

2. 추상 데이터 타입

2.1 구체 데이터 타입

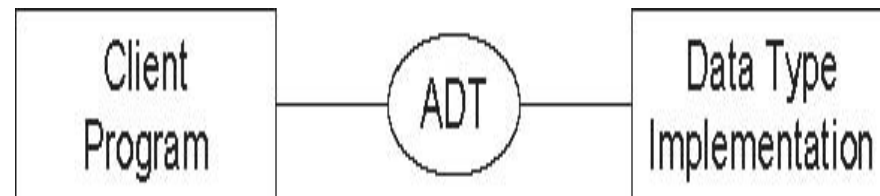
- Java
 - 강한 타입(strongly typed)의 프로그래밍 언어
 - 변수에 저장될 데이터는 특정 데이터 타입을 갖도록 명시적으로 선언되어야 함
 - 예
 - `int n = 44;`
 - `String s="Hello!";`
 - **Java의 데이터 타입**
 - 프리미티브 데이터 타입
 - 인터페이스
 - 클래스
 - 배열 타입

2.2 추상화

- 추상(화)
 - 상이한 그러나 구체적인 인스턴스들에 대해 적용되는 일반적인 개념에 초점 맞춤
 - 한 추상화는 여러 인스턴스에 적용됨
 - 예: $2 + 3 = 5$
 - 두 사람 더하기 세 사람
 - 두 열쇠 더하기 세 열쇠
 - 2개 CD 더하기 3개 CD
 - 두 손가락 더하기 세 손가락
- 추상화는 Java와 같은 프로그래밍 언어에서 데이터타입을 구현하기 전에 그에 대한 명확한 정의를 얻기 위해 사용한다.

2.3 추상 데이터 타입 (ADT: Abstract Data Type)

- 클라이언트 프로그램과 데이터 타입의 구현간의 인터페이스



- 클라이언트 프로그래머에게 데이터 타입의 어떤 연산을 사용할 수 있는지만을 알려줌;
그 연산들이 실제로 구현되는 방법은 숨김
-> **정보 은폐(information hiding)**
 - 예: String 타입의 사용자가 문자열의 길이를 구하려면 length() 연산을 호출하면 됨; 길이의 결정 방법은 모르게 됨

- 데이터 타입과 연산에 대한 일반적인 명세
 - 연산이 무엇인지를 명세; 그러나, 그 연산들이 실행되는 방법에 대한 세부 사항은 제공하지 않음
- 데이터 추상화
 - 객체 지향 프로그래밍 방법론의 기반
 - 데이터와 연산을 독립적으로 구현될 수 있는 별도의 모듈로 캡슐화
 - > 대형 프로젝트 개발을 용이하게 만들어 줌
- ADT 예
 - 백(bag): 중복 객체들을 포함할 수 있는 컨테이너 (그림 2.3)
 - 집합(set): 중복 원소를 허용하지 않는 컨테이너 (그림 2.4)
 - 각 ADT는 2개의 수정자(mutator) 연산과 4개의 접근자(accessor) 연산을 명세

ADT: Bag

A Bag is a collection of objects

boolean contains(Object object)

후조건: 백은 변경되지 않음

리턴: 주어진 객체가 백에 있으면 참

Object getFirst()

리턴: 백의 원소

후조건: 백은 변경되지 않음

Object getNext()

리턴: 마지막으로 호출된 getFirst()와 이어서 호출된 getNext()에 의해 이미 리턴 된 것이 아닌 백의 일부 원소. 모든 원소들이 이전의 호출들에 의해 모두 접근 된 경우에는 null을 리턴.

후조건: 백은 변경되지 않음

integer size()

후조건: 백은 변경되지 않음

리턴: 백의 원소의 수

void add(Object object)

후조건: 주어진 객체가 이 백에 있음

boolean remove(Object object)

리턴: 백이 변경되었으면 참

- ADT의 정의는 알고리즘에서 사용하기에 충분한 정보를 제공한다.

Printing a *Bag*

Input: a *Bag* *b*.

Output: a printed copy of all the elements of *b*.

Postcondition: the bag *b* is unchanged.

1. Let *x* be the object returned by *getFirst()* for the bag *b*.
2. Repeat steps 3-5:
3. If *x* is *null*, return.
4. Print *x*.
5. Let *x* be the object returned by *getNext()* for the bag *b*.

ADT: Set

boolean contains(Object object)

후조건: 집합은 변경되지 않음

리턴: 주어진 객체가 집합에 있으면 참

Object getFirst()

리턴: 집합의 원소

후조건: 집합은 변경되지 않음

Object getNext()

리턴: 마지막으로 호출된 getFirst()와 이어서 호출된 getNext()에 의해 이미 리턴된 것이 아닌 집합의 일부 원소. 모든 원소들이 이전의 호출들에 의해 모두 접근된 경우에는 null을 리턴.

후조건: 집합은 변경되지 않음

integer size()

후조건: 집합은 변경되지 않음

리턴: 집합의 원소의 수

boolean add(Object object)

후조건: 주어진 객체가 이 집합에 있음

리턴: 집합이 변경되었으면 참

boolean remove(Object object)

후조건: 주어진 객체는 이 집합에 존재하지 않음

리턴: 집합이 변경되었으면 참

2.4 선조건과 후조건

- 연산의 구성
 - 선조건, 후조건, 리턴값
- 선조건 (precondition)
 - 연산의 시작 전에 참으로 가정되는 조건
- 후조건 (postcondition)
 - 연산의 종료 후 참인 것이 보장되는 조건
- 선조건과 후조건은 클라이언트와 연산 간의 계약임
 - 클라이언트가 선조건을 만족시키면 연산은 후조건의 만족을 보장함
- Bag과 Set ADT: 후조건만을 가짐
- Date ADT (그림 2.7): 선조건과 후조건을 모두 가짐

IntsDate (달력 날짜)

x

day	4
month	7
year	1776

IntsDate

ADT: Date

A Date represents a calendar date, such as July 4, 1776.

integer getDay()

후조건: 날짜는 변경되지 않음

리턴: 이 날짜의 일

integer getMonth()

후조건: 날짜는 변경되지 않음

리턴: 이 날짜의 월

integer getYear()

후조건: 날짜는 변경되지 않음

리턴: 이 날짜의 년도

void setDay(integer day)

선조건: $1 \leq \text{day} \leq 31$

후조건: 이 날짜의 일은 주어진 값을 가짐

void setMonth(integer month)

선조건: $1 \leq \text{month} \leq 12$

후조건: 이 날짜의 월은 주어진 값을 가짐

void setYear(integer year)

선조건: $1700 \leq \text{year} \leq 2100$

후조건: 이 날짜의 년도는 주어진 값을 가짐

- 수정자(mutator)
 - 객체의 상태를 변경시킬 수 있는 연산자
- 접근자(accesor)
 - 변경시킬 수 없는 연산
- 설정자 (setter) 연산
 - setX 형태
 - void 리턴 타입으로 되어 있어 아무것도 리턴하지 않는 특별한 수정자
 - 정보는 클라이언트로부터 객체쪽으로 일방으로 흐름
 - 수정자의 특별한 형태
- 판독자 (getter) 연산
 - getX 형태
 - 인자가 없어 아무것도 수신하지 않는 특별한 접근자
 - 정보는 객체로부터 클라이언트 방향으로 일방으로 흐름
 - 접근자의 특별한 형태

2.5 알고리즘에서 ADT의 사용

- ADT는 데이터 타입의 연산적 정의(operational definition)를 제공
- ADT의 구현 코드에서 데이터 타입에 대해 사용하게 될 실제 저장 구조를 결정
 - 이러한 구현 세부 사항은 문제를 풀기 위한 알고리즘에서 ADT를 사용할 때는 필요하지 않음
- 예: Bag ADT에 대한 클라이언트 알고리즘
 - Bag에서 한 Object의 모든 인스턴스를 제거: 알고리즘 2.2
 - 이 알고리즘은 ADT Bag 만을 필요로 함
 - 알고리즘을 명세하고 분석하는 단계에서는 완전히 구현된 데이터 타입이 필요하지 않음

Bag에서 한 Object의 모든 인스턴스를 제거

- ALGORITHM 2.2: Remove All Instances of an Object from a *Bag*

입력: Bag b; 객체 x

출력: 제거된 객체의 개수

후조건: 백 b는 객체 x를 포함하지 않음

1. n을 정수 0으로 놓음.
2. contains(x)가 백 b에 대해 거짓이면, n을 리턴.
3. b에서 remove(x).
4. n에 1을 더함.
5. 단계 2로 이동.

2.6 구체 데이터 타입

- ADT의 목적
 - 실제 프로그래밍 언어에서 객체를 선언하기 위해 사용되는 구체 데이터 타입의 정의와 구현을 용이하게 해주는 것
- 실체화: ADT -> 인터페이스 -> 클래스 -> 객체
 - ADT Bag의 Java 인터페이스로의 실체화: 리스팅 2.1
 - Bag 인터페이스를 배열을 이용해 클래스로 구현: 리스팅 2.3
 - 객체를 생성해서 클래스를 인스턴스화
 - Bag bag = new ArrayBag();
 - bag.add("CA");

Bag ADT에 대한 인터페이스

- LISTING 2.1: An Interface for a *Bag* ADT

```
1 public interface Bag {  
2     public void add(Object object);  
3     public boolean contains(Object object);  
4     public Object getFirst();  
5     public Object getNext();  
6     public boolean remove(Object object);  
7     public int size();  
8 }
```


Interface를 사용하는 예

- LISTING 2.2: Removing Elements from a Bag

```
1 public int remove(Object object, Bag bag) {  
2     int n=0;  
3     while (bag.remove(object))  
4         ++n;  
5     return n;  
6 }
```

Implementing an Interface

- In Java, classes *implement* interfaces.
- The interface specifies *what* the type can do.
- The class specifies *how* the type does it.
- This is done by adding a block of executable statements (the body) to each method header.
- The class also specifies the data structure(s).

Bag 인터페이스의 ArrayBag 구현

- LISTING 2.3:

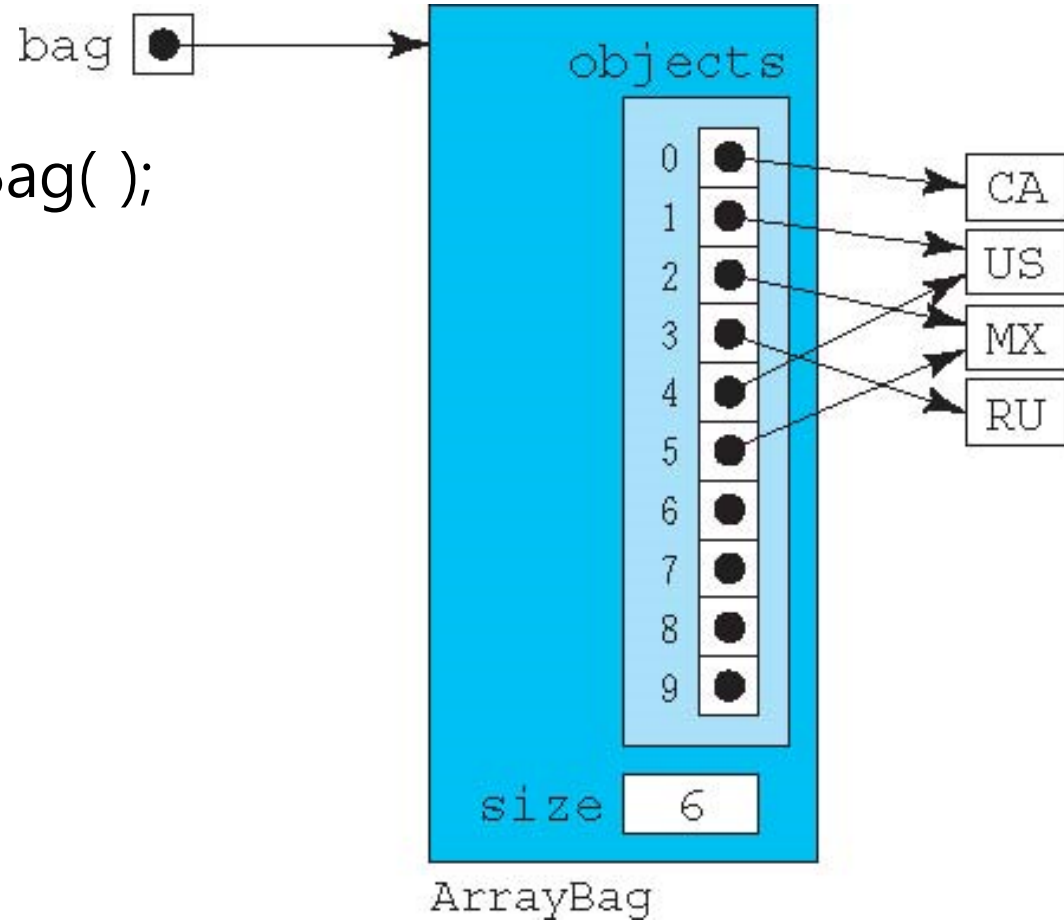
```
1 public class ArrayBag implements Bag {  
2     private Object[] objects = new Object[1000];  
3     private int size, i;  
4  
5     public void add(Object object) {  
6         objects[size++] = object;  
7     }  
8  
9     public boolean contains(Object object) {  
10         for (int i=0; i<size; i++)  
11             if (objects[i]==object) return true;  
12         return false;  
13     }  
14 }
```

```
15  public Object getFirst() {
16      i = 0;
17      return objects[i++];
18  }
20  public Object getNext() {
21      return objects[i++];
22  }
24  public boolean remove(Object object) {
25      for (int i=0; i<size; i++)
26          if (objects[i]==object) {
27              System.arraycopy(objects, i+1, objects, i, size-i-1);
28              objects[--size] = null;
29              return true;
30          }
31      return false;
32  }
```

```
34  public int size() {  
35      return size;  
36  }  
37}
```

ArrayBag 객체

```
Bag bag = new ArrayBag( );  
bag.add("CA");  
bag.add("US");  
bag.add("MX");  
bag.add("RU");  
bag.add("US");  
bag.add("MX");
```

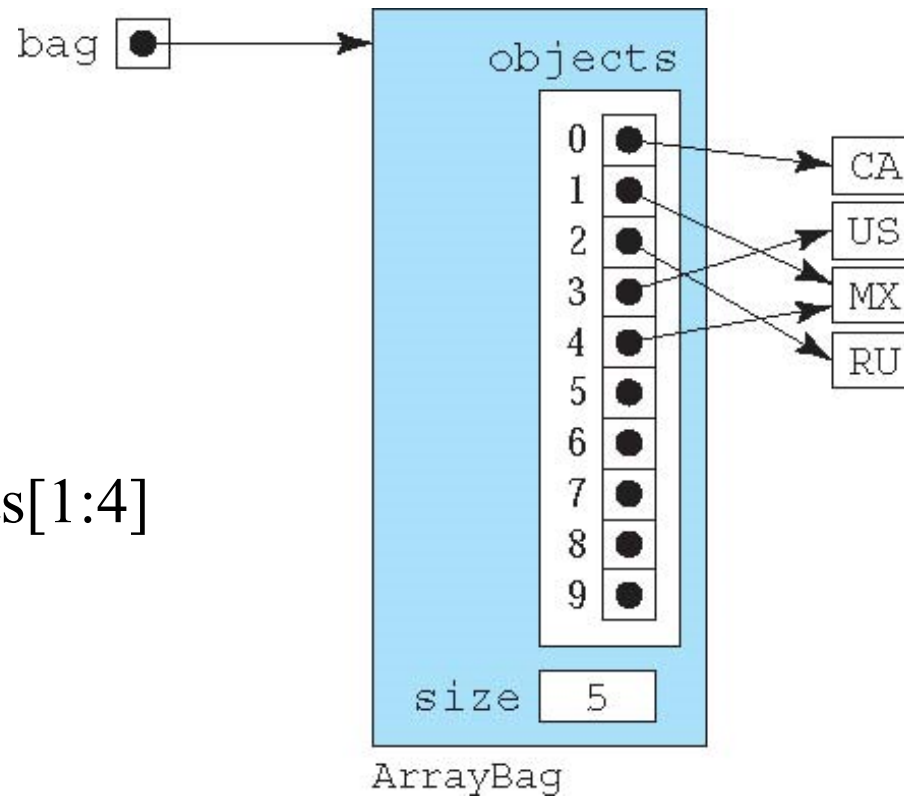


ArrayBag 객체

`bag.remove("US");`

호출시

`arraycopy` 메소드가
`objects[2:5]`를 `objects[1:4]`
의 위치로 이동시킴



실행 후의
ArrayBag 객체