자료구조: 2022년 1학기 [강의]

Bag, Set, List

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

Bag / Set / List

비슷하면서도 다른 세 가지 [1]

- 비슷한 점
  - 원소를 모아 둔다
  - 자주 사용된다
  - 구현할 때 유사한 내부 구조를 사용한다
    - 배열, 연결 체인

- 다른 점
  - 원소의 중복이 가능한지
  - 원소들의 순서가 필요한지
  - 그에 따라서, 사용하는 방법이 조금씩 다르다

비슷하면서도 다른 세 가지 [2]

- 생각할 점
  - 이런 차이점을 어떻게 추상화시켜 표현하면 좋을까?
    - 사용법이 어떻게 다를까?
  - 구현자가 구현은 어떻게 해야 할까?
    - 사용법이 다르면, 각각의 효율적인 구현 방법에 어떤 영향을 줄까?
    - 구체적인 효율적인 방법은?
- 이런 점들을 생각하면서, 다음을 보자...

Bag 이란?

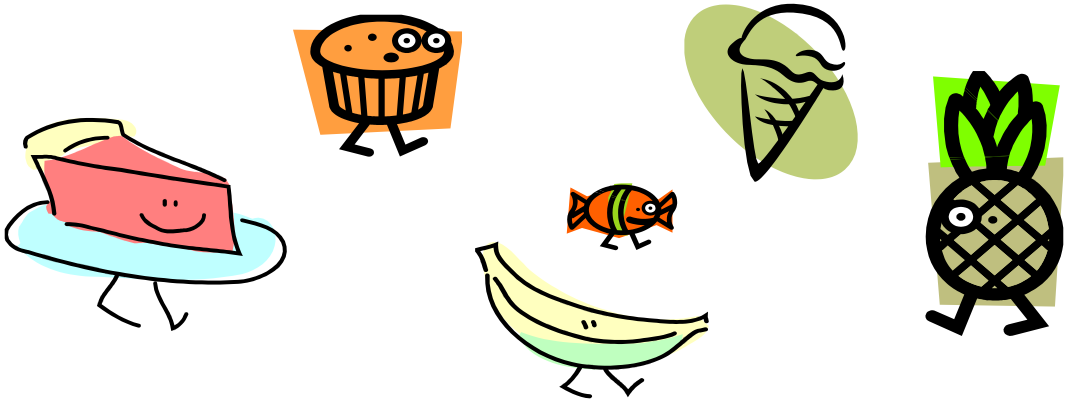
- 원소들을 단순히 모아 놓은 것
  - 원소들 사이에 아무 순서가 없다.
  - 중복된 원소들도 있을 수 있다.

- 예: 내가 구매한 기호 식품들

Set 이란?

- 원소들의 집합
  - 원소들 사이에 아무 순서가 없다.
    - 원소들의 순서에 의미를 둘 필요가 없을 경우
  - 중복된 원소는 존재하지 않는다.

- 예: 기호 식품 종류



List 란?

- 원소들이 순서 있게 나열되어 있다
  - 원소들의 순서가 중요한 경우
  - 원소가 중복될 수도 있다

- 예: 내가 좋아하는 기호 식품 구매 순서

Generic Class

Why Generic Class ? [1]

- 창고 짓는 설계도를 그렸다:
  - 이 설계도가 컴퓨터 모니터 전용 창고 용도라면,
    이 설계도로 지은 창고는 컴퓨터 모니터만 보관 가능.

- 그런데, 모니터나 키보드나 보관 방법이 다르지 않다면?
  - 모니터 보관용 창고 설계도로, 키보드 보관용 창고도 지을 수 있을 것이다.

- 하나의 창고 설계도로 다양한 type 의 객체를 보관할 수 있는 창고를 지을 수 있다.

Why Generic Class ? [2]

- 예를 들어, class "ArrayList" 는 담아야 할 원소의 종류가 무엇이든지 그 기능은 동일한, 그러한 객체들의 설계도 이다.

- 그렇다면, 실제 생성된 ArrayList (창고)에 어떤 자료형의 원소를 담을 지는, 설계할 때 미리 결정해 놓을 것이 아니라, "ArrayList"를 사용하는 시점에 결정할 수 있다면 편리할 것이다.

Why Generic Class ? [3]

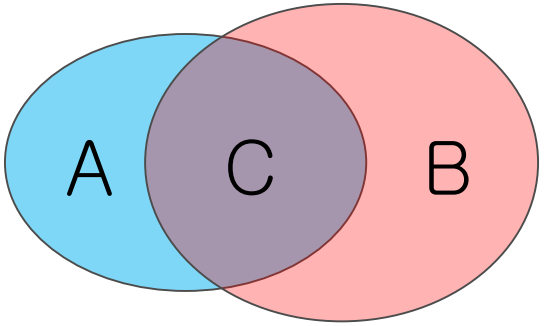
- 만일, "ArrayList" 를 정의하는 시점에 담을 원소의 자료형을 고정시켜야만 된다면 ?
  - 그 "ArrayList" 는 고정된 한 type 의 객체 만을 담을 수 있는 "ArrayList" 가 될 것이다.
  - 또한 우리는 원소의 class 마다 별도의 "ArrayList" 를 만들어 사용해야 할 것이다.
    - ArrayListForMoniter, ArrayListForKeyboard, ...

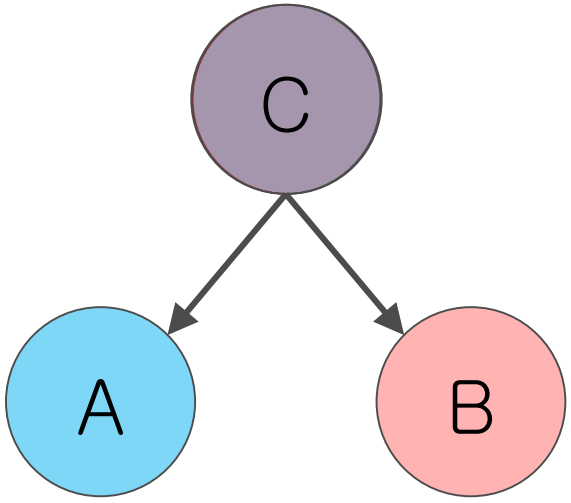
- 담아야 할 객체들의 다양한 type 마다,
  - 별도의 설계도를 작성하는 것이 아니다.
  - 하나의 설계도만 작성하면 된다.

- 그러므로, generic class 를 사용하는 것은, 코드의 생산성을 높이게 된다.

Why Generic Class ? [4]

- 설계도가 유사하지만, 똑같지는 않다면?
  - 이 문제는, generic class 의 문제는 아니다.
  - (설계도) 의 상속 (inheritance) 을 고려해 본다.
    - 공통 부분은 상위 class 로 선언.
    - 다른 부분은 상위 class 를 상속받은 class 로 구현한다.





Generic Class:

- Java 는, 클래스를 설계할 때 그 내부에서 사용되는 자료형을 특정하지 않은 채로 통칭적인 (generic) 그래서 형식적인 식별자 (identifier) 를 사용할 수 있게 해 준다.
  - 이러한 형식적인 식별자로 정의되는 자료형을 generic type 이라고 한다.
  - 이 generic identifier 는 나중에 실제로 사용하는 시점에 실제로 존재하는 자료형 이름으로 대체가 된다.

- (예) 다음과 같은 식별자를 종종 사용한다. 그러나 식별자의 이름은 사용자가 임의로 정할 수 있다.
  - Class ArrayList <**T**>
    - Class "ArrayList"에서 원소의 type으로 사용되는 generic type "**T**"
  - Class LinkedNode <**E**>
    - Class "LinkedNode"에서 원소의 type으로 사용되는 generic type "**E**"

- 사용하는 관점에서, 동일한 class 일지라도 사용 시점에 generic type 이 다르게 주어진 class 들은 서로 다른 class 라고 할 수 있다:
  - Class "ArrayList<Student>" 와 class "ArrayList<Coin>" 은 서로 다른 class 이다.

Generic Class 를 배열 원소의 자료형으로 사용 [1]

```
public class  ArrayList <T> {
    private T[]  _elements ;


    public  ArrayList () {
       this._elements = (T[]) new Object [100];
    }
}
```

- 객체를 생성하는 new 를 사용할 때에는 generic type 이 아닌, 반드시 구체적인 type 으로 언급되어야 한다.
- Class "ArrayList" 컴파일 하는 시점에, T 의 자료형은 결정되어 있지 않다. 따라서, new 에 T 를 사용하면 컴파일러는 오류를 내보낸다. (오류 메시지: "Cannot create a generic array of T")
- 보통 배열을 생성하는 이런 경우에는, T 대신에 최상위 Class 인 "Object" 를 사용한다. 즉, 아무 원소나 저장할 수 있는 배열을 만드는 것이다.
- 컴파일러는 생성된 배열이 _elements 의 자료형과 일치하지 않다고 여전히 오류 메시지 "Type mismatch: cannot convert from Object[] to T[]" 를 내보낸다.

Generic Class 를 배열 원소의 자료형으로 사용 [2]

```
public class   ArrayList <T> {
    private T[]   _elements ;


    public   ArrayList () {
       this._elements = (T[]) new Object [100] ;
    }
}
```

- 자료형 불일치를 해결하기 위해서, new 앞에 "(T[])" 를 삽입하여 생성된 배열의 자료형을 강제로 T[] 로 변환시킨다.
- 이렇게 하여 컴파일 오류는 제거할 수 있다.
- 그렇지만, 나중에 사용자가 잘못 사용하여 자료형이 T 가 아닌 원소를 배열에 넣을 가능성이 있다. (앞에서 언급하였듯이 생성된 배열은 원소의 자료형이 "Object" 이므로 어떠한 자료형의 객체도 넣을 수 있다.)
- 이렇게 잘못 사용할 가능성이 있으므로 컴파일러는 경고 메시지를 내보낸다. (경고 메시지: "Type Safety: Unchecked cast from Object[] to T[]")

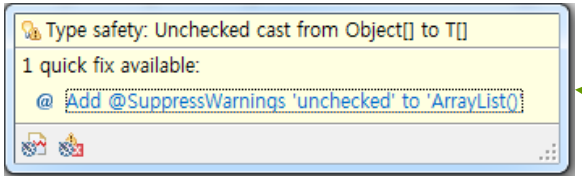
Generic Class 를 배열 원소의 자료형으로 사용 [3]

```
public class   ArrayList <T> {
    private T[]   _elements ;

    @SuppressWarnings("unchecked")
    public   ArrayList () {
       this._elements = (T[]) new Object [100] ;
    }
}
```

- 프로그램 작성자는 이러한 강제적인 자료형 변환으로 인하여 발생할 수 있는 위험성을 인지하고 대처할 수 있는 방안을 가지고 있어야 한다.
- 이를테면, 배열에 들어가는 객체의 자료형은 언제나 T 임 스스로 보장할 수 있고, 또 관련 프로그램 코드를 그에 합당하게 작성할 것임을 스스로 보장할 수 있어야 한다.
- 그렇다면, 프로그램 작성자는, "컴파일러는 이러한 강제적인 자료형 변환을 걱정할 필요가 없으므로, 자료형 변환으로 인한 점검을 하지 않아도 된다" 고 컴파일러 알려 더 이상 경고 메시지가 통보되지 않게 할 수 있다.

Generic Class 를 배열 원소의 자료형으로 사용 [4]



오류를 클릭하면 이와 같은 창이 뜬다. 경고를 해결할 수 있는 방법을 선택하면 컴파일러가 자동으로 필요한 행위를 한다.

```
public class  ArrayList <T> {
    private T[]  _elements ;

    @SuppressWarnings("unchecked")
    public ArrayList () {
       this._elements = (T[]) new Object [100] ;
    }
}
```

- 이러한 통보를 컴파일러에게 하기 위해, @SuppressWarning("unchecked") 를 삽입한다. 프로그램 작성자는 이 통보 사항을 직접 입력하기 보다는 오류를 클릭하여 해결책을 선택하는 방법으로 처리하게 된다.

가방 (Bag)

Class "Bag"

Bag 객체 추상화: 사용법

- Bag 객체 생성과 소멸
- Bag 상태 알아보기
- Bag 내용 알아보기
- Bag 내용 바꾸기

객체를 정의할 때
맨 먼저 생각할 것은 언제나

"객체의 사용법"

Bag 객체 추상화: 사용법

- Bag 객체 생성과 소멸
  - Bag 객체 생성
- Bag 상태 알아보기
  - Bag 에 들어있는 원소의 개수를 알려주시오
  - Bag 이 비어 있는지 알려주시오
  - Bag 이 가득 찾는지 알려주시오
  - 주어진 원소가 Bag 에 있는지 알려주시오
  - 주어진 원소가 Bag 에 몇 개 있는지 알려주시오
- Bag 내용 알아보기
  - Bag 에서 아무 원소 하나를 얻어내시오
- Bag 내용 바꾸기
  - Bag 에 주어진 원소를 넣으시오
  - Bag 에서 아무 원소 하나를 제거하여 얻어내시오
  - Bag 에서 지정된 원소를 찾아서 있으면 제거하시오
  - Bag 을 비우시오

Bag 객체 추상화: 사용법

- Bag 객체 생성과 소멸
  - Bag 객체 생성
- Bag 상태 알아보기
  - Bag 에 들어있는 원소의 개수를 알려주시오
  - Bag 이 비어 있는지 알려주시오
  - Bag 이 가득 찾는지 알려주시오
  - 주어진 원소가 Bag 에 있는지 알려주시오
  - 주어진 원소가 Bag 에 몇 개 있는지 알려주시오
- Bag 내용 알아보기
  - Bag 에서 아무 원소 하나를 얻어내시오
- Bag 내용 바꾸기
  - Bag 에 주어진 원소를 넣으시오
  - Bag 에서 아무 원소 하나를 제거하여 얻어내시오
  - Bag 에서 지정된 원소를 찾아서 있으면 제거하시오
  - Bag 을 비우시오

Bag 객체 추상화: 사용법

- Bag 객체 생성과 소멸
  - Bag 객체 생성
- Bag 상태 알아보기
  - Bag 에 들어있는 원소의 개수를 알려주시오
  - Bag 이 비어 있는지 알려주시오
  - Bag 이 가득 찾는지 알려주시오
  - 주어진 원소가 Bag 에 있는지 알려주시오
  - 주어진 원소가 Bag 에 몇 개 있는지 알려주시오
- Bag 내용 알아보기
  - Bag 에서 아무 원소 하나를 얻어내시오
- Bag 내용 바꾸기
  - Bag 에 주어진 원소를 넣으시오
  - Bag 에서 아무 원소 하나를 제거하여 얻어내시오
  - Bag 에서 지정된 원소를 찾아서 있으면 제거하시오
  - Bag 을 비우시오

Bag 의 사용법은 공개함수로

- Bag 객체 사용법을 Java로 구체적으로 표현해보자
  - public Bag() {...}
  - public int size () {...}
  - public boolean isEmpty () {...}
  - public boolean isFull () {...}
  - public boolean doesContain (E anElement) {...}
  - public int frequencyOf (E anElement) {...}
  - public E any() {...}
  - public boolean add (E anElement) {...}
  - public E removeAny () {...}
  - public boolean remove (E anElement) {...}
  - public void clear () {...}

Class "Bag"의 초기 형태는 이렇게!

```
public class  Bag<E>
{
      public Bag ( ) {
            // 수정해야 함
      }

      public int size ()
      {
            return 0 ; // 수정해야 함
      }
      public boolean       isEmpty ()
      {
            return true ; // 수정해야 함
      }
      public boolean       isFull ()
      {
            return true ; // 수정해야 함
      }
      public boolean       doesContain (E anElement)
      {
            return true ; // 수정해야 함
      }
      public int           frequencyOf (E anElement)
      {
            return 0 ; // 수정해야 함
      }

      public E    any()
      {
            return   null ; // 수정해야 함
      }

      public boolean       add (E anElement) { return true ; /* 수정해야 함 */ }
      public E             removeAny () { return null ; /* 수정해야 함 */ }
      public boolean       remove (E anElement) { return true ; /* 수정해야 함 */ }
      public void          clear () { /* 수정해야 함 */ }

} // End of Class "Bag"
```

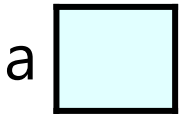
이렇게만 정의해 두어도 사용하는 곳에서 프로그래밍 하는 데는 전혀 지장이 없다. 즉 컴파일 오류가 발생하지 않는다.

Array in Java

- Class “Bag” 구현에 사용 -

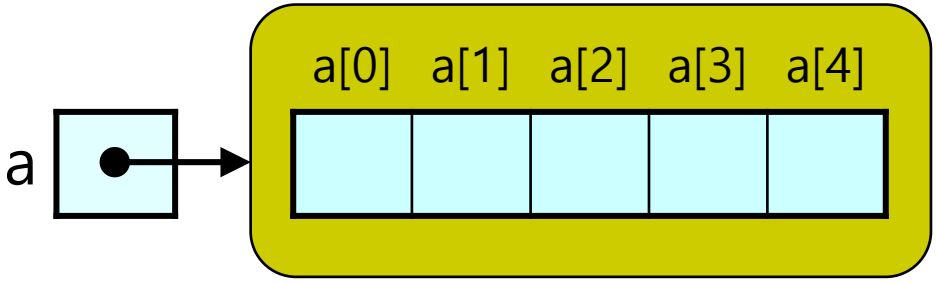
Java 에서의 배열

- 배열은 객체
  - int[] a ;
    - 선언 직후 배열 객체 변수 a는 아무 내용도 가지고 있지 않다



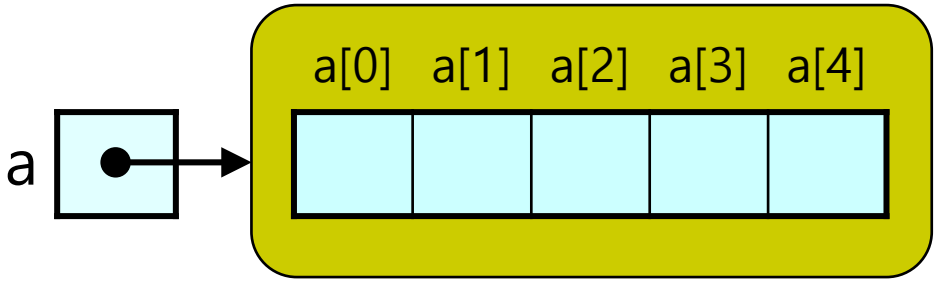
Java 에서의 배열

- 배열은 객체
  - int[] a ;
    - 선언 직후 배열 객체 변수 a는 아무 내용도 가지고 있지 않다
  - a = new int[5] ;



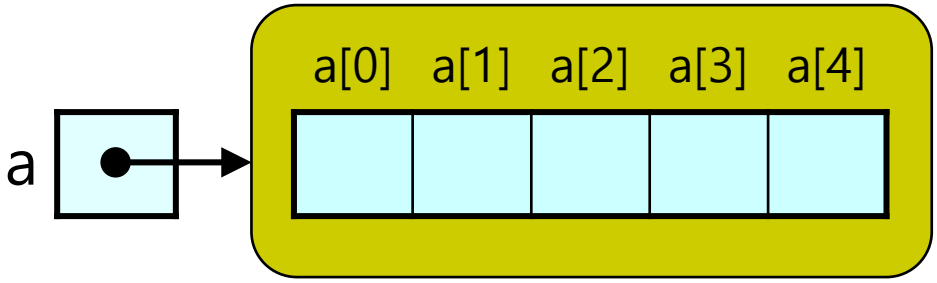
    - new 를 실행한 후에야 배열 객체 변수 a는 배열 객체를 소유하게 된다

Java 에서의 배열 객체 사용법



- 배열의 크기 알아보기
  - int numberOfElements = a.**length** ;
- 특정 위치의 값 얻기 (getter)
  - int value = a[2] ;
  - int currentLocation = 2 ;
    int value = a[currentLocation] ;
- 특정 위치의 값 설정하기 (setter)
  - a[3] = -20 ;
  - int newLocation= 3 ;
    a[newLocation] = -20 ;

Java 에서의 배열: 인덱스



- 배열의 유효한 인덱스 범위를 벗어나면?
  - int value = a[6] ;
  - a[-1] = 10 ;
  - int lastValue = a[a.length] ;
- 예외처리
  - ArrayIndexOutOfBoundsException

End of

"Bag, Set, List"