GENERICS IN JAVA

1.Εισαγωγη

Τα Generics προστέθηκαν στην Java 5 για να παρέχουν τον έλεγχο τύπου compile-time και την απομάκρυνση του **κινδύνου ClassCastException** που ήταν συνηθισμένος κατά την **εργασία με collection classes**. Το σύνολο του πλαισίου συλλογής ανατυπώθηκε για να χρησιμοποιήσει τα **Generics** για type-safety. Ας δούμε πώς τα **Generics** μας βοηθούν να χρησιμοποιούμε κώδικα με ασφάλεια.

```
List list = new ArrayList();
list.add("abc");
list.add(new Integer(5)); //OK

for(Object obj : list){
      //type casting leading to ClassCastException
at runtime
    String str=(String) obj;
}
```

Ο παραπάνω κώδικας μεταγλωττίζει χωρίς πρόβλημα, **αλλά ρίχνει ClassCastException κατά το χρόνο εκτέλεσης** επειδή προσπαθούμε να μεταδώσουμε Object στη λίστα των String ενώ ένα από τα στοιχεία είναι τύπου Integer.

Το παρακάτω παράδειγμα λύνει μερικώς το πρόβλημα του προηγουμένου παραδείγματος

```
List<String> list1 = new ArrayList<String>();
list1.add("abc");
//list1.add(new Integer(5)); //compiler error

for(String str : list1){
     //no type casting needed, avoids
ClassCastException
}
```

Επιπλέον στο παρακάτω παράδειγμα:

```
List list = new LinkedList();
list.add(new Integer(1));
Integer i = list.iterator().next();
```

Ο μεταγλωττιστής θα παραπονεθεί για την τελευταία γραμμή. Δεν γνωρίζει **ποιος τύπος δεδομένων επιστρέφεται**. Ο μεταγλωττιστής **θα απαιτήσει explicit casting:**

```
Integer i = (Integer) list.iterator.next();
```

Δεν υπάρχει σύμβαση που θα μπορούσε να εγγυηθεί ότι ο τύπος επιστροφής της λίστας είναι ακέραιος. Η καθορισμένη λίστα θα μπορούσε να κρατήσει οποιοδήποτε αντικείμενο. Ξέρουμε μόνο ότι ανακτάμε μια λίστα. Κατά την εξέταση των τύπων, μπορεί μόνο να εγγυηθεί ότι είναι ένα αντικείμενο, συνεπώς απαιτεί ένα ρητό cast για να διασφαλιστεί ότι ο τύπος είναι ασφαλής.

Το cast μπερδεύει επίσης τον κώδικα μας. Μπορεί να προκαλέσει σφάλματα χρόνου εκτέλεσης που σχετίζονται με το είδος, αν ένας προγραμματιστής κάνει λάθος με το **explicit casting**.

2.Generic – κλάσεις

Μια generic κλάση ή παραμετρική κλάση ορίζεται όπως και οι κανονικές κλάσης με την διαφορά ότι στο όνομα της πρέπει να ακολουθεί τουλάχιστον μια τυπική παράμετρος αναμεσά στα σύμβολα <>.

Μπορεί να υπάρχουν περισσότερες τυπικοί παράμετροι διαχωρισμένοι με το κομμα (,).

```
class GenericTest <T>{

private T t;

public void add(T t){ this.t=t;}
public T get(){return t;}

public static void main(String args[]){

GenericTest<Integer> i1 = new GenericTest<Integer>();
GenericTest<String> s1= new GenericTest<String>();

s1.add(new String ("I WANT TO PASS THE LESSON WITH:"));
i1.add(new Integer(10));

System.out.println(s1.get());
System.out.println(i1.get());}}
```

3.Generic Methods

Οι **Generic methods** είναι εκείνες οι μέθοδοι που είναι γραμμένες με μια **απλή δήλωση μεθόδου** και μπορούν να **καλούνται με ορίσματα διαφορετικών τύπων**. Ο μεταγλωττιστής θα διασφαλίσει την ορθότητα του όποιου τύπου χρησιμοποιείται. Αυτές είναι μερικές ιδιότητες γενικών μεθόδων:

1. Οι **Generic methods** έχουν παράμετρο (**diamond operator <> που περικλείει τον τύπο) πριν από τον τύπο επιστροφής** της δήλωσης μεθόδου.
2. Οι παράμετροι τύπου **μπορούν να οριοθετηθούν.**
3. Οι **Generic methods** μπορούν να έχουν διαφορετικές παραμέτρους τύπου που διαχωρίζονται με κόμματα.
4. Το σώμα της μεθόδου για μια **Generic method** είναι ακριβώς όπως μια κανονική μέθοδος

Παράδειγμα 1

```
package generic_class;

public class generic_method {

public static < A > void printArray( A[] inputArray )
//   ypografi tis methodou
     {

     for ( A element : inputArray )
     {
     System.out.print(element + "  ");
     }

     System.out.println();
     }

     public static void main( String args[] )
     {
     Character[] charArray = { 'J', 'A', 'V', 'A'};
     Double[] doubleArray = { 2.1, 5.3, 1.2, 8.4 };
     Integer[] intArray = { 4, 32, 45, 67, 89 };
     System.out.println( "Pinakas haractirvn : " );
     printArray( charArray );
     System.out.println();
     System.out.println( "Pinakas dekadikvn : " );
     printArray( doubleArray );
     System.out.println();
     System.out.println( "Pinakas akeraivn : " );
     printArray( intArray  );
     }
}
```

Παράδειγμα 2

```
public class GenericsMethods {
        //Java Generic Method
public static <T> boolean isEqual(GenericsType<T> g1, GenericsType<T> g2){
                return g1.get().equals(g2.get());
        }
        public static void main(String args[]){
                GenericsType<String> g1 = new GenericsType<>();
                g1.set("MERRY CHRISTMAS EVERYBODY");

                GenericsType<String> g2 = new GenericsType<>();
                g2.set("MERRY CHRISTMAS EVERYBODY");

        boolean isEqual = GenericsMethods.<String>isEqual(g1, g2);
                //above statement can be written simply as
                isEqual = GenericsMethods.isEqual(g1, g2);

//This feature, known as type inference, allows you to invoke a generic method as an ordinary method, without specifying a type between angle brackets.
                //Compiler will infer the type that is needed
        }
}
```

4. Bounded Types

Πολλές φορές υπάρχουν περιπτώσεις όπου πρέπει να καθορίσουμε έναν γενικό τύπο, αλλά θέλουμε να ελέγξουμε ποιους τύπους μπορούμε να καθορίσουμε. Οι **Bounded Types** μπορούν να χρησιμοποιηθούν για να περιορίσουν τα όρια του γενικού τύπου χρησιμοποιώντας ένα άνω ή κάτω όριο αντίστοιχα. Για παράδειγμα, εάν θέλουμε να περιορίσουμε έναν τύπο σε συγκεκριμένο τύπο ή σε έναν υποτύπο αυτού του συγκεκριμένου τύπου, χρησιμοποιούμε την ακόλουθη συμβολοσειρά:

<T extends UpperBoundType>

Η αντίστοιχά για το κάτω όριο έχουμε:

<T super LowerBoundType>

Το LowerBoundType σε πρόσφατες εκδόσεις είναι κατηργημένο.

Τι θα συμβεί στο παρακάτω παράδειγμα και γιατί;

```
public class GenericNumberContainer <T extends Number> {
    private T obj;

    public GenericNumberContainer(){
    }

    public GenericNumberContainer(T t){
        obj = t;
    }
    /**
     * @return
 the obj
     */
    public T getObj() {
        return obj;
    }

    /**
     * @param obj the obj to set
     */
    public void setObj(T t) {
        obj = t;
    }
}
```

public static void main(String args[]){

```
GenericNumberContainer<Integer> gn = new
GenericNumberContainer<Integer>();

gn.setObj(3);

GenericNumberContainer<String> gn2 = new
GenericNumberContainer<String>();}}
```

Είναι πιθανό να μην γνωρίζουμε τον τύπο ενός **argument** που μεταβιβάζεται σε μια μέθοδο. Τα Generics μπορούν να εφαρμοστούν σε επίπεδο μεθόδου για την επίλυση τέτοιων καταστάσεων.

Έστω η περίπτωση όπου επιθυμούμε να αναπτύξουμε μια κλάση αριθμομηχανής που δέχεται τύπους αριθμών. Τα Generics θα μπορούσαν

να χρησιμοποιηθούν για να εξασφαλίσουν ότι οποιοσδήποτε τύπος αριθμού θα μπορούσε να περάσει ως **argument** στις μεθόδους υπολογισμού αυτής της κλάσης. Για παράδειγμα, η μέθοδος add() καταδεικνύει τη χρήση Generics για περιορισμό των τύπων.

```
public static <N extends Number> double add(N a, N b){
     double sum = 0;
     sum = a.doubleValue() + b.doubleValue();
     return sum;
}

double genericValue1 = Calculator.add(3, 3f);
```

ΠΗΓΕΣ

https://www.oracle.com/technetwork/articles/java/juneau-generics-2255374.html

https://www.baeldung.com/java-generics

https://www.journaldev.com/1663/java-generics-example-method-class-interface#generics-java