Java 기초

제네릭(Generic)

제네릭(Generic)

- 제네릭(Generic)이란?
  - 기존 일반 오브젝트 타입은 타입을 명시하기 위해 반드시 캐스팅 연산자를 붙여야만 쓸 수 있었다.
  - Java 5 이후 제네릭(Generic)타입이 추가되었는데 제네릭 타입을 이용함으로써 잘못된 타입이 사용될 수 있는 문제를 컴파일 과정에서 제거할 수 있게 되었다.
  - 제네릭은 클래스와 인터페이스, 그리고 메소드를 정의할 때 타입(type)을 파라미터(parameter)로 사용할 수 있도록 한다.
  - 제네릭을 사용하는 코드는 비 제네릭 코드에 비해 다음과 아래와 같은 이점을 제공한다.
    - 컴파일 시 강한 타입 체크를 할 수 있다.
    - 타입 변환(casting)을 제거한다.

제네릭(Generic)

- 제네릭(Generic)이란?

```
List list = new ArrayList();
list.add("hello");
String str = (String)list.get(0); //타입 변환 필요
```

```
List<String> list = new ArrayList();
list.add("hello");
String str = list.get(0); //타입 변환이 필요가 없다.
```

제네릭(Generic)

- 제네릭(Generic)이란?
  - 기본적인 선언 방식은 아래와 같다.

```
public class 클래스명<T>
public interface 인터페이스명<T>
```

  - 멀티 타입으로 선언시에는 <>안에 타입을 더 추가하여 쓸 수 있다.

```
public class 클래스명<K,V…>
public interface 인터페이스명<K,V…>
```

제네릭(Generic)

- 단일 제네릭 클래스 예제

```java
public class Box<T> {
    private T t;
    public T get() { return t; }
    public void set(T t) { this.t = t; }
}
```

```java
public class BoxExample {
    public static void main(String[] args) {
        Box<String> box1 = new Box<String>();
        box1.set("hello");
        String str = box1.get();

        Box<Integer> box2 = new Box<Integer>();
        box2.set(6);
        int value = box2.get();
    }
}
```

제네릭(Generic)

- 다중 제네릭 클래스 예제

```java
public class Product<T, M> {
    private T kind;
    private M model;

    public T getKind() { return this.kind; }
    public M getModel() { return this.model; }

    public void setKind(T kind) { this.kind = kind; }
    public void setModel(M model) { this.model = model; }
}
public class ProductExample {
    public static void main(String[] args) {
        Product<Tv, String> product1 = new Product<Tv, String>();
        product1.setKind(new Tv());
        product1.setModel("스마트Tv");
        Tv tv = product1.getKind();
        String tvModel = product1.getModel();

        Product<Car, String> product2 = new Product<Car, String>();
        product2.setKind(new Car());
        product2.setModel("디젤");
        Car car = product2.getKind();
        String carModel = product2.getModel();
    }
}
```

제네릭 메소드(Generic Method)

- 다중 제네릭 클래스 예제
  - 메소드에서도 파라미터 타입과 리턴 타입에 제네릭 객체를 쓸 수가 있다.
  - 제네릭 타입은 아래와 같이 선언이 가능하다

```
public <타입파라미터…> 리턴타입 메소드명(매개변수…) { }
```

  - 제네릭 메소드는 아래 두 가지 방식으로 호출이 가능하다.

```
리턴타입 변수 = <구체적타입> 메소드명(매개값) //타입 파라미터를 명시적으로 지정
리턴타입 면수 = 메소드명(매개값); //매개값을 보고 구체적 타입을 추정
```

제네릭 메소드(Generic Method)

- 제네릭 메소드 예제 - 1

```java
public class Box<T> {
    private T t;
    public T get() { return t; }
    public void set(T t) { this.t = t; }
}
public class Util {
    public static <T> Box<T> boxing(T t) {
        Box<T> box = new Box<T>();
        box.set(t);
        return box;
    }
}
```

```java
public class BoxingMethodExample {
    public static void main(String[] args) {
        Box<Integer> box1 = Util.<Integer>boxing(100);
        int intValue = box1.get();

        Box<String> box2 = Util.boxing("홍길동");
        String strValue = box2.get();
    }
}
```

제네릭 메소드(Generic Method)

- 제네릭 메소드 예제 - 2

```java
public class Pair<K, V> {
    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public void setKey(K key) { this.key = key; }
    public void setValue(V value) { this.value = value; }
    public K getKey()   { return key; }
    public V getValue() { return value; }
}
public class Util {
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
        boolean keyCompare = p1.getKey().equals(p2.getKey()) ;
        boolean valueCompare = p1.getValue().equals(p2.getValue());
        return keyCompare && valueCompare;
    }
}
```

```java
public class CompareMethodExample {
    public static void main(String[] args) {
        Pair<Integer, String> p1 = new Pair<Integer, String>(1, "사과");
        Pair<Integer, String> p2 = new Pair<Integer, String>(1, "사과");
        boolean result1 = Util.<Integer, String>compare(p1, p2);
        if(result1) {
            System.out.println("논리적으로 동등한 객체입니다.");
        } else {
            System.out.println("논리적으로 동등하지 않는 객체입니다.");
        }

        Pair<String, String> p3 = new Pair<String, String>("user1", "홍길동");
        Pair<String, String> p4 = new Pair<String, String>("user2", "홍길동");
        boolean result2 = Util.compare(p3, p4);
        if(result2) {
            System.out.println("논리적으로 동등한 객체입니다.");
        } else {
            System.out.println("논리적으로 동등하지 않는 객체입니다.");
        }
    }
}
```

제한된 타입 파라미터

- 제한된 타입 파라미터<T extends 최상위 타입>
  - 타입 제한 시 <T extends 최상위타입>을 적어줌으로써 타입을 제한할 수 있다.

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

제한된 타입 파라미터

- 제한된 타입 파라미터<T extends 최상위 타입> 예제

```java
public class Util {
    public static <T extends Number> int compare(T t1, T t2) {
        double v1 = t1.doubleValue(); //Number의 doubleValue 메서드 사용
        //System.out.println(t1.getClass().getName());
        double v2 = t2.doubleValue(); //Number의 doubleValue 메서드 사용
        //System.out.println(t2.getClass().getName());
        return Double.compare(v1, v2);
    }
}
```

```java
public class BoundedTypeParameterExample {
    public static void main(String[] args) {
        //String str = Util.compare("a", "b"); (x)

        int result1 = Util.compare(10, 20);
        System.out.println(result1);

        int result2 = Util.compare(4.5, 3);
        System.out.println(result2);
    }
}
```

와일드 카드<?>,<? extends …>, <? super …>

- 와일드 카드<?>,<? extends …>, <? super …>
  - 제네릭으로 쓸 수 있는 기호는 크게 세가지 정도로 나뉘는데 다음과 같다.
  - T : Type의 약자 , E : Element의 약자 , ? : 와일드카드
  - 여기서 와일드 카드는 모든 값을 포함하는 기호이다.
  - 와일드 카드는 아래와 같이 3가지 형태로 사용할 수 있다.

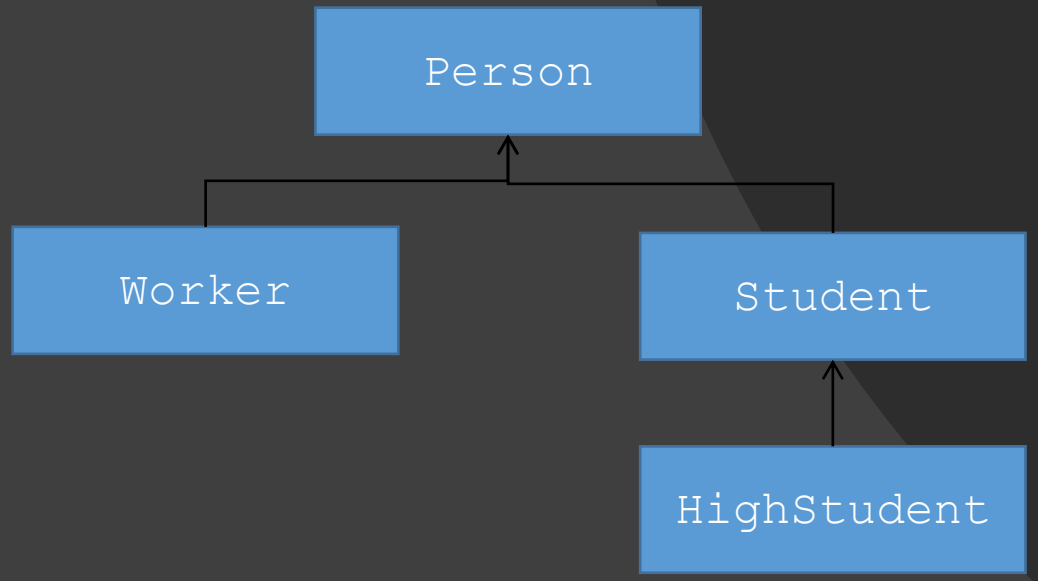
제네릭타입<?> : 모든 클래스나 인터페이스 타입이 올 수 있다.

제네릭타입<? extends 상위타입> : 타입 파라미터를 대체하는 해당 상위타입이나 하위타입만 올 수 있다.

제네릭타입<? super 하위타입> : 타입 파라미터를 대체하는 해당 하위타입이나 상위타입만 올 수 있다.

와일드 카드<?>,<? extends …>, <? super …>

- 와일드 카드<?>,<? extends …>, <? super …> 예제



와일드 카드<?>,<? extends …>, <? super …>

- 와일드 카드<?>,<? extends …>, <? super …> 예제

```java
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() { return name; }
    public String toString() { return name; }
}
```

```java
public class Worker extends Person {
    public Worker(String name) {
        super(name);
    }
}
```

```java
public class Student extends Person {
    public Student(String name) {
        super(name);
    }
}
```

```java
public class HighStudent extends Student {
    public HighStudent(String name) {
        super(name);
    }
}
```

와일드 카드<?>,<? extends …>, <? super …>

- 와일드 카드<?>,<? extends …>, <? super …> 예제

```java
public class Course<T> {
    private String name;
    private T[] students;

    public Course(String name, int capacity) {
        this.name = name;
        students = (T[]) (new Object[capacity]);
    }

    public String getName() { return name; }
    public T[] getStudents() { return students; }
    public void add(T t) {
        for(int i=0; i<students.length; i++) {
            if(students[i] == null) {
                students[i] = t;
                break;
            }
        }
    }
}
```

```java
public class WildCardExample {
    public static void registerCourse(Course<?> course) {
        System.out.println(course.getName() + " 수강생 : " +
                Arrays.toString(course.getStudents()));
    }

    public static void registerCourseStudent(Course<? extends Student> course) {
        System.out.println(course.getName() + " 수강생 : " +
                Arrays.toString(course.getStudents()) );
    }

    public static void registerCourseWorker(Course<? super Worker> course) {
        System.out.println(course.getName() + " 수강생 : " +
                Arrays.toString(course.getStudents()));
    }

    public static void main(String[] args) {
        Course<Person> personCourse = new Course<Person>("일반인과정", 5);
            personCourse.add(new Person("일반인"));
            personCourse.add(new Worker("직장인"));
            personCourse.add(new Student("학생"));
            personCourse.add(new HighStudent("고등학생"));
```

와일드 카드<?>,<? extends …>, <? super …>

- 와일드 카드<?>,<? extends …>, <? super …> 예제

```
일반인과정 수강생: [일반인, 직장인, 학생, 고등학생, null]
직장인과정 수강생: [직장인, null, null, null, null]
학생과정 수강생: [학생, 고등학생, null, null, null]
고등학생과정 수강생: [고등학생, null, null, null, null]

학생과정 수강생: [학생, 고등학생, null, null, null]
고등학생과정 수강생: [고등학생, null, null, null, null]

일반인과정 수강생: [일반인, 직장인, 학생, 고등학생, null]
직장인과정 수강생: [직장인, null, null, null, null]
```

```java
public static void main(String[] args) {
    Course<Person> personCourse = new Course<Person>("일반인과정", 5);
        personCourse.add(new Person("일반인"));
        personCourse.add(new Worker("직장인"));
        personCourse.add(new Student("학생"));
        personCourse.add(new HighStudent("고등학생"));

    Course<Worker> workerCourse = new Course<Worker>("직장인과정", 5);
        workerCourse.add(new Worker("직장인"));

    Course<Student> studentCourse = new Course<Student>("학생과정", 5);
        studentCourse.add(new Student("학생"));
        studentCourse.add(new HighStudent("고등학생"));

    Course<HighStudent> highStudentCourse = new Course<HighStudent>("고등학생과정", 5);
        highStudentCourse.add(new HighStudent("고등학생"));

        registerCourse(personCourse);
        registerCourse(workerCourse);
        registerCourse(studentCourse);
        registerCourse(highStudentCourse);
        System.out.println();

        //registerCourseStudent(personCourse);          (x)
        //registerCourseStudent(workerCourse);          (x)
        registerCourseStudent(studentCourse);
        registerCourseStudent(highStudentCourse);
        System.out.println();

        registerCourseWorker(personCourse);
        registerCourseWorker(workerCourse);
        //registerCourseWorker(studentCourse);          (x)
        //registerCourseWorker(highStudentCourse);      (x)
    }
}
```

제네릭 타입의 상속과 구현

- 제네릭 타입도 다른 타입과 마찬가지로 부모 클래스가 될 수 있다.
- 자식 제네릭 타입은 추가적으로 타입 파라미터를 가질 수 있다.

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

제네릭 타입의 상속과 구현

- 제네릭 타입의 상속과 구현 예제

```java
public class Product<T, M> {
    private T kind;
    private M model;

    public T getKind() { return this.kind; }
    public M getModel() { return this.model; }

    public void setKind(T kind) { this.kind = kind; }
    public void setModel(M model) { this.model = model; }
}

class Tv {}
public class ChildProduct<T, M, C> extends Product<T, M> {
       private C company;
       public C getCompany() { return this.company; }
       public void setCompany(C company) { this.company = company; }
}
```

```java
public interface Storage<T> {
    public void add(T item, int index);
    public T get(int index);
}
```

```java
public class StorageImpl<T> implements Storage<T> {
    private T[] array;

    public StorageImpl(int capacity) {
        this.array = (T[]) (new Object[capacity]);
    }

    @Override
    public void add(T item, int index) {
        array[index] = item;
    }

    @Override
    public T get(int index) {
        return array[index];
    }
}
```

제네릭 타입의 상속과 구현

- 제네릭 타입의 상속과 구현 예제

```
public class ChildProductAndStorageExample {
    public static void main(String[] args) {
        ChildProduct<Tv, String, String> product = new ChildProduct<>();
        product.setKind(new Tv());
        product.setModel("SmartTV");
        product.setCompany("Samsung");

        Storage<Tv> storage = new StorageImpl<Tv>(100);
        storage.add(new Tv(), 0);
        Tv tv = storage.get(0);
    }
}
```