**객체지향 프로그래밍** | 최윤정 교수님 | 실습#9 Summary 이서현 2276219

**Generic & Collection**

* **Generic:** 제네릭 타입, 멀티 타입 파라미터, 제네릭 메소드, 제한된 타입 파라미터, 와일드카드 타입, 제네릭 타입의 상속과 구현
* **Collection:** List 컬렉션, Set 컬렉션, LIFO/FIFO 컬렉션, Map 컬렉션, 검색 기능을 강화한 컬렉션, 동기화된(synchronized) 컬렉션, 동시실행(Concurrent) 컬렉션

**제네릭(Generic)**

**제네릭 프로그래밍**

 일반적인 코드를 작성하고, 이 코드를 다양한 타입의 객체에 대하여 재사용하는 프로그래밍 기법이다.

 타입을 파라미터로 가지는 클래스와 인터페이스를 정의한다.

* 선언 시 클래스 또는 인터페이스 이름 뒤에 “<T>"  
  <> 사이에는 타입 파라미터\*를 명시한다.

\*타입 파라미터: 일반적으로 대문자 알파벳 한 문자로 표현. (ex- <E, V> : element, value)

- 장점: ‘컴파일 단계’에서 “잘못된 타입”으로 발생할 수 있는 문제를 파악할 수 있다.

컬렉션, 람다식(함수적 인터페이스), 스트림, NIO에서 널리 사용된다.

**제네릭을 사용하는 이유**(1)컴파일 시 강한 '타입 체크' 가능

클래스들의 구조와 연결관계가 복잡해지면서 TypeMiscasting 등의 타입오류들이 발생할 수 있음

컴파일 시에 미리 타입을 강하게 체크하여 에러를 발생시킨다.

(실행 시 런타임-타입 에러가 나는 것 방지.)

(2) 타입명시 / 타입변환을 제거 가능

list에 일관된 타입의 정보를 저장하도록 명시한다.

* List<String> list = new ArrayList<String>(); list.add("aaa"); String str = list.get(0);  
  🡪 제너릭 사용한 경우
* List list = new ArrayList(); list.add("aaa"); String str = (String)list.get(0);  
  🡪 제너릭 안 쓴경우, list의 타입을 정해주지 않아서 타입변환문을 사용해야하는 번거로움

**최상위 클래스인 Object 타입 사용하는 기존 방법**

(제네릭 타입 사용x, 일반적인 객체 처리 방법임) 🡪 빈번한 타입 변환 발생, 프로그램 성능 저하

**제네릭 타입을 사용하는 예**

클래스 선언할 때 타입 파라미터 사용, 컴파일 시 타입 파라미터가 구체적인 타입으로 변경텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

**멀티 타입 파라미터**

두 개 이상의 타입 파라미터도 사용 가능하다. 콤마로 구분. (class<K, V, …> { … }, interface<K, V, …>{…})

**제네릭 메소드**

매개변수 타입과 리턴 타입으로 타입 파라미터를 갖는 메소드.

제네릭 메소드 선언 방법: 리턴 타입 앞에 “<\_>” //\_는 타입 파라미터

* 제네릭 메소드 호출 - 두 가지 방법

1. 명시적으로 구체적 타입 지정  
   리턴타입 변수 = <구체적타입> 메소드명(매개값);  
   Box<Integer> box = <Integer>boxing(100);
2. 매개값을 보고 구체적 타입을 추정  
   리턴타입 변수 = 메소드명(매개값);

Box<Integer> box = boxing(100);

Lab#: 제네릭 클래스/메소드의 예 : ArrayList 미리보기 (Box.java, Main\_Test.java 파일 참고)

|  |
| --- |
| **public** **class** Array {  **public** **static** <T> getLast(T[] a){  **return** a[a.length-1];  }  } |

타입 매개변수의 범위가 메소드 내부로 제한된다.

**제한된 타입 파라미터**

상속 및 구현 관계 이용해 타입 제한

 메소드의 인자 또는 리턴타입에 제한을 둘 수 있다.

**public** <T **extends** 상위타입> 리턴타입 메소드(매개변수){…}

**public** <T **extends** 동물> T getAnimal(){…}

T는 상위타입 제한. 상위 타입은 클래스뿐만 아니라 인터페이스도 가능하다.

 타입 파라미터를 대체할 구체적인 타입을 명시

Box<동물> box1= **new** Box();

Box<강아지> box2= **new** Box();

Box<오리> box3= **new** Box();

같은 상속트리에 있거나, 인터페이스로 연결된 경우만 지정가능

**와일드카드 타입**

와일드카드 타입의 세가지 형태 – 매개변수(인자)에서 허용  
(1) 제네릭타입<?>: Unbounded Wildcards (제한없음)

구체적인 타입, 모든 클래스나 인터페이스 타입 가능

(2) 제네릭타입<? extends 상위타입>: Upper Bounded Wildcards (상위클래스 제한)  
 '상위타입 or 상위타입의 하위타입'만 올 수 있다

(3) 제네릭타입<? super 하위타입>: Lower Bounded Wildcards (하위클래스 제한)  
 '하위타입 or 하위타입의 상위타입'만 올 수 있다

**제네릭 타입의 상속과 구현**

* 제네릭 타입을 부모 클래스로 사용할 경우 타입 파라미터는 자식 클래스에도 기술해야함.
* 추가적인 타입 파라미터 가질 수 있음.

public class ChildProduct<T, M, C> extends Product<T, M> {…}

* 제네릭 인터페이스를 구현할 경우, 그걸 구현한 클래스도 제네릭 타입임.

**Lab: Generic 타입/메소드/클래스**

# Store 클래스 – Generic

|  |
| --- |
| **class** Store <T> {  **private** T data;  **public** **void** set(T data) {  **this**.data=data;  }  **public** T get() {  **return** data;  }  } |

그리고 Store<String> store = new Store<String>(); 식으로 사용한다.

**컬렉션Collection:** 자바에서 자료구조를 구현한 클래스

객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 제공되는 라이브러리

* 리스트(list), 스택(stack), 큐(queue), 집합(set), 해쉬 테이블(hash table) 등.

 java.util 패키지에 포함됨

 컬렉션 클래스와 컬렉션 인터페이스로 제공

컬렉션 인터페이스 : 주요 인터페이스와 메소드

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

List 컬렉션과 주요메소드

구현클래스: ArrayList, Vector, LinkedList

반복자 iterator: 컬렉션의 원소들을 하나씩 처리하는 데 사용

+ 그외 컬렉션들: Set, Stack, Queue, Map