**Datastructors(ds):**

Is manly used for storing the data and can be used efficiently, almost all enterprise applications will uses various types of data structures in one or other way.

Datastructor is a systematic way to organize a data which follows 2 things:

1. **Interface**: is a set of operations that supports ds, contains types of parameters that can accepts and return the type of operations.
2. **Implementation**: it provides the internal representation of the datastructure. It also provides the definition of the algorithm.

**Characteristics of ds:**

**Correctness**: implementation should implement the interface correctly.

**Time** **complexity**: the runtime and execution of operation shold be less.

**Space** **complexity**: memory usage of ds operations should take less.

**Ds is mainly used for some problms which will occur in every application**:

**Datasearch**: sdearching a item in a million of item is nnot simple.

**Processor speed**: it will be low when data grows hifh.

**Multiple req**:

**To overcome all above ds is introduced**.

**Execution time cases:**

Worst case: maximum time to execute.

Average case: mf(n).

Best case: minimum, f(n) then result will be.

**DIFFERENCE BTWN STATIC , SEALED AND ABSTRACT CLASS**:

**Static class**: declared with static keyword, methods inn static class are static and we cannot instantiate that is we cannot createonbjects we call methods by using classname.method.

A non-static class can have static methods but static class should have all static members.

We cannot inherit the static class.

**Sealed**: declared with sealed keyword, which enable the class to seal its variables, methods an dproperties. We cannot inherit this class but can instantiate i.e can have any number of objects.

**Abstract class:** declared with abstract keyword. This class can be inherit and cannot be instantiate. Abstract class can have abstract methods and non-abstract methods.

**DEFAULT METHODS:**

In java 8

**Why we need default method in java8:**

In java we know that the interface consists method which doesnot have body , we need to implement the interface inorder to have method body by overriding the method.

So by using the default method we can write the body for a method in interface without creating the class for interface.

**Example:**

Public **interface** Car

{

Int add(**int** a. **int** b);

Default **int** (**int** a, **int** b);

{

Return a\*b;

}

}

**Virtual extension method:**

**Example:**

**public** **interface** InterfaceExp {

**default** **void** sayHello()

{

System.***out***.println("hello");

}

}

**public** **class** Interfacechild1 **implements** InterfaceExp {

}

**public** **class** Main{

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

Interfacechild1 v = **new** Interfacechild1();

v.sayHello();

}

}

**MULTIPLE INTERFACE WITH SAME SIGNATURE: we can implement multiple interfaces having same signature by overriding explicitly in child class.**

**EXAMPLE:**

**public** **interface** InterfaceExp {

**default** **void** show() {

// **TODO** Auto-generated method stub

System.***out***.println("child1show");

}

}

**interface** Interfacechild1 {

**default** **void** show() {

// **TODO** Auto-generated method stub

System.***out***.println("child1display");

}

}

**public** **class** Interfacech2 **implements** InterfaceExp, Interfacechild1 {

//override the show() to resolve ambiguivity.

**public** **void** show()

{

}

}

To call a particular interface we can write: **interfacename.super.methodname();**

**public** **class** Interfacech2 **implements** InterfaceExp, Interfacechild1 {

//override the show() to resolve ambiguivity.

**public** **void** show()

{

InterfaceExp.super.show();

}

}

**Difference btwn default and abstract methods:**

Where as the default methods are look like abstract class the mian diff is abstract class can hold state of object and have constructors and member varibales. Where as default methods doesn’t

**GENERIC METHOD:**

Generic method and generic class enables programmers to specify with single method declaration and set of related methods, and single class declaration and set of related types.

We can write single generic method declaration that can be called by arguments of diff types.Based on the types of argument passed to generic method.

Example:

**public** **class** Genrics {

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

// **TODO** Auto-generated method stub

Integer[] intArray = {1, 2, 3, 4, 5, 6};

Double[] doubleArray = {1.2, 2.4, 5.6, 7.8, 8.9};

Character[] characterArray = {'a', 'd', 'g','j'};

System.***out***.println("calling integer arrays");

*genericArray*(intArray);

System.***out***.println("calling double arrays");

*genericArray*(doubleArray);

System.***out***.println("calling character arrays");

*genericArray*(characterArray);

}

//will use<E> parameter type which precedes the return type.

**public** **static** <E> **void** genericArray(E[] arrayEle){

//to display elements

**for**(E element : arrayEle){

System.***out***.printf("%s", element);

}

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

}

}

**ARRAYLIST WITH GENERICS AND WITHOUT GENERICS:**

**Example:**

**Arraylist with generics:**

**public** **class** ArrayListExample {

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

ArrayList<Integer> numberList = **new** ArrayList<Integer>();

numberList.add(1);

numberList.add(2);

numberList.add(3);

numberList.add(4);

numberList.add(5);

numberList.add(5);

Integer result = numberList.get(3);//here no need to add cast

System.***out***.println("the 4rth element is" + result);

}

}

Without gnerics we need to add cast like:

Integer result = Ineteger numberList.get(3);

**Creating Arraylist object with list interface:**

Arraylist implements list , list is the base for arraylist and linkedlist , if you want to change it into linked list no need to change the code just change the linkedlist name:

Its same set and map classes.

Example:

**public** **class** ArraylistInterfase {

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

// **TODO** Auto-generated method stub

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

s.add(1);

s.add(4);

s.add(0);

s.add(3);

s.add(2);

Iterator iterator = s.iterator();

**while**(iterator.hasNext())

System.***out***.println(iterator.next());

}

}