Java SE & Spring

Module 1: Java SE

09.Generics in Java

What is Java Generics?

The Java Generics allows us to create a single class, interface, and method that can be used with different types of data (objects). This helps us to reuse our code. As like **Collection** framework, **Generics** does not work with primitive types too.

Generic Classes

We can create a class that can be used with any type of data. Such a class is known as **Generics Class**.

Example:

```java
public class GenericClass<T>{
    private T genericObject;

    public GenericClass(T genericObject){
        this.genericObject = genericObject;
    }

    public T getGenericObject(){
        return this.genericObject;
    }
}

// in main class:
GenericClass<Integer> intGeneric =  new GenericClass<>(5);
GenericClass<String> stringGeneric =  new GenericClass<>("Java is awesome!");
System.out.println(intGeneric.getGenericObject());
System.out.println(stringGeneric.getGenericObject());
```

Generic Classes

In the example, Here, **T** used inside the angle bracket **<>** indicates the type parameter. Inside the Main class, we have created two objects of **GenericsClass intGeneric** - Here, the type parameter T is replaced by Integer. Now, the GenericsClass works with integer data. stringGeneric - Here, the type parameter T is replaced by String. Now, the GenericsClass works with string data. While creating objects for GenericClass, Integer and String are called type argument.

Further reading: https://docs.oracle.com/javase/tutorial/java/generics/types.html

Type Parameter Naming

By convention, type parameter names are single, uppercase letters. This stands in sharp contrast to the variable naming conventions that you already know about, and with good reason: Without this convention, it would be difficult to tell the difference between a type variable and an ordinary class or interface name.

The most commonly used type parameter names are:

- E - Element (used extensively by the Java Collections Framework)
- K - Key
- N - Number
- T - Type
- V - Value
- S, U, V etc. - 2nd, 3rd, 4th types

Further reading: https://docs.oracle.com/javase/tutorial/java/generics/types.html

Generic Methods

Similar to the generics class, we can also create a method that can be used with any type of data. Such a methods are known as Generics Method.

Example:

```java
public class ClassWithGenericMethod {
    public <T> void displaySomething(T genericTypeObject){
        System.out.println("This is a generic method, and has type: "
                + genericTypeObject.getClass()
                + " with value: "
                + genericTypeObject);
    }
}
//-------------------------------------------------------------------------
public class Main {
    public static void main(String[] args) {

        ClassWithGenericMethod classWithGenericMethod =  new
ClassWithGenericMethod();
        classWithGenericMethod.<Integer>displaySomething(3);
        classWithGenericMethod.<String>displaySomething("We love Java!");
    }
}
```

Generic Methods

In the example, the type parameter <T> is inserted after the modifier public and before the return type void.

We can call the generics method by placing the actual type <String> and <Integer> inside the bracket before the method name. We can call the generics method without including the type parameter. In this case, the java compiler can match the type parameter based on the value passed to the method.

Bounded Types

In general, the type parameter can accept any data types (except primitive types). However, if we want to use generics for some specific types (such as accept data of number types) only, then we can use bounded types. In the case of bound types, we use the **extends** keyword.

Example:

```java
class GenericsClass <T  extends Number> {

  public void display() {
    System.out.println("This is a bounded type generics class.");
  }
}

class Main {
  public static void main(String[] args) {

    // create an object of GenericsClass
    GenericsClass<Integer> obj =  new GenericsClass<>();
  }
}
```

Questions ?

Next:Enums & Widely Used Java APIs