Java Programming

12/08/24

CSA - 0985

Assignment - 3

G. Sugirbalaji

192321111

12/8/2024

Generic class in Java :-

Java Generics was introduced to deal with type-safe objects. it makes the code stable. Java Generics methods and classes, Enables programmer with a single method declaration, a set of related methods, a set of related types. Generics also provide compile-time type safety which allows programmers to catch invalid types at compile time.

Generic means parameterized types. A Generic class simply means that the items or functions in that class can be generalized with the Parameter (example T) to specify that we can add any type as a Parameter in place of T like integer, char, string, double or any other user-defined type.

Eg: single type parameter.

1) class Solution<T>
{
T data;
Public static T getdata(){
return data;
}
}

2) class A<T>
{
Private T data;
A(T data){
this.data = data;
}
Public T getdata()
{
return this.data;
}
}

Multiple type parameters :-

```
Public class Pair <K, V> {
    Private K key;
    Private V value;

    Public Pair (K key, V value) {
    this . key = key;
    this . value = value;
    }

    Public K getkey () { return key; }
    Public V getvalue () { return values; }
}
```

Generic method in Java :-

Generic methods are methods that introduce their own type parameters. This is similar to declaring a generic type. but the type parameter's scope is limited to the method where it is declared static

and non-static generic methods are allowed. as well as generic class constructor.

Eg:-

```
class A{
    Public <T> void gmethod (Tdata){
        System.out.Println("Generics method:");
        System.out.Print("Data Passed : "+data);
    }
}
Public class main{
    Public static void main(string[] args){
        A obj = new A();
        obj <string> g method("Java Programing);
        Obj < Integer > g method (25)
    }
}
```

Output:-

Generics method:

Data Passed : Javaprogramming

Generics method:

Data Passed : 25

Hasnext :-

hasnext() method to check if we have any inputs left in the console to be read. If it returns true, we read the next input; otherwise we break from the loop.

Eg:-

```
class main{
    Public static void main (String() args[]){
    vector < String > fruits = new vector
    fruits . add ("apple");
    fruits . add ("orange");
    System . out . Println (fruits).
```

```
String . out . Println (elements)

Vector <String> Indian fruits = new Vector();
indianfruits .add ("Pone");
indianfruite . add all (fruits);
System . out . Println (fruits);
Herator <Strings> "terate = indian fruits
System . out . fruintln ("vectors:");
    while (iterate . hesNext ()) {
        System . out . Println (iterate next());
        System . out . Print (" ");
    }
}
```

Stack :-

A Stack is a generic data structure that represents a LIFO (last in first out) collection of objects allowing for pushing/popping elements in constant time. For the new implementations, we should favor the Deque interface and its implementations.

Eg :-

```
import java.util.*; Stack;
class main{
    Public Static void main(String [3args){
    Stack <String> fruits = new Stack<>
                                ()
    fruits. Push (iten = "APPle");
    fruits Pust (iten : "orange");
    fruits Pust ( iten : "mango");
    System. out. Println (" Stack :" + fruits)
    String ramove = fruits. POP()
    System. out. Println ("stack :" +
                                remove);
```

```
fruits Pust ("Pinapple");
  fruits . Peek();
  System . out . Println (Display);
int Pos = fruits . Search ("orange");
  boolean = fruits . empty ();
  System . out . Println (e1);
```

Deque in Java;

A linear collection that supports element insertion and removal at both ends. The name deque is short for double ended queue" and is usually pronounced "deck".

Eg:-

```
import java long;
import Java util . linked list;
import . Java util . A rray deque;
class main.
{
  Public Static void main (String[] args)
{ Array Deque <Strin> fruits = new.
```

```
fruits.add("Apple");
fruits.add("Banana");
fruits.addFirst("oranges");
fruits.addLast("mangs");
System.out.println("Queue : " + fruits);
String= fruits.remove();
System.out.println("Dequeues: " + r);
System.out.println("Queue : " + fruits);
String display = fruits.peek();
System.out.println("Queue : " +
boolean e= fruits.isEmpt(); display);
System.out.println("is the Queue is
                    Empty: " + a);
System.out.println("Empty Dequeue " +
boolean e= fruits.isEmpty(); fruits)
system.out.println("is the Queue is
        Empty: " + e());
    }
}
```

Priority Queue :

```
import java.util.PriorityQueue;
import java.util.Queue;
Public class fruit Priorityqueue
{
    Public static void main (String[], args)
{
    Queue<String> fruitpriority que new
                        Priority queue<>();
fruit Priority Queue.add("Blueberry")
fruit Priority queue.add("Strausberry)
{ System.out.Println(fruitpriority queue.poll
}
}
```