CSA0985
JavA.

ASSINGMENT-3

3

U. Jayakrishna
192324060

18/08/24

Generic class: A Generic class means items or functions in that class can be generalized with parameter to specify we can add parameter in place of T like int, string, char, etc

```
class A < T > {
   private T data;
   A (T data) {
   this.data=data;
  }
  public T get data ( ) {
   return this.data;
  }
}
class main {
   public static void main (String [] args) {
      A <int> obj1 = new A <> (5);
   system.out.println ("Generic class returns :" + obj1.getdata());
   A <string> obj2 = new A<> ("Java programming");
   system.out.println ("Generic class returns :"+ obj2.getdata());
  }
}
```

Generic Method:

Generic Methods that introduces their own type parameters. this similar to a generic type. it includes list of parameters, inside angle brackets, which appears before method's return type.

```
class A {
  public <T> void g Method (T data){
   system.out.println ("Generic method");
   system.out.println ("data passed:"+data);

}}
class main {
   public static void main (String[] args) {
       A obj = new A();
   obj.<String> g Method ("Java programming");
    obj.<integer> g Method (25);

}}
```

2) Array List in java. : It is a part of Java collection Framewor it provides dynamic arrays in java. It is used if we declare an array then we need to mention size. but in array list not needed to mention size.

```
   Import java.util.List;
    import java.util.Arraylist;

class main {
     public Static void main (String[] args) {
      List<Integer> number = new array list <> ();
      numbers.add (1);
      numbers.add (2);
      numbers.add (3);
syslem.out . printhn ("List:"+number);
int get number = numbers. get(2);
 syslem.out. printIn ("Acessed elements: "+ get number);
 int remove Number = numbers. remove(1).
syslem.out. printIn ("Remove Element:"+ remove Number);
}
}
   output :
       List : [1,2,3]
   acessed elements: 3
   removed elements: 2
```

Linked List : It is a linear data structure where elements are not stored in contiguous locations and every element is a specifi object with data part and address part.

```
    import java.util. List;
    import java.util.Linked List;
 class Main {
      Public static void main (String[] args){
      List <String> numbers = new Linked List <> () ;
      numbers. add ("Apple");
      numbers. add ("orange");
      numbers. add ("Mango");
     syslem.out. printIn ("List:" + numbers);
     String number = numbers. get(2);
     syslem.out. printIn ("Accessed Elements:" + number);
```

```
int index = numbers."index of ("Apple");
system.out.println ("position of 2 is:"+ index);
numbers.set (2, "Banana");
system.out.println ("updated list "+ numbers);
 }
}
```

output:

```
    Acessed element : [Apple, orange, mango]
       position : 2
       updated list : [Apple, orange, Banana]
```

Vector: vector class implements a growable array of objects. it implements a dynamic array. It contains components that can be acessed using integer index

```
      import java.util. iterator;
      import java.util.vector;
   class main {
        Public static void main (string[] args) {
            vector <string > fruits = new vector <> ();
             fruits.add ("Apple");
             fruits.add ("orange");
             fruits.add ("Mango");
        system.out.println ("vector:"+ fruits);
            string element = fruits.get(2);
         system.out.println ("Element of index 2:"+ element);
```

```
  fruits.add (3, "Banana");
system.out.println ("vector:"+ fruits);
vector<String> indian fruits = new vector<>();
   indian fruits.add ("promagranate");
   indian fruits.add All (fruits);
 system.out.println ("New vector:"+ Indian fruits);
 vector <String> iterale = indian fruits.iterator";
    system.out.println ("vector");
 while (iterale.hasNext(?)){
    System.out.println next();
  }
}
```

stack : The stack follows Last in First out. The stack class extends vector and provides additional functionality specifically like push, Pop, peek etc. Stack class refered as subclass of vector,

```
import java.util.stack;
class Main {
  public static void main (String[] args){
    stack <String> fruits = new stack <> ();
      fruits.push ("Apple");
      fruits.push ("orange");
      fruits.push ("Mango");
   system.out.println ("stack:"+ fruits);
```

```
    string element = fruits pop ();
  system. out. println ("popped element is : " + element);
    string element 1 = fruits = peek ();
  system. out. println ("Last added Element is : " + element1);

}}
output:    stack : (Apple, orange, Mango)
           Popped element = Apple
           Last-added element : Mango
```

Queue: It abstract data type or linear data structure from elements can be inserted at rear of queue and elements can be deleted from front of queue.

```
      import. java. lang ;
      import java.util . Linked list;
      import java-util . Queue;
class Main {
   public static void main (String [] args) {
  Queue < String > fruits = new linked list <> ();
   fruits . add ("Apple");
   fruits.add ("orange");
   fruits . add ("Mango");
  system. out. println ("Queue" + fruits);
     String r = fruits . peek ();
    system. out. println ("Stack:" + display);
boolean e =fruits is empty();
system-out. println ("Emply Queue:"+e);

fruits clear ();
   boolean e1 = fruits . is emply ();
   syslem-out . println ("is queue is emply :"+e);

 }
}
```

output:

```
     Queue : [Apple, orange , Mango]
      queue : Apple.
      Queue = [orange, Mango]
      stack : orange
     Smpty Queue : False.
```

Deqveue: The double ended queue is an abstract data type that generalize a queue from which elements can be inserted or deleted either from both front or real ends.

```
       Import java . lang;
       import java . util. Linkedlist;
       import java . util. Array deqveue.

  class Main {
       public static void main (String[] args) {
    Array deqveue <String> fruits= new array Deqveue ();
           fruits.add ("Apple");
           fruits . add ("Banana");
           fruits . add ("orange").
           fruits , add ("Mango");
     System. out - println ("deqveue :" + fruits);
       string r = fruits - remove ();
    system.out.println ("Deques"+r);
    system.out.println ("Dequeue :"+ fruits];
  string display = fruits.peek ();
  system.out.println ("Dequeue :"+display);
   boolean e = fruits.is empty ();
   system.out.println ("is dequeue is empty :"+e);
  fruits.clear ();
   system.out.println ("empty dequeue :"+fruits)
    boolean () = fruits.is empty ();
   System.out.println ("is dequeue is empty :"+e);
 }
}
```

Hash map: Hash map is similar to hash table, but it is unsynchronized. it allows to store null keys as well..

```
         import java.util.Map ();
         import java, util.HashMap ();
  class Main{
    public static void main (string[] args){
    Map <integer> string fruits = new Hash Map <> ();
```

```
fruits . put (1, "apple");
fruits - put (2, "orange");
fruits . put (3, "mango");
system - out . println ("Map : " + fruits);
system.out.println ("keys :" + fruits.keyset ());
system.out - println ("values :" + fruits.values());
system.out. println ("Entries :" + fruits-entryset());

boolean value = fruits.remove (2, "orange");
system.out.println ("removed value :" + fruits);
system.out. println ("New Map:" + fruits);
    boolean value1 = fruits. contains key (1);
system.out . println ("Available in basket :" + value);
fruits. replace (3, "Mango", "papaya);
system.out. println ("replaced basket:" + fruits);
}
}
```