Java Generics

Owner	sajeed mohammed
Verification	
Tags	
Last edited time	@July 10, 2023 11:31 PM
Created time	@July 10, 2023 9:17 PM

Java Generics

Java Generics is a feature of the Java programming language that allows developers to create classes, interfaces, and methods that operate on specified types.

To avoid these runtime errors, we can use Generics in Java. When using Generics, we need to specify the type of object that the collection will contain. In the example code below, a List object is created with type Integer, so it can only store integers. If we try to add a string to this list, it will result in a compile-time error.

```
List<Integer> list2 = new ArrayList<Integer>();
//list2.add("NameOne"); //Compile Time Error
list2.add(10);
System.out.println(list2); // [10]
```

Using Generics makes the Java code more **type-safe and easier to read and understand.** It also helps in detecting errors at compile-time rather than runtime.

TypeSafety

```
import java.util.ArrayList;
import java.util.List;

//Type Safety
public class Example01 {

  public static void main(String[] args) {

    //Without Generic type we can store any type of the Object
    List list1 = new ArrayList();
    list1.add(10);
    list1.add("NameOne");
    list1.add('c');
    list1.add(10.00);
    list1.add(5.0f);
    System.out.println(list1); // [10, NameOne, c, 10.0, 5.0]

    //We can hold only single type of objects in generic. It doesnot allow to store other objects
    //With GenericType, It is required to specify the type of the Object.
    List<Integer> list2 = new ArrayList<Integer>();
    //list2.add("NameOne"); //Compile Time Error
    list2.add(10);
    System.out.println(list2); // [10]

  }
}
```

Type Casting

TypeCasting is required, Need to type cast the object

```
import java.util.ArrayList;
import java.util.List;

//TypeCasting
public class Example02 {

  public static void main(String[] args) {

    // TypeCasting is required, Need to type cast the object
    List list1 = new ArrayList();
    list1.add("10");
    String s1 = (String) list1.get(0); // Type Casting
    System.out.println(s1); // 10

    list1.add(10);
    Integer s3 = (Integer) list1.get(1); // Type Casting
    System.out.println(s3); // 10

    // Type casting is not required, No need to type cast the object
    List<String> list2 = new ArrayList<String>();
    list2.add("Sai Kiran");
    String s2 = list2.get(0);
    System.out.println(s2); // Sai Kiran
  }
}
```

Type Casting

We can create generic method that can accept any type of arguments, the scope of the arguments is limited to the method

```
import java.util.List;

//Generic Method
//We can create generic method that can accept any type of arguments, the scope of the arguments is limited to the method
public class Test {

  // E : Element
  public static <E> void printArray(E[] elements) {

    for (E e : elements) {
      System.out.println(e);
    }
  }

  public static <E> void printList(List<E> list) {

    for (E e : list) {
      System.out.println(e);
    }
  }
}
-------------------------------------------------------------------------------
import java.util.ArrayList;
import java.util.List;

public class Client extends Test {

  public static <E> void main(String[] args) {
```

```
    //Arrays
    Integer[] intArrays = { 10, 20, 30, 40, 50 };
    printArray(intArrays);

    String[] stringArrays = {"NameOne", "NameTwo", "NameThre", "NameFour"};
    printArray(stringArrays);

    //List
    List<Integer> list = new ArrayList<>();
    list.add(1);
    list.add(2);
    list.add(3);
    list.add(4);
    printList(list);

  }
}

Console
10
20
30
40
50
NameOne
NameTwo
NameThre
NameFour
1
2
3
4
```

Generic Class

The code block provided is an example of a generic class. The class is called `GC` and has a type parameter `T`. The field `obj` is of type `T`, and the `add` method takes a parameter of type `T` and assigns it to the `obj` field. The `get` method returns the value of the `obj` field.
By using a generic type parameter `T`, the class can be used with different types without having to create a separate class for each type. This makes the code more flexible, reusable, and easier to maintain.

```
//Generic Class
public class GC<T> {

  //Fields
  private T obj;

  //add Method
  public void add(T obj) {
    this.obj = obj;
  }

  //get Method
  public T get() {
    return obj;
  }
}
--------------------------------------------------------------------------------
public class Client {

  public static void main(String[] args) {

    GC<Integer>  gc = new GC<Integer>();
    gc.add(2);
```

```
    //gc.add("Sai Kiran"); //The method add(Integer) in the type GC<Integer> is not applicable for the arguments (String)
    System.out.println(gc.get()); // 2

    GC<String>  gc1 = new GC<String>();
    //gc1.add(2); //The method add(String) in the type GC<String> is not applicable for the arguments (int)
    gc1.add("Sai Kiran");
    System.out.println(gc1.get()); // Sai Kiran
  }
}
```