List<Object> list= new ArrayList<>();

list.add(1);

list.add("apple");

List list1= new ArrayList<Integer>();

list1.add("apple");

class A<T>{

void m1( T obj){

Sysout(obj.getClass().getName());

}

}

A obj= new A();

obj.m1("apple");

class A{

void m1(Object obj){

Sysout(obj.getClass().getName());

}}

//Type erasure : generics: Code is compiled, all generic information is removed.

class A<T extends Person>{

void print(T obj){

Sysout(obj.getClass().getName();

}

}

//after compilation

class A{

void print(Person obj){

Sysout(obj.getClass().getName());

}}

List: does not allow duplicates, preserves insertion order. Index based

List: 1,3,6,2 , 2=> size: 5, elements => list.get(0) => 1

arrayList: faster in searching compared to linkedlist. Linked list is faster if u have more of inserts/ deletes.

Set : does not allow duplicate.

TreeSet: sorted set: comparable : else Exception; Else, You can create the treeset object passing in ur own Comparator to tell it how to compare

Comparable: natural order sorting & it must include primary key/ unique key else it may miss records in some collection

Comparator: anywhere

Searching is much faster is Treeset as compared to ur arraylist

HashSet: If u cannot compare 2 objects and u want searching to be faster. It uses hashing algorithm for doing insert/select/searching

hashCode(): int representation of memory address| equals(): Object class

Set<Integer> set= new Hashset();

10 elements : 12,43,11,65,12,87,23,76,43……

hashCode() : index in an array where it should add the element

equals(): if an element already exist at that index, to find that they are not equal to remove duplicates

Contract: If 2 elements are equal, they should return same hashcode value.

Q) Can the 2 elements which are not equal return the same hashcode value?

Array of size 16

11

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  | 43 | 12 |  |  |  |

Generic Method:

**package** com.java;

**public** **class** GenericDemo {

**public** **static** **void** main(String[] args) {

A obj= **new** A();

Integer i=obj.<Integer>print(3);

Integer i1=obj.print(3);

// String i2= obj.print(45);

String str=obj.print("apple");

}

}

//extends & super : wildcard characters: restrict the scope

**class** A{

<T> T print(T obj){

**return** obj;

}

**void** add(Object o){

System.***out***.println("adding "+o);

}

}

//Non veg : veg+ non veg

**public** **class** GenericDemo2 {

**public** **static** **void** main(String[] args) {

List<Integer> list= **new** ArrayList<>();

list.add(24);

//list.add("apple"); //compile time error

*printList*(list);

list.add(35);

**int** i= list.get(1);//runtime error : ClassCastException

System.***out***.println(i);

}

//Advantages: Compile time check, saves u from type casting, promotes reusability

**private** **static** **void** printList(List list) {

list.add("apple");

System.***out***.println(list);

}

}