Collections Framework

Arrays using Class

```
A   ob1;

A   ob2;
```

Arrays using Class

```
A   ob1  =  new A();

A   ob2  =  new B();

A   ob3  =  s1;

A   ob4  =  s2;
```

```
A s1=new A();
  s1.x=100;

B s2=new B();
  s2.x=200;
  s2.y=300;
```

Arrays using Class

```
A ob[]=new A[2];

   ob[0]=s1;
   ob[1]=s2;

   ob[0].add();
   ob[1].add();

 B o3=(B)ob[1];

   o3.add();
   o3.sum();
```

```
A s1=new A();
    s1.x=100;

B s2=new B();
    s2.x=200;
    s2.y=300;
```

Arrays

- int x[] = new int[4];
- A ob[] = new A[2];

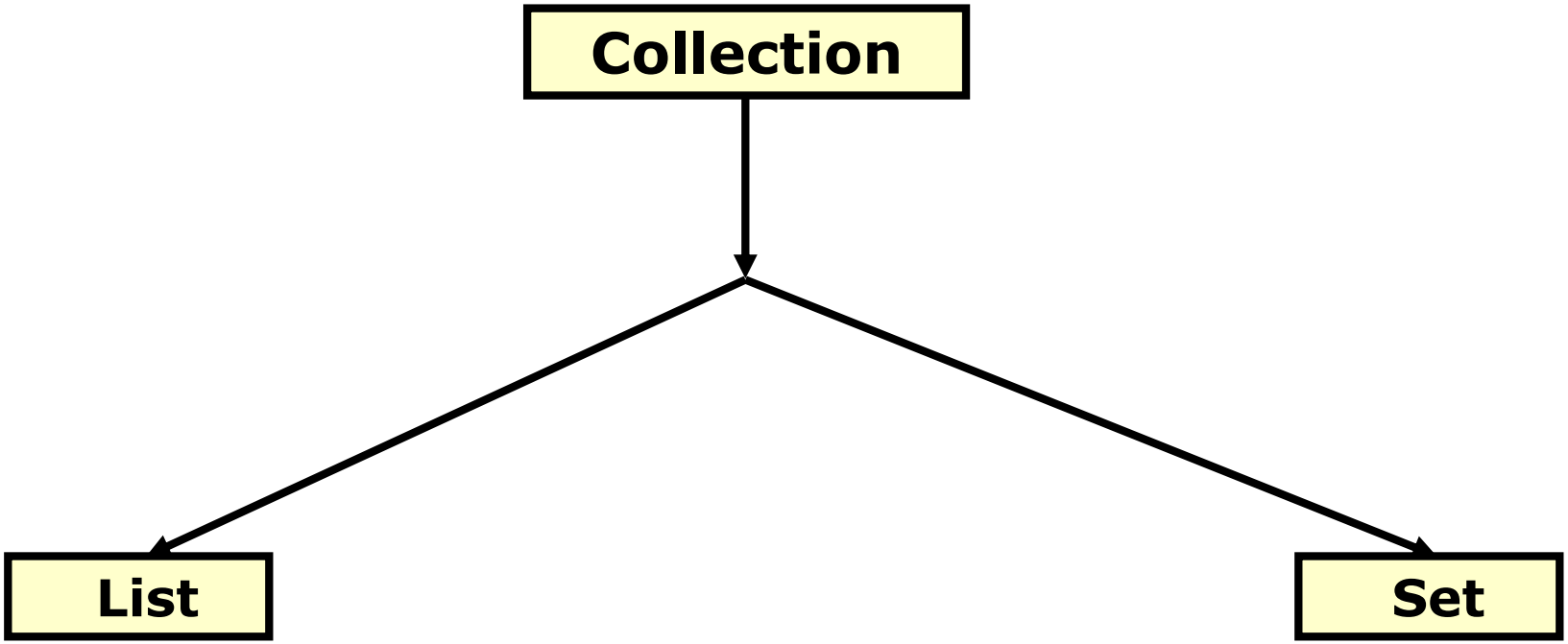
Static Arrays

- int x[] = new int[4];
- A ob[] = new A[2];

Collections Framework

- Its for handling Collections of Objects.
- Its a Dynamic array.

Interface



Set	List
No	**Name**
1	Ram
2	Hari
3	Sam
4	Ram
5	Guna

```
class Demo
{
    void add(Object ob)
    {

    }
    Object get()
    {

    }
}
```

```
class Demo
{
    void add(Object ob)
    {

    }
    Object get()
    {

    }
}
```

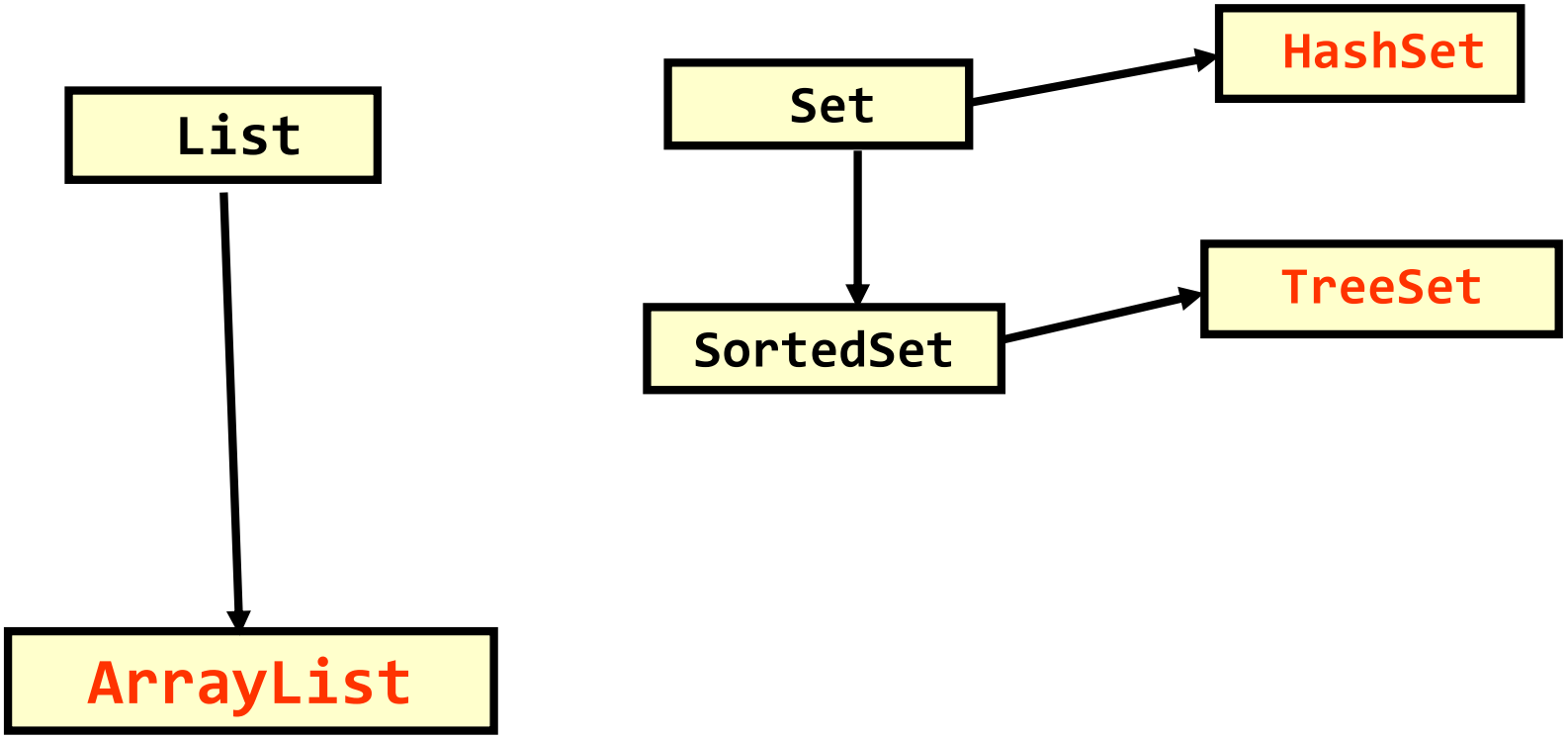
Object class reference as Parameter

Object class reference as Return Type

```
void    add(Object ob);

Object get();
```

Collection Sub-classes



Collection

```
class ArrayList
{
    boolean     add(Object);

    Object      get(int);
}
```

```
import java.util.*;
class A
{
 int x;
 void sum()
 {
  System.out.println("Class A  :  "+x);
 }
}
class B
{
 int y;
 void  test()
 {
  System.out.println("Class B  :  "+y);
 }
}
```

```
A o1=new A();
o1.x=500;

B o2=new B();
o2.y=700;

ArrayList al=new ArrayList();

al.add(o1);
al.add(o2);
```

Mapping

Normal Collections

```
0   Hari
1   Siva
2   Ramu
3   Guru
```

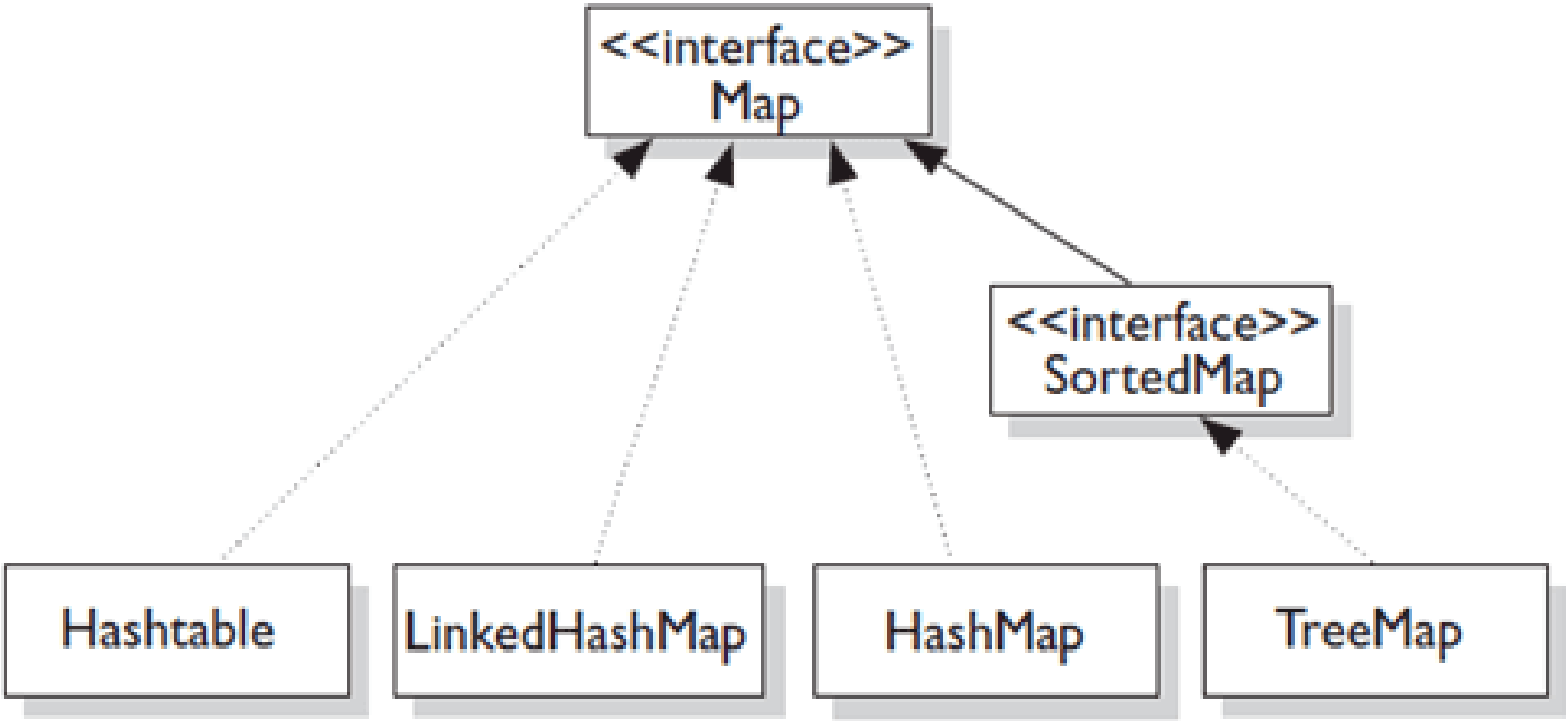
Mapping Collections

0	03BEECE45	Hari
1	03BEECE46	Siva
2	03BEECE47	Ramu
3	03BEECE48	Guru

Mapping Collections

	Keys	Values
0	03BEECE45	Hari
1	03BEECE46	Siva
2	03BEECE47	Ramu
3	03BEECE48	Guru

Mapping


<<interface>>
Map
<<interface>>
SortedMap
Hashtable
LinkedHashMap
HashMap
TreeMap

ArrayList Class

```
String s1=new String("Hari");
String s2=new String("Siva");
String s3=new String("Ramu");
String s4=new String("John");

ArrayList<String> al=new ArrayList<String>();


al.add(s1);
al.add(s2);
al.add(s3);
al.add(s4);
```

HashMap class

```
HashMap<String,String> map=new HashMap<String, String>();

map.put("C10","JAVA");
map.put("C20","C");
map.put("C30","C++");
map.put("C40","PHP");

String s1 = map.get("C40");
```

Interface

Comparable

Comparator

```
public interface Comparable<T> {

  public abstract int compareTo(T);

}
```

```
public interface Comparator<T> {

  public abstract int compare(T, T);

}
```

Comparable vs Comparator

java.lang.Comparable	**java.util.Comparator**
int objOne.**compareTo**(objTwo)	int **compare**(objOne, objTwo)
Returns Negative, if objOne < objTwo Zero, if objOne == objTwo Positive, if objOne > objTwo	Same as Comparable
You must **modify** the class whose instances you want to sort	You **build** a class separate from the class whose instances you want to sort
Only **one** sort sequence can be created	**Many** sort sequences can be created
Implemented frequently in the **API** by: String, Wrapper classes, Date, Calendar	Meant to be implemented to sort instances of **third-party classes**

Java Comparable interface

Java Comparable interface is used to order the objects of user-defined class.

public int compareTo(Object obj):

```
class Student {

    int rollno;
    String name;

}

Student s1 = new Student();
s1.rollno=10;
s1.name="Hari";


Student s2 = new Student();
s2.rollno=20;
s2.name="Ramu";
```