Generics

- Purpose
- Generic class
- Generic method
- Names
- Wildcards
- Bounded types
- PECS
- Generics after compilation

Generics

- add stability to your code by making more of your bugs detectable at compile time,
- enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods.

Generic Types

```java
public class GenericExample<T> {
    private T value;

    public GenericExample(T value){
        this.value = value;
    }

    public T getT(){
        return value;
    }

    public static void main(String[] args) {
        GenericExample<Integer> intObject = new GenericExample<>(1);
        Integer valueInteger = intObject.getT();

        GenericExample<String> stringObject = new GenericExample<>("word");
        String valueString = stringObject.getT();
    }
}
```

Generic Methods

```
public class GenericExample {

    public static <T> T getTheFirst(List<T> list){
        return list.get(0);
    }

    public static void main(String[] args) {
        List<Integer> listOfInts = new ArrayList<>();

        listOfInts.add(0);
        Integer intValue = getTheFirst(listOfInts);

        List<String> listOfStrings = new ArrayList<>();

        listOfStrings.add("Java is the best!");
        String stringValue = getTheFirst(listOfStrings);
    }
}
```

Names of type parameters

- E: Element
- K: Key
- N: Number
- T: Type (generic)
- V: Value
- S, U, V, and so on: Second, third, and fourth types in a multiparameter situation

```
public class GenericExample<T, U, V> {
     private T valueT;
     private U valueU;
     private V valueV;

…
}
```

Wildcards

Wildcard (<?>) is specifies an unknown type using generic code.

```
void checkList(List<?> myList){
    // myList can contain anything
    // myList can be accessed
    // but you can't add to the list
}
```

Bounded Types

Integer is a Number

List is a Collection

But

List<Integer> is not a List<Number>

Bound types

<? extends UpperBoundType> //upper bound

<? super LowerBoundType> //lower bound

List<Integer> is a List<? extends Number>

PECS

Producer Extends Consumer Super

Suppose you have a method that takes as its parameter a collection of things, but you want it to be more flexible than just accepting a Collection<Thing>.

Case 1: You want to go through the collection and do things with each item.

Then the list is a producer, so you should use a Collection<? extends Thing>.

Case 2: You want to add things to the collection.

Then the list is a consumer, so you should use a Collection<? super Thing>.

Generics after compilation

Not in Java