제네릭

- 타입 매개변수(Type parameter)로 다양한 타입에 대한 유여한 대처를 가능하게 함
- 컬렉션의 List, Set, Map 모두 타입을 선언하는 부분에 <>를 사용하며 해당 부분에 타입 매개변수를 지정
  ⇒ 매개변수화 타입(Parameterized type)을 정의한다고 표현

```
// 실제 List 클래스 구현 부분
abstract class List<E> implements EfficientLengthIterable<E> {
...
void add(E value);
...
}
```

- 필요에 따라 매개변수화 타입을 제한 ⇒ 키워드 `extends` 를 사용

```
class Person {
    eat() {
        print("Person eat");
    }
}

class Student extends Person {
    eat() {
        print("Student eat");
    }
}

// 매개변수화 타입을 클래스 Person과 해당 클래스를 상속한 자식 클래스로 제한
class Manager<T extends Person> {
    eat() {
        print("Manager eat");
    }
}

class Dog {
    eat() {
        print("Dog eat");
    }
}

void main() {
    var manager1 = Manager<Person>();
    manager1.eat();

    var manager2 = Manager<Student>();
    manager2.eat();

        // 제네릭 타입을 명시하지 않는 경우 Person 객체를 포함하며
        // extends를 사용해서 매개변수화 타입을 제한하지 않는 경우 dynamic 객체를 포함
    var manager3 = Manager();
    manager3.eat();

    //var manager4 = Manager<Dog>(); // 에러 발생
}
```

- 제네릭 메서드
  - 필요에 따라 클래스가 아닌 메서드에도 사용 가능
  - 메서드의 리턴타입 혹은 매개변수를 타입 매개변수로 지정

```
class Person {
  T getName<T>(T param) {
    return param;
  }
}

void main() {
  Person person = new Person();
  var result = person.getName<String>('Kim');
  print(result);
}
```