Java Generic

Rawlabs Academy

What is Java Generics?

Java **generic methods** and **generic classes** enable programmer to specify, with a **single method declaration**, a set of **related methods**, or with a **single class declaration**, a set of **related types**, respectively.

> *Get your data structures correct first and the rest of the program will write itself.* **David Jones**

Java Generic Methods

A **single generic method** declaration that can be called with arguments of **different types**. Based on the **types of the arguments** passed to the generic method, the compiler handles each method call appropriately.

Java Generic Methods - Rules

- All generic method declarations **have a type parameter** section delimited by angle brackets **(< >)** that precedes the method's return type.
- Each type parameter section contains **one or more** type parameters separated by **commas**. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
- The type parameters can be used to **declare the return type** and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
- A generic method's body is declared like that of any other method. **Note** that type parameters can represent **only reference types**, not primitive types (like int, double and char).
- The convention is to use a **1-letter name** such as: **T** for Type, **E** for Element, **N** for Number, **K** for Key, or **V** for Value

Java Generic Methods - Example

Print the different type, the parameter **T** mean type and should define with `<T>` before method type.

```
public class GenericMethod {
    public static <T> void print(T value) {
        System.out.println("Hello : " + value)
    }
    public static void main(String[] args) {
        print(1);
        print("dua");
        print(3.0);
        print('a');
    }
}
```

Cont..

```
public class GenericMethod {
    public static <T> T max(T a, T b, T c) {
        T max = a;
        if (Double.parseDouble(String.valueOf(b)) >
             Double.parseDouble(String.valueOf(max))) {
            max = b;
        }
        if (Double.parseDouble(String.valueOf(c)) >
             Double.parseDouble(String.valueOf(max))) {
            max = c;
        }
        return max;
    }
    public static void main(String[] args) {
        System.out.println(max(1, 2, 3));
        System.out.println(max(2.5, 9.4, 8));
    }
}
```

Generic Method - Bounded Type Parameters

There may be times when you'll want to **restrict the kinds of types** that are allowed to be passed to a type parameter. For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what bounded type parameters are for

```
public class GenericMethod {
    public static <T extends Comparable<T>> T maximum(T x, T y, T z) {
        T max = x;
        if(y.compareTo(max) > 0) {
            max = y;
        }
        if(z.compareTo(max) > 0) {
            max = z;
        }
        return max;
    }
    public static void main(String[] args) {
        System.out.println(max(1, 2, 3));
        System.out.println(max(2.5, 9.4, 8.0));
    }
}
```

Java Generic Classes

A generic class declaration looks like a non-generic class declaration, except that the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas. These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

Generic Classes - Example

```
public class Box<T> {
    private T data;

    // Setter and getter method
}
```

```
public static void main(String[] args) {
    Box<String> stringBox = new Box<>();
    stringBox.setData("Any box");

    Box<Integer> integerBox = new Box<>();
    integerBox.setData(1);

    Box<Double> doubleBox = new Box<>();
    doubleBox.setData(3.0);

    System.out.println(stringBox.getData());
    System.out.println(integerBox.getData());
    System.out.println(doubleBox.getData());
}
```

Exercise

1. Create method to print an array of different type using single generic method.
2. Create method to merge 2 arrays of different type using single generic method.
3. Create generic class to store 2 data with different type

Task - Array Unique

Create a method to identify the unique value between 2 array of different type using single **generic method**.

Test Case :

- Input : `[1, 2, 3, 4]` and `[1, 3, 5, 10, 16]`
  Output : `[2, 4, 5, 10, 16]`
- Input : `["one", "two"]` and `["two", "six"]`
  Output : `["one", "six"]`

Task - Generic Class

Create class like the output json below using **generic class**. The `data` is generic type that can be used for `String`, `Integer`, `Object`, `List`, etc.

```
{
    "responseCode": "SUCCESS",
    "responseDesc": "Success",
    "timestamp: ": "2023-02-10T18:19:26.492",
    "data": {}
}
```

```
{
    "responseCode": "SUCCESS",
    "responseDesc": "Success",
    "timestamp: ": "2023-02-10T18:19:26.492",
    "data": []
}
```