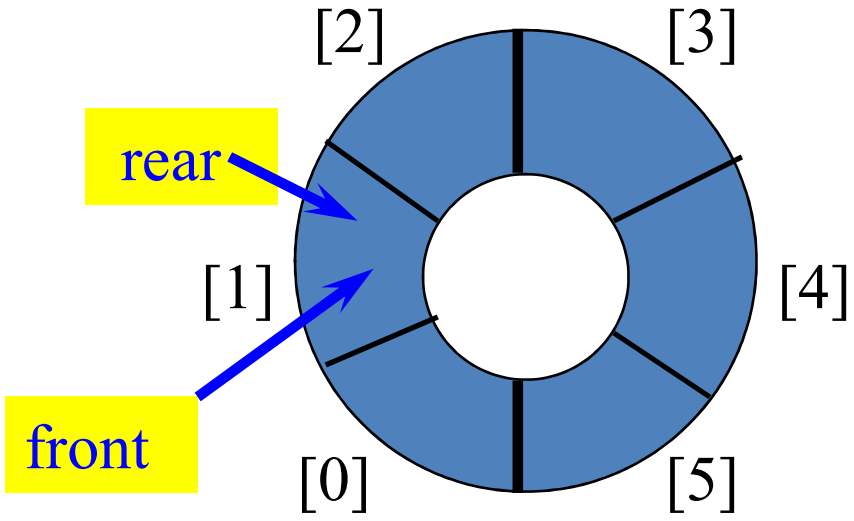
Empty The Queue



Empty The Queue

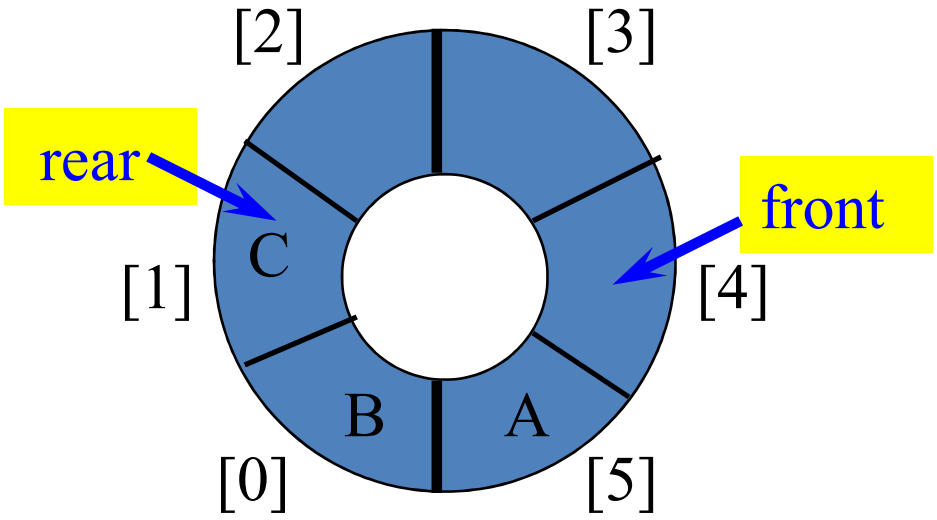


Empty The Queue

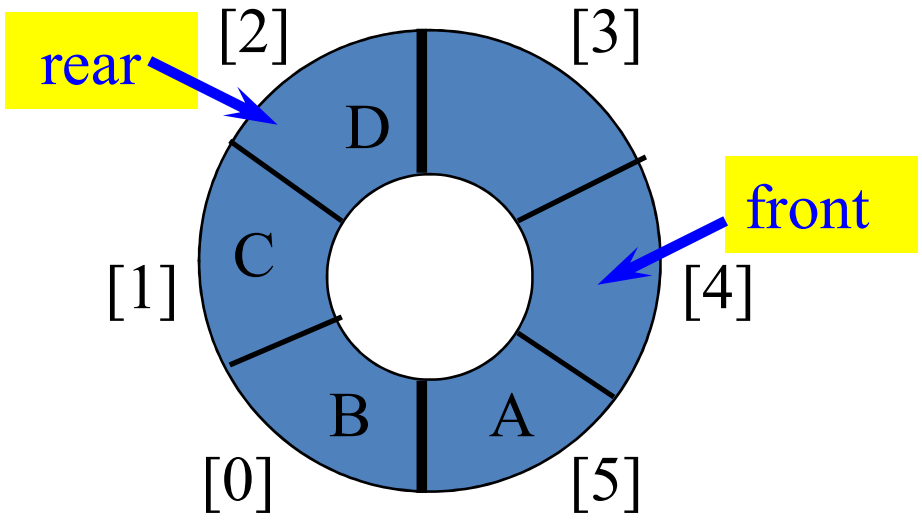


- When a series of removes causes the queue to become empty, front == rear.
- When a queue is constructed, it is empty.
- So initialize front = rear = 0.

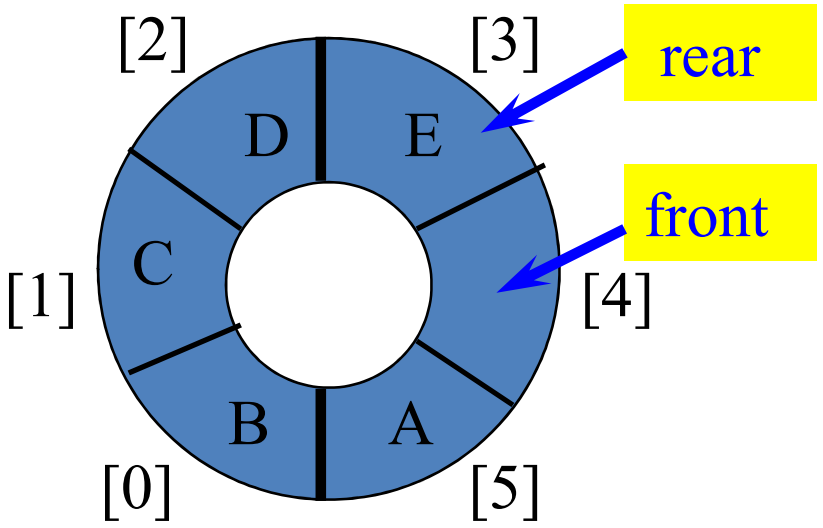
A Full Tank



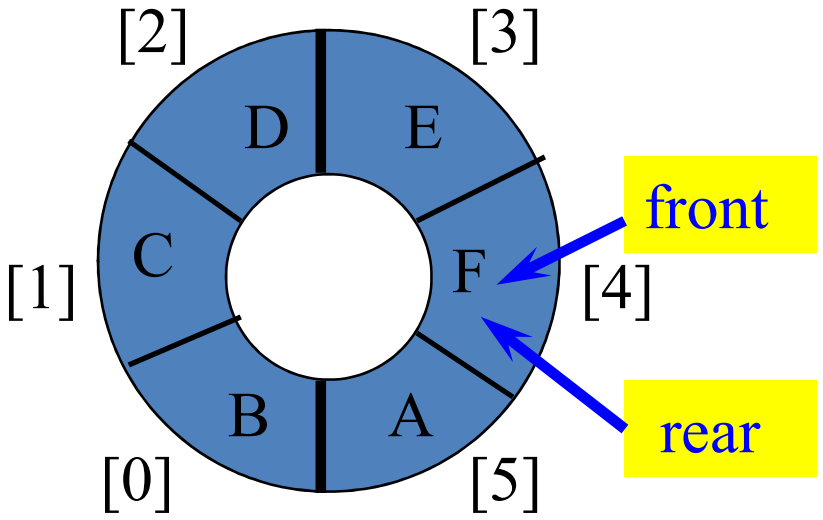
A Full Tank



A Full Tank



A Full Tank

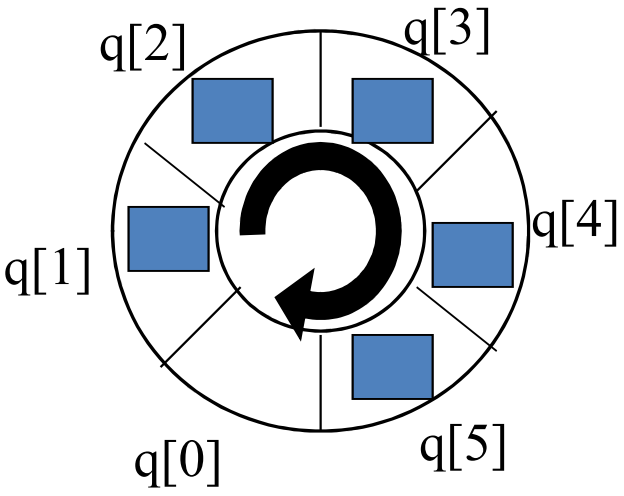


- When a series of adds causes the queue to become full, front = =rear.
- So we cannot distinguish between a full queue and an empty queue!

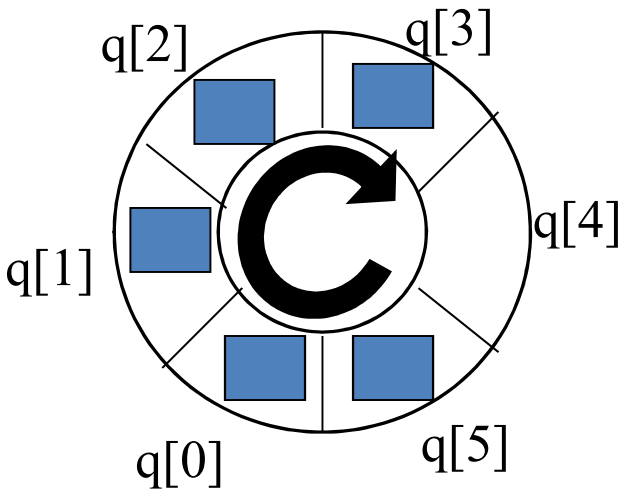
Circular queue

- *Permit to hold at most MAX_QUEUE_SIZE –1 elements*

front = 0, rear = 5



front = 4, rear = 3



6.3 연결 구현(A Linked Implementation)

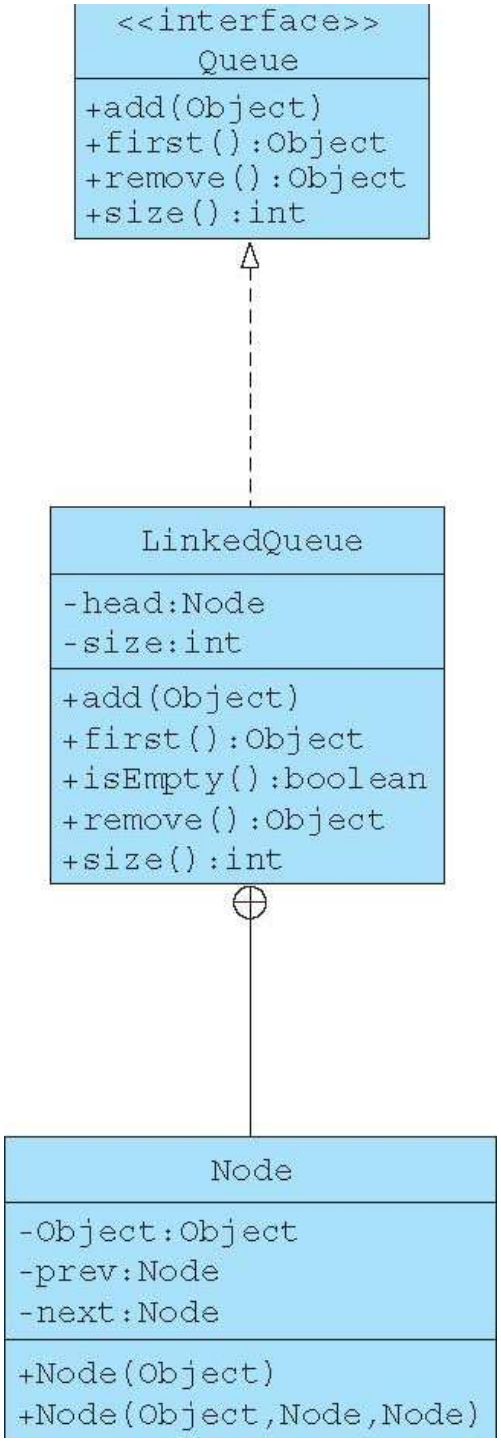
- Queue 인터페이스를 구현하는 방법
  - 배열 구현
  - 연결 구현

- 배열구현에 대한 연결구현의 장점
  - 구현이 더 빠르다.
  - 공간을 낭비하지 않는다.

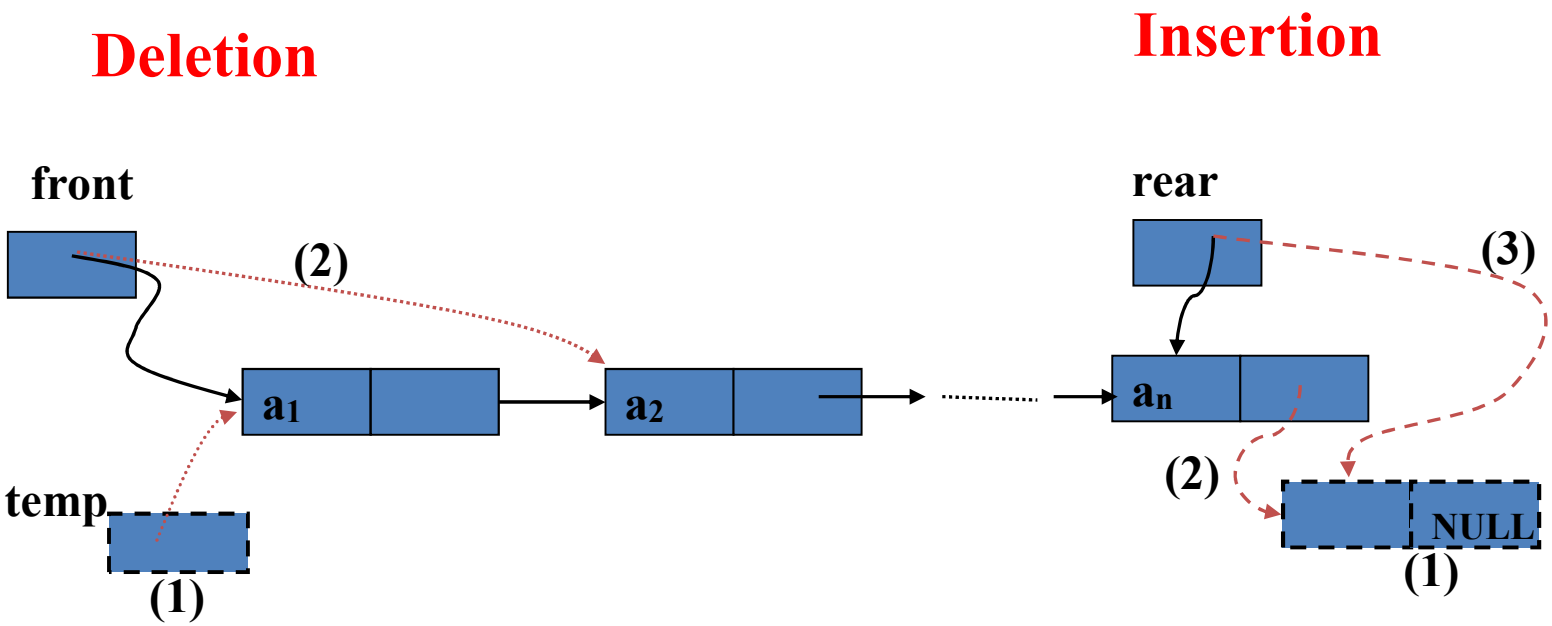
- 연결 구현이 더 빠른 이유
  - 삽입과 삭제를 위한 위치가 항상 동일하게 뒤와 앞이기 때문이다.

- 연결 구현이 공간을 낭비하지 않는 이유
  - 제거된 노드가 자동 쓰레기 수집 프로세스에 의해서 삭제되기 때문이다.

큐 구현


<<interface>>
Queue
+add(Object)
+first():Object
+remove():Object
+size():int
LinkedQueue
-head:Node
-size:int
+add(Object)
+first():Object
+isEmpty():boolean
+remove():Object
+size():int
Node
-Object:Object
-prev:Node
-next:Node
+Node(Object)
+Node(Object,Node,Node)

Queue by Singly Linked List



Singly Linked Queue

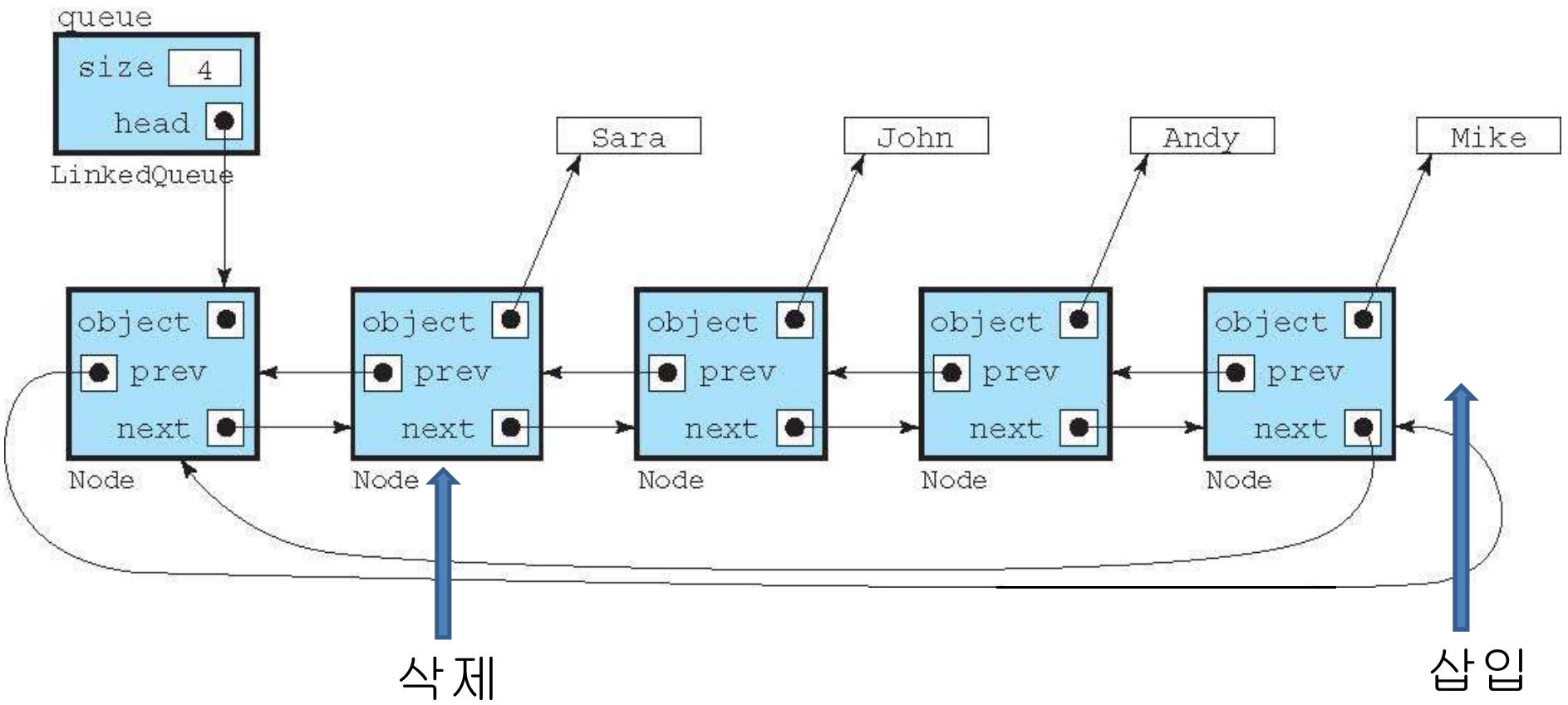
```
1  Public class SLinkedQueue implements Queue {
2      private Node head;
3      private Node rear;
3      private int size;

5      public void add(Object object) {
6          ???
7          ++size;
8      }
10     public Object first() {
11         if (size==0) throw new IllegalStateException("the queue
           is empty");
12         return head.object;
13     }
15     public boolean isEmpty() {
16         return size==0;  }
```

```
19   public Object remove( ) {
20       if (size==0) throw new IllegalStateException("queue is empty");
21       ???
24       --size;
25      return object;
26   }
28    public int size() {  return size;  }
32
     private static class Node {
33       Object object;
34       Node next;
36       Node(Object object) {    this.object=object; }
40       Node(Object object, Node next) {
41            this.object=object;
43            this.next=next;
44       }
45    }
46 }
```

Queue by Double Linked List

빈 헤드 노드를 가진 원형 이중 연결 리스트로 구현



Doubly Linked Queue

```
Public class LinkedQueue implements Queue {
    private Node head = new Node(null);
    private int size;
    public void add(Object object) {
        head.prev = head.prev.next =
            new Node(object, head.prev, head);
        ++size;
    }
    public Object first() {
        if (size==0) throw new IllegalStateException("the queue is empty");
        return head.next.object;
    }
    public boolean isEmpty() {
        return size==0;
    }
```

```
19  public Object remove( ) {
20      if (size==0) throw new IllegalStateException("queue is empty");
21      Object object=head.next.object;
22      head.next = head.next.next;
23      head.next.prev = head;
24      --size;
25      return object;
26  }
28  public int size() {  return size;  }
32  private static class Node {
33      Object object;
34      Node prev=this, next=this;
36      Node(Object object) {   this.object=object; }
40      Node(Object object, Node prev, Node next) {
41          this.object=object;
42          this.prev=prev;
43          this.next=next;
44  } } }
```

6.4 사례 연구: 큐를 이용한 시뮬레이션

- 큐를 이용한 예 : 클라이언트/서버 시스템
  - 컴퓨터 시스템
    - 클라이언트 : 웹 페이지를 요청하는 사용자 컴퓨터
    - 서버 : 요청을 받은 컴퓨터. 다른 컴퓨터에 클라이언트 될 수 있음
  - 실세계
    - 클라이언트 : 요금 정산소에 도착하는 차량들
    - 서버 : 요금 정산소

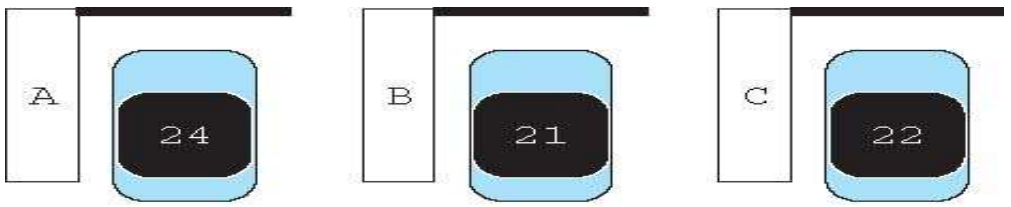
- *객체-지향 컴퓨터 시뮬레이션(object-oriented computer simulation) :* 사건을 객체가 관리하는 시뮬레이션 작업
  - 1. 객체를 식별.
  - 2. 사건을 식별.
  - 3. 알고리즘을 유도.
  - 4. 알고리즘을 구현.
  - 5. 객체들에 대한 인터페이스를 정의.
  - 6. 다른 클래스들을 정의.

사례연구: 톨게이트 구현

- 객체들
  - 클라이언트 (자동차)
  - 서버 (요금 정산소)
  - 큐

- 사건
  - client가 큐에 도착.
  - server가 클라이언트에게 서비스를 시작.
  - server가 클라이언트에 대한 서비스를 종료.

3개 요금 정산소를 가진 클라이언트/서버 시스템



요금 정산소

큐에 대기중

클라이언트/서버 시뮬레이션 알고리즘

1. 시간 t = 0, 1, …에 대해 단계 2에서 6을 반복.
2. 만약 t = ‘다음 도착 시간’이면, 단계 3-5를 수행.
3. 새로운 client를 생성.
4. client를 큐에 삽입.
5. time을 다음 도착으로 설정.
6. 각 서버에 대해 단계 7과 8을 반복.
7. t = ‘server가 서비스를 종료할 시간’이면, 서비스를 중단.
8. 만약 server가 유휴 상태이고 큐가 공백이 아니면, 단계 9-10을 수행.
9. client를 큐에서 제거.
10. server에게 client에 대한 서비스를 시작하도록 알려줌.

클라이언트/서버 시뮬레이션

- LISTING 6.3: Client/Server Simulation

```
1 for (int t=0; ; t++) {
2    if (t==nextArrivalTime) {
3          Client client = clients[i++] = new SimClient(i,t);
4          queue.add(client);
5          nextArrivalTime = t + randomArrival.nextInt();   }
7    for (int j=0; j<numServers; j++) {
8          Server server = servers[j];
9          if (t==server.getStopTime()) server.stopServing(t);
10         if (server.isIdle() && !queue.isEmpty()) {
11            Client client = (SimClient)queue.remove();
12            server.startServing(client,t);
13         }
14   }
15 }
```

서버와 클라이언트에 대한 인터페이스

- LISTING 6.4: An Interface for Servers

```
1  public interface Server {
2      public int getMeanServiceTime();
3      public int getStopTime();
4      public boolean isIdle();
5      public void startServing(Client client, int t);
6      public void stopServing(int t);
7  }
```

- LISTING 6.5: An Interface for Clients

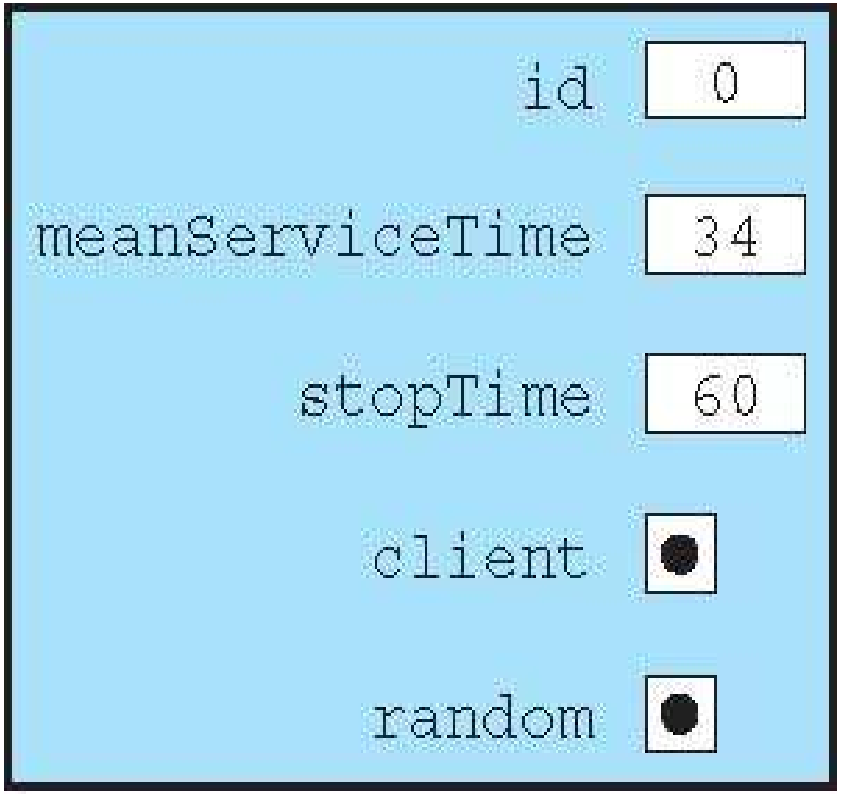
```
1  public interface Client {
2      public void setStartTime(int t);
3      public void setStopTime(int t);
4  }
```

서버 클래스

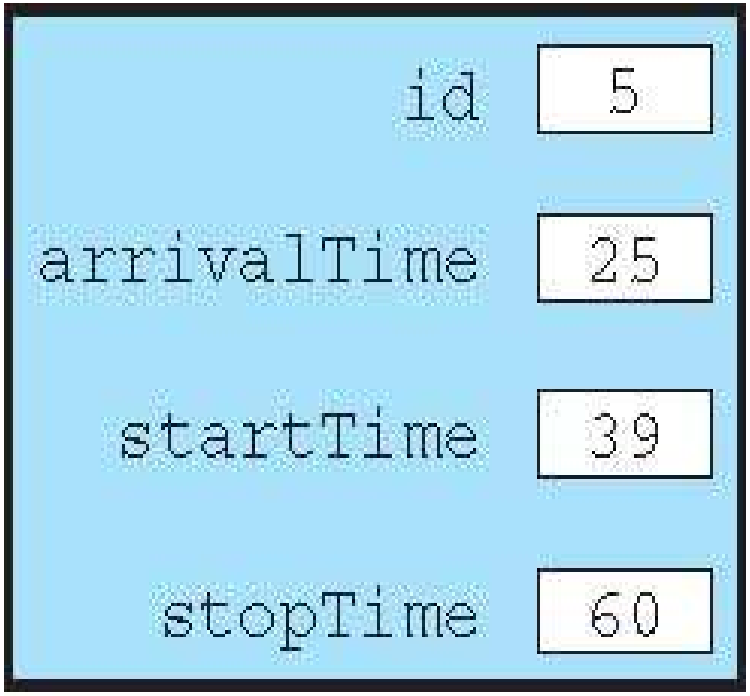
```
1  public class SimServer implements Server {
2      private Client client;
3      private int id, meanServiceTime, stopTime=-1;
4      private java.util.Random random;
6      public SimServer(int id, int meanServiceTime) {
7          this.id = id;
8          this.meanServiceTime = meanServiceTime;
9          this.random = new ExponentialRandom(meanServiceTime);
10     }
12     public int getMeanServiceTime() {
13         return meanServiceTime;    }
16     public int getStopTime() {
17         return stopTime;  }
20     public boolean isIdle() {
21         return client==null;  }
```

```
24    public void startServing(Client client, int t) {
25        this.client = client;
26        this.client.setStartTime(t);
27        this.stopTime = t + random.nextInt();
28        System.out.println(this + " started serving " + client
29                + " at time " + t + " and will finish at time " + stopTime);
30    }
32    public void stopServing(int t) {
33        client.setStopTime(t);
34        System.out.println(this+ " stopped serving " + client
35                + " at time " + t);
36        client = null;
37    }
39    public String toString() {
40        String s="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
41        return "Server " + s.charAt(id);
42   }
43 }
```

SimServer (simulated server) 객체



SimClient (simulated client) 객체



클라이언트 클래스

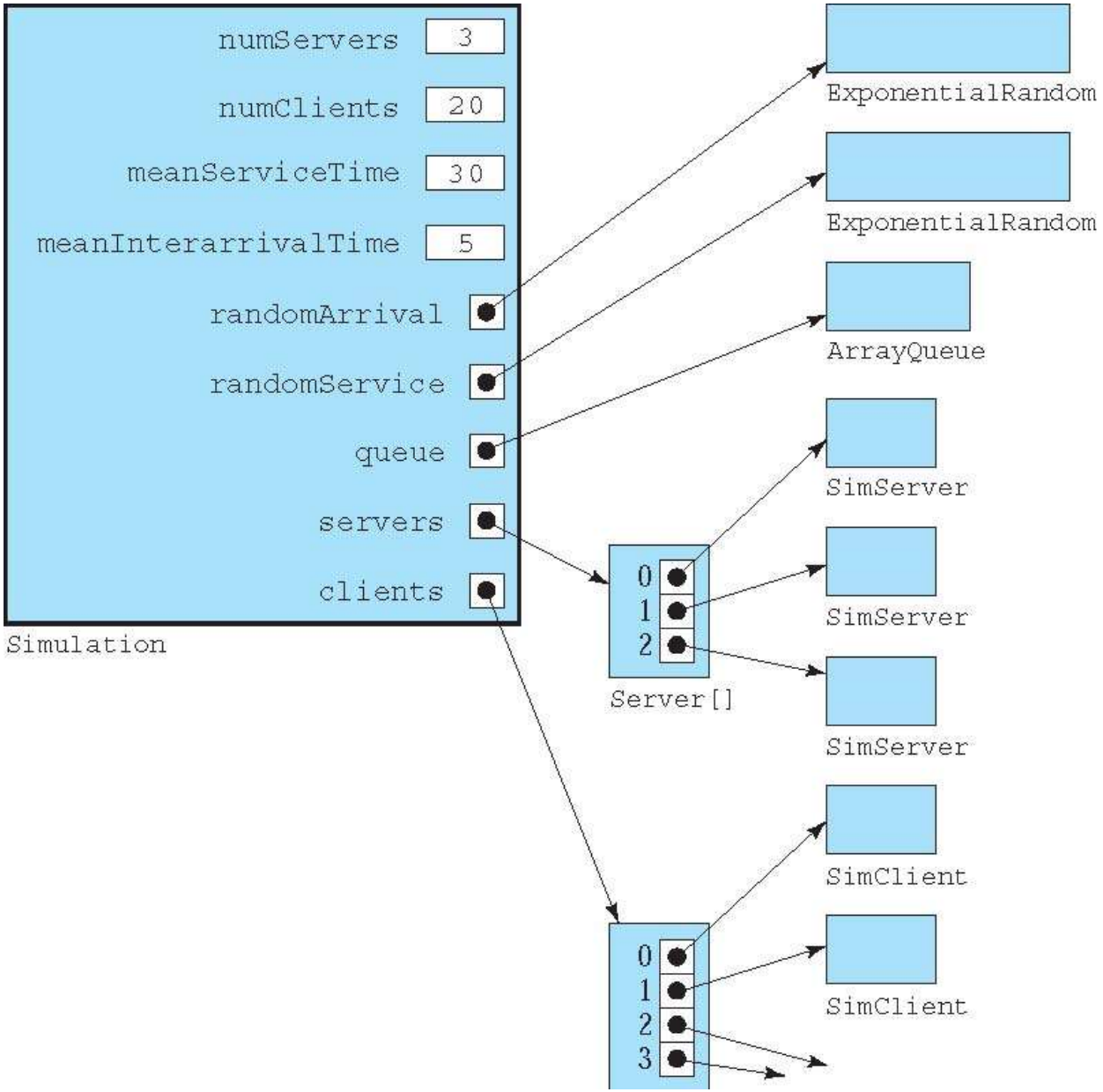
```
1   public class SimClient implements Client {
2       int id, arrivalTime=-1, startTime=-1, stopTime=-1;
4       public SimClient(int id, int t) {
            this.id = id;
            arrivalTime = t;
            System.out.println(this + " arrived at time " + t); }
10      public int getStartTime() {
            return startTime;  }
14      public int getStopTime() {
            return stopTime;  }
18      public void setStartTime(int t) {
            startTime = t;  }
22      public void setStopTime(int t) {
            stopTime = t;  }
26      public String toString() {
            return "Client " + id;   }
29  }
```

지수 분포에 대한 난수 클래스

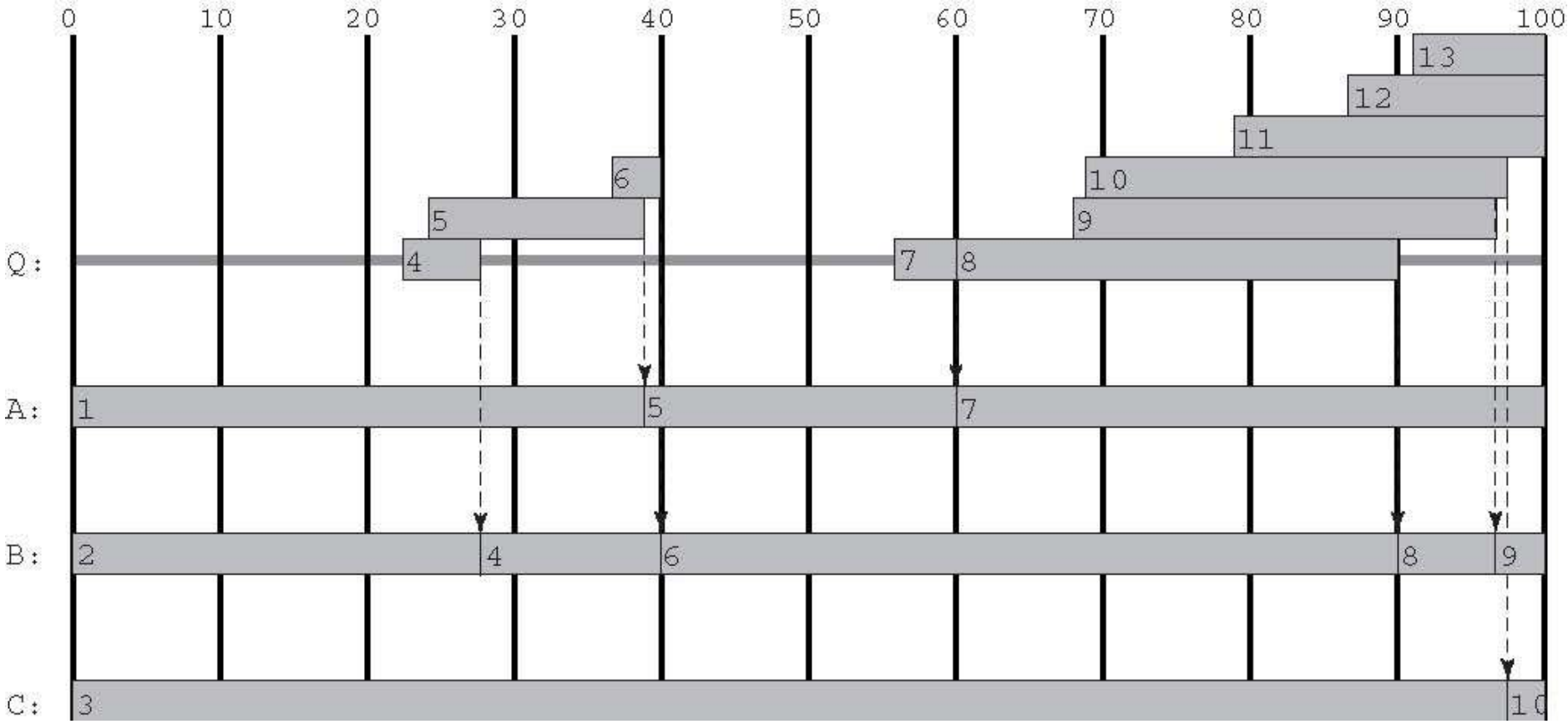
```
1  public class ExponentialRandom extends java.util.Random {
2      private double mean;

4      public ExponentialRandom(double mean) {
5          super(System.currentTimeMillis());
6          this.mean = mean;
7      }
9      public double nextDouble() {
10         return -mean*Math.log(1.0-super.nextDouble());
11     }
13     public int nextInt() {
14         return (int)Math.ceil(nextDouble());
15     }
16 }
```

시뮬레이션 객체들


numServers
3
numClients
20
meanServiceTime
30
meanInterarrivalTime
5
randomArrival
randomService
queue
servers
clients
Simulation
ExponentialRandom
ExponentialRandom
ArrayQueue
SimServer
SimServer
SimServer
0
1
2
Server[]
SimClient
SimClient
0
1
2
3

도착과 출발



시뮬레이션 클래스 [리스팅6.9]

```
1 public class Simulation {
2       static int numServers;
3       static int numClients;
4       static int meanServiceTime;
5       static int meanInterarrivalTime;
6       static Server[] servers;
7       static Client[] clients;
8       static Queue queue = new ArrayQueue();
9       static java.util.Random randomArrival;
10      static java.util.Random randomService;

12      public static void main(String[] args) {
13          init(args);
14           //See listing 6.3
15      }
```

시뮬레이션 클래스 [리스팅6.9]

```
17    static void init(String[] args) {
18        if (args.length<4) {
19            System.out.println("Usage: java Simulation <numServers> “
20            + "<numClients> <meanServiceTime><meanInterarrivalTime>");
21           System.out.println(" e.g.: java Simulation 3 100 12 4");
22            System.exit(0);
23         }
24        numServers = Integer.parseInt(args[0]);
          numClients = Integer.parseInt(args[1]);
          meanServiceTime = Integer.parseInt(args[2]);
          meanInterarrivalTime = Integer.parseInt(args[3]);
          servers = new Server[numServers];
          clients = new Client[numClients];
```

```
        randomService = new ExponentialRandom(meanServiceTime);
        randomArrival = new ExponentialRandom(meanInterarrivalTime);
        queue = new ArrayQueue();
        for (int j=0; j<numServers; j++)
            servers[j] = new SimServer(j,randomService.nextInt());
        System.out.println(" Number of servers = " + numServers);
        System.out.println(" Number of clients = " + numClients);
        System.out.println(" Mean service time = " + meanServiceTime);
        System.out.println("Mean interarrival time = "
             + meanInterarrivalTime);
        for (int j=0; j<numServers; j++)
            System.out.println("Mean service time for " + servers[j]
              + " = "+ servers[j].getMeanServiceTime());
42      }
43}
```

출력 결과-1

- Number of servers = 3
  Number of clients = 20
  Mean service time = 30

- Mean interarrival time = 5
  Mean service time for Server A = 34
  Mean service time for Server B = 19
  Mean service time for Server C = 78
  Client 1 arrived at time 0
  The queue has 1 clients
  The queue has 0 clients
  Server A started serving Client 1 at time 0 and will finish at time 39
  Client 2 arrived at time 6
  The queue has 1 clients
  The queue has 0 clients
  Server B started serving Client 2 at time 6 and will finish at time 28
  Client 3 arrived at time 10

출력 결과-2

The queue has 1 clients
The queue has 0 clients
Server C started serving Client 3 at time 10 and will finish at time 98
Client 4 arrived at time 23
The queue has 1 clients
Client 5 arrived at time 25
The queue has 2 clients
Server B stopped serving Client 2 at time 28
The queue has 1 clients
Server B started serving Client 4 at time 28 and will finish at time 40
Client 6 arrived at time 37
The queue has 2 clients
Server A stopped serving Client 1 at time 39
The queue has 1 clients

출력 결과-3

Server A started serving Client 5 at time 39 and will finish at time 60
Server B stopped serving Client 4 at time 40
The queue has 0 clients
Server B started serving Client 6 at time 40 and will finish at time 90
Client 7 arrived at time 56
The queue has 1 clients
Client 8 arrived at time 60
The queue has 2 clients
Server A stopped serving Client 5 at time 60
The queue has 1 clients
Server A started serving Client 7 at time 60 and will finish at time 149
Client 9 arrived at time 68
The queue has 2 clients
Client 10 arrived at time 71
The queue has 3 clients