Java Collections

Presented by Sagar
Java Consultant

Java Collections Framework

Framework

Def : An Architecture to simplify regular operations of an application

Ex: Sorting,Searching
Elimination of Duplicates
Sorting a group of objects based on particular attribute
Placing the objects in order while inserting
Auto implementation of synchronization
Auto implementation of data structures

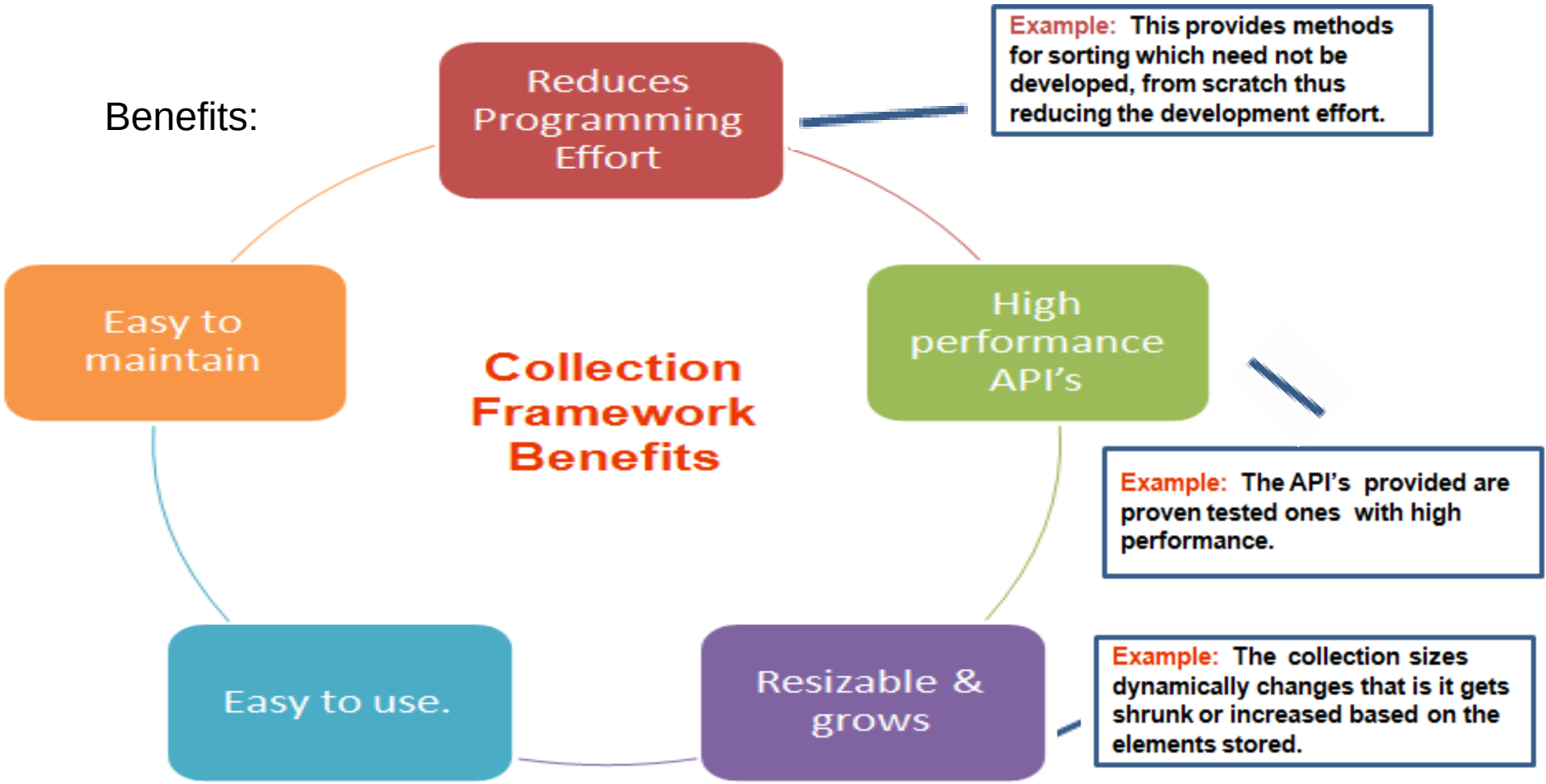
A frame work Reduces Programming Effort

Collections Framework :

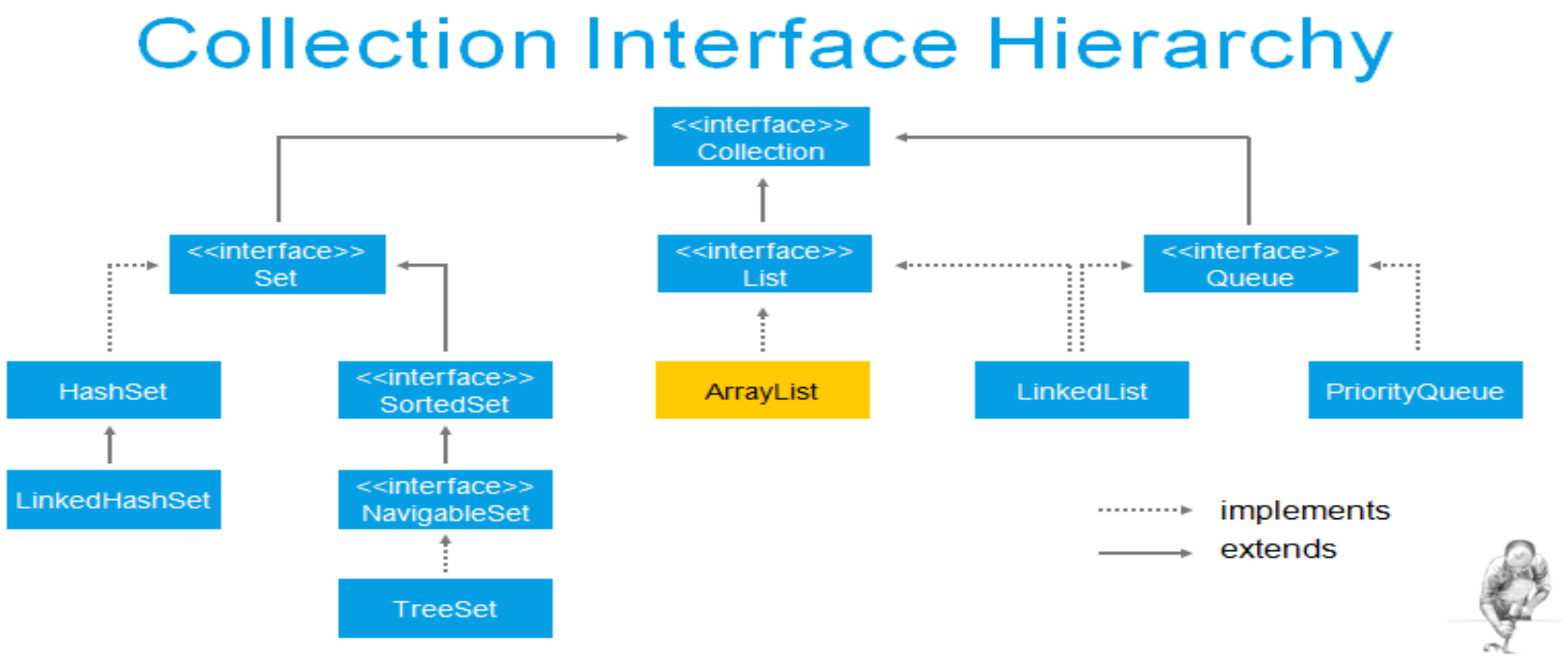
Unified architecture for manipulating variety of collection objects

Provides ordered, unordered and sorted collections

Java Collections and benefits



Java Collections API



Types of Collections

Ordered Collection – To insert the elements in the order they entered

Ex: To insert the country names in the following order

1. India
2. USA
3. UK
4. Australia
5. New Zealan

List is used.

To insert the country names in an automatically sorted fashion
Set is used

Set vs List

List :

Set:

List Interface

- Used for storing the elements in a ordered way.
- Supports duplicate entries.
- Supports index based additions and retrieval of items.

List implementations:

1. Vector
2. ArrayList
3. LinkedList

ArrayList

- Implements ***List*** interface
- ***ArrayList*** can grow and shrink in size dynamically based on the elements stored hence can be considered as a variable array.
- Values are stored internally as an array hence random insert and retrieval of elements are allowed.
- Can be traversed using a ***foreach*** loop, ***iterators***, or ***indexes.***

ArrayList

Ex :

```
List countries = new ArrayList();

ArrayList countries = new ArrayList();

countries.add(“UK”);
countries.add(“USA”);
countries.add(“New Zealand”);
countries.add(“India”);
countries.add(“Canada”);


counries.get(4);
```

Navigating ArrayList

1. Using indexes

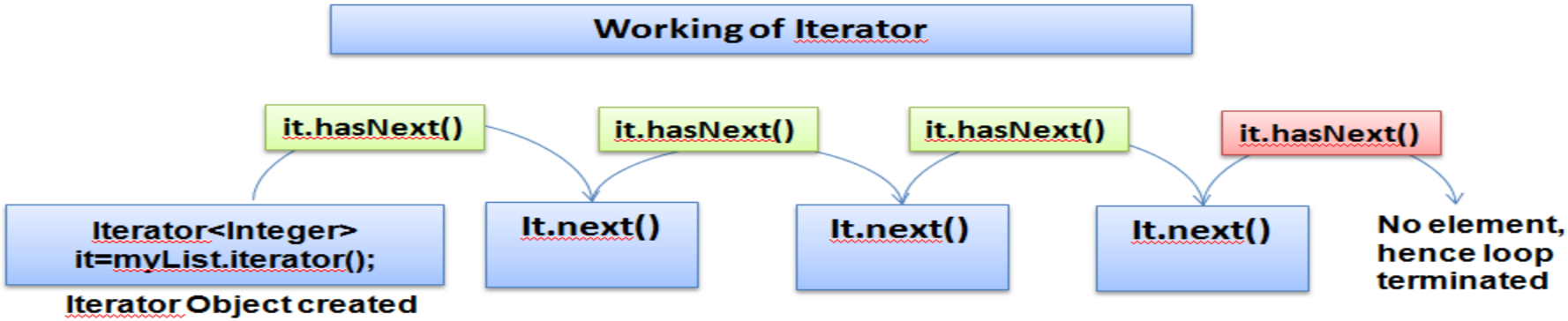
```
for(int index=0;index<countryList.size(); index++) {
    System.out.println(countryList.get(index) );
}
```

2. Using for each loop

```
for(String country : countryList{ // countries is a generic collection
    System.out.println(country);
}
```

Navigating ArrayList

3. Using iterator



Example:

```
Iterator iterator = countryList.iterator();

while(iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

Set Interface

An interface that can not contain duplicate elements

Single null can be added to the object

SortedSet interface maintains its elements in ascending order

Set implementing classes :

HashSet, TreeSet

HashSet

Implements Set interface and does not contain duplicates

We can not predict the order in which the elements are stored

Allows only one null element

Ex :

```
Set  cities = new HashSet();

        or

HashSet cities = new HashSet();

cities.add(“Bangalore”);
cities.add(“Pune”);
cities.add(“Hyderabad”);
cities.add(“New Delhi”);
cities.add(“Bangalore”);
```

TreeSet

Implements Set interface and does not contain duplicates

Elements are stored in sorted order

Allows only one null element

Ex :

```
Set  cities = new TreeSet();

        or

TreeSet cities = new TreeSet();

cities.add(“Bangalore”);
cities.add(“Pune”);
cities.add(“Hyderabad”);
cities.add(“New Delhi”);
cities.add(“Bangalore”);
```

Navigating HashSet

1. Using indexes

```
for(int index=0;index<citiesSet.size(); index++) {
    System.out.println(citiesSet.get(index) );
}
```

2. Using iterator

```
Iterator iterator = citiesSet.iterator();
while(iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

3. Using for each loop

```
for(String city: citiesSet { // countries is a generic collection
    System.out.println(city);
}
```

TreeSet

Implements Set interface and does not contain duplicates

Elements are stored in sorted order

Allows only one null element

Ex :

```
Set  cities = new TreeSet();

        or

TreeSet cities = new TreeSet();

cities.add(“Bangalore”);
cities.add(“Pune”);
cities.add(“Hyderabad”);
cities.add(“New Delhi”);
cities.add(“Bangalore”);
```

Generic Collection

- Generics is a feature allows the programmer to specify the data type to be stored in a collection or a class when developing the program.
- Compile throws error if the data stored is different from the data type specified in the generic.

Generics …

List <String> countries = new ArrayList<String>();

Advantages :

Without Generics

```
public void printList() {

    List countries = new ArrayList();
    countries.add("India");
    countries.add("Australia");
    String[] countryArray = new String[countries.size()];
    for (int i = 0; i < countries.size(); i++) {
        countryArray[i] = (String) countries.get(i);
    }
}
```

Type Casted to String

With Generics

```
public void printList() {

    List<String> countries = new ArrayList<String>();
    countries.add("India");
    countries.add("Australia");
    String[] countryArray = new String[countries.size()];
    for (int i = 0; i < countries.size(); i++) {
        countryArray[i] = countries.get(i);
    }
}
```

No need of type casting

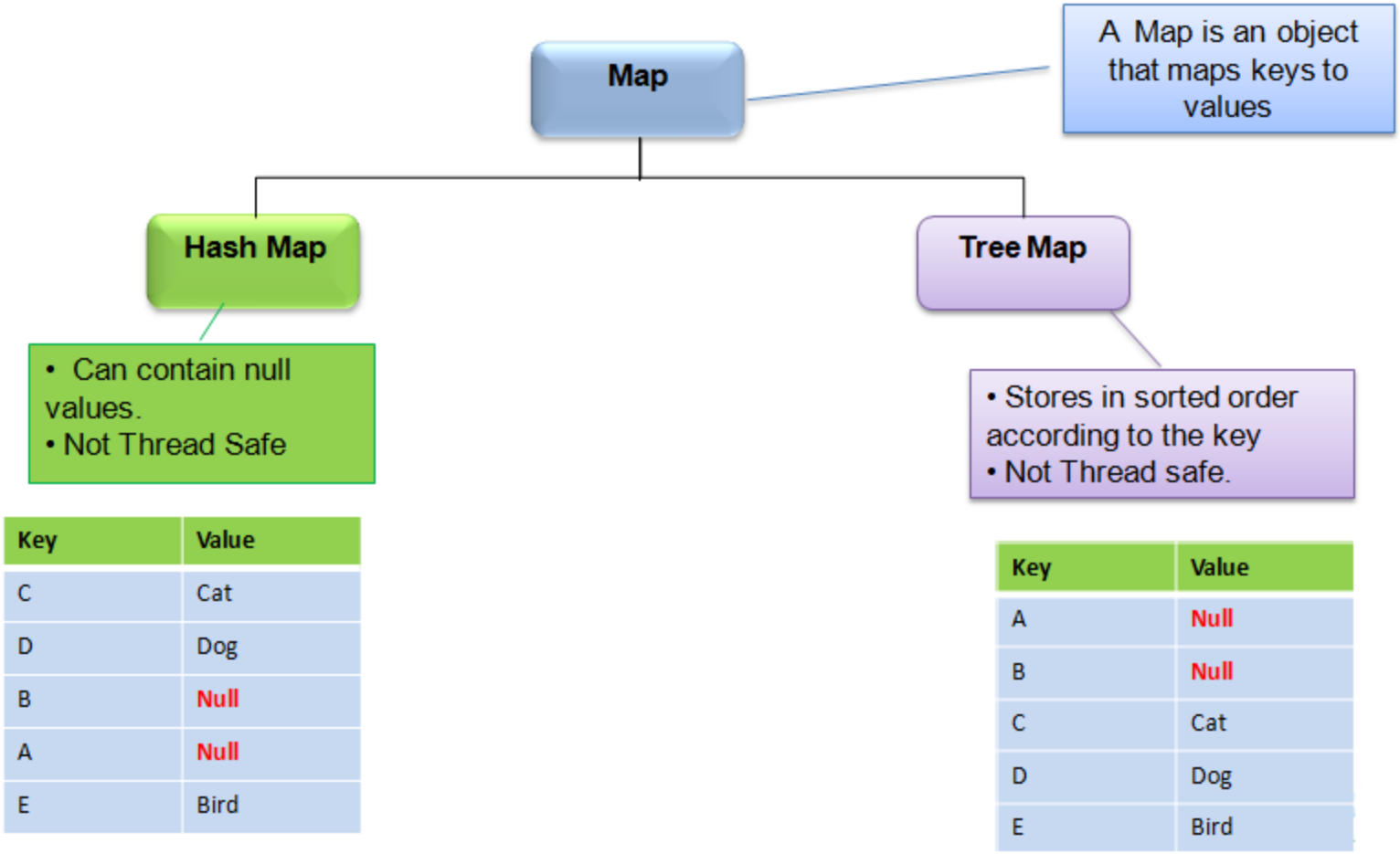
Easy Maintenance & Readability : Looking at the code the developer can easily know the data type stored in the collection and thus helps him to process the collection easily.

Map Interface

- Map stores Elements as key – value pairs.
- Each key maps to one value stored in the map.
- The values can be accessed by passing the key.
- The key should be unique used to identify the value.
- The key and value can only be objects.

Key	Value
1	India
2	Australia
3	England
4	South Africa
5	New Zealand

Map implementing classes



Key	Value
C	Cat
D	Dog
B	Null
A	Null
E	Bird

Key	Value
A	Null
B	Null
C	Cat
D	Dog
E	Bird

Hash Map

- Implementation of the ***Map*** interface.
- Permits null values for key(**only one key**) and value.
- Does not maintain the order of elements in the Map.

Example: Creates map object supporting Integer key and String value.

```
Map<Integer,String> myMap=new HashMap<Integer,String>();
```

HashMap Adding, Deleting elements

Adding Elements to a Map: Elements can be added to a map as key - value pairs using the put() method

Syntax : map.put(key,value);

Example: map.put(1,"India"); // This adds an country ***India*** with a key value 1.

Reading Elements From a Map: Elements can be retrieved from a map by passing the key using the get() method.

Syntax : variable =map.get(key);

Example: String country=map.get(1); // This retrieves the country specific to key 1 which is "***India***".

keySet() method

- ***keySet()*** returns the set of keys for the map
- Using the keys obtained one can iterate the map.

```
Set<Integer> keys=map.keySet();
Iterator<Integer> iterator=keys.iterator();
while(iterator.hasNext())
{
 System.out.println(iterator.next());
}
```

Get the keys of the Map.

Get the iterator of the Set.

Iterating the set and printing the key values.

TreeMap Example

```
Map<String,String> studentMap=new TreeMap<String, String>();
```

Adding user defined objects to Collections

User defined classes can be stored in any of the collections like list , set map etc.

A Real scenario

To send all students details to a method ,say 50 students details

The student details includes roll number , name , address, phone number and email id. Suppose there are 50 students what can be done to pass the details of all the 50 students together ?

Let us see how to solve it?

Step 1 : Create a Student class with the roll number, name , address as it's fields.

Step 2 : Create an object of Student class for each student and load the details.

Step 3 : Create an ArrayList and add each of the Student objects to the list.

Step 4 : Pass the list as argument to the method.

Adding user defined objects to Collection ...

Let us create a ***StudentManager*** class with method ***printStudentDetails()*** accepts the student details as a list of student objects.

Components to be developed:

1. **Student Class :** For holding the student details.
2. **StudentManager class :** Contains method for printing the student details.
3. **MainClass class :** Creates 5 student objects , adds them to a list and pass it to the ***printStudentDetails()*** method of the ***StudentManager*** class.

Adding user defined objects to Collection ...

The Student class should have the following fields to hold the student details

1. roll number
2. name
3. address
4. phone number
5. email id

All the fields should be created as private and accessed using public methods.

Adding user defined objects to Collection ...

```
public class Student {
    private String rollNo;
    private String name;
    private String address;
    private String phone;
    private String email;

    public Student(String rollNo, String name, String address, String phone,
            String email) {
        this.rollNo = rollNo;
        this.name = name;
        this.address = address;
        this.phone = phone;
        this.email = email;
    }
    public String getRollNo() {
        return rollNo;
    }

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public String getPhone() {
        return phone;
    }

    public String getEmail() {
        return email;
    }

}
```

All the methods starting with the word "get" are used to read the associated field value.

Example : *getName()* returns the name.

The values to the member variables can be initialized using the overloaded constructor.

Adding user defined objects to Collection ...

The printStudentDetails() method accepts list of students as argument.

```
public class StudentManager {
    public void printStudentDetails(List<Student> studentList) {
        Iterator<Student> iterator = studentList.iterator();
        while (iterator.hasNext()) {
            Student student = iterator.next();
            System.out.println("Roll No : " + student.getRollNo());
            System.out.println("Name : " + student.getName());
            System.out.println("Address : " + student.getAddress());
            System.out.println("Phone : " + student.getPhone());
            System.out.println("Email : " + student.getEmail());
        }
    }
}
```

Java Collections

```
public class MainClass {
    public static void main(String args[]) {
        Student s1 = new Student("1", "Arun", "Flat No: 126/A ", "1223654789",
                "arun@gmail.com");
        Student s2 = new Student("2", "Irfan", "Flat No: 134/A ", "1298774914",
                "irfanz@gmail.com");
        Student s3 = new Student("3", "Tom", "Flat No: 180/c ", "9874587487",
                "tomy@gmail.com");
        Student s4 = new Student("4", "Jancy", "Flat No: 116/A ", "8087874698",
                "jancy@gmail.com");
        Student s5 = new Student("5", "Vikram", "Flat No: 141/A ",
                "7879521455", "vikz@gmail.com");
        List<Student> studentList = new ArrayList<Student>();
        studentList.add(s1);
        studentList.add(s2);
        studentList.add(s3);
        studentList.add(s4);
        studentList.add(s5);
        new StudentManager().printStudentDetails(studentList);

    }

}
```

Creates five Student objects

Creates ArrayList objects and adds the student objects to the list.

Passes the list as input to printStudentDetails method


return
THANK
YOU
control