JAVA CHAPTER 3 – GENERICS

```
//GENERICS PROGRAM
import java.util.ArrayList;


public class Generics {
    public static void main(String[] args) {
        ArrayList<String> arr = new ArrayList<>();
        arr.add("noob");
        arr.add("nalla");
        //arr.add(7435);   <--this line give me error becuse i give
INT datatype
        System.out.println(arr);
        // in generics we can't give primitive data type like
int,char,boolean


    }
}
```

POINT:-1

3.1= Generic Classes and Interfacesc

1]what is generics --> Generics was added in Java 5 to provide compile-time type checking and removing risk of ClassCastException that was common while working with collection classes. The whole collection framework was re-written to use generics for type-safety. Let's see how generics help us using collection classes safely

2] Generic Classes-->

1.A class that can refer to any type is known as a generic class. Here, we are using the

T type parameter to create the generic class of specific type.

2.example:-class MyGen<T>{

```
T obj;
void add(T obj){this.obj=obj;}
T get(){return obj;}
}
```

POINT:-2

3.2 Type Parameters vs Type Arguments

[Type Parameters:-

there are 5 type Parameters in generics

T - Type

E - Element

K - Key

N - Number

V - Value

]

difrence between Type Parameters vs Type Arguments-->

type Parameters-1.use of class declaration

2.ex class box<T>{//code}

3.type Parameter are placeholder for type of argument

type argument -1.used for instantiation of generics

2.ex.Box<String> b=new <String>

3.they are not place holders

POINT:- 3

3.3 Generic Methods

1.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.

2.Static and non-static generic methods are allowed, as well as generic class constructors.

3.Like the generic class, we can create a generic method that can accept any type of arguments.

4.Example:-

```
public static < E > void printArray( E[] inputArray ) {


      for(E element : inputArray) {
         System.out.printf("%s ", element);
      }
      System.out.println();
   }


   public static void main(String args[]) {
      Integer[] intArray = { 1, 2, 3, 4, 5 };
      Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
      Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };


      System.out.println("Array integerArray contains:");
      printArray(intArray);   // pass an Integer array


      System.out.println("\nArray doubleArray contains:");
      printArray(doubleArray);   // pass a Double array


      System.out.println("\nArray characterArray contains:");
      printArray(charArray);   // pass a Character array
   }
  }
```

POINT:- 4

3.4 Bounded Generics

1|There may be times when you'll want to restrict the kinds of types that are allowed to be passed to

a type parameter. For example, a method that operates on numbers might only want to accept instances

of Number or its subclasses. This is what bounded type parameters are for.

2|You can declare a bound parameter just by extending the required class with the type-parameter, within

the angular braces

3|syntax = class Sample <T extends Number>

4|example =

```
 class Sample <T extends Number>{
   T data;
   Sample(T data){
      this.data = data;
   }
   public void display() {
      System.out.println("Data value is: "+this.data);
   }
}
public class BoundsExample {
   public static void main(String args[]) {
      Sample<Integer> obj1 = new Sample<Integer>(20);
      obj1.display();
      Sample<Double> obj2 = new Sample<Double>(20.22d);
      obj2.display();
      Sample<Float> obj3 = new Sample<Float>(125.332f);
      obj3.display();
```

```
    }
  }
```