Generics
- Class Generic
- Method Generic
- Interface Generic

```
Comparable<? extends T> c
```

```
class Box<T>{
T obj;

void setObj(T obj){
}

T getObj(){
}

static <T>void display(Box <? extends T> b){

}
}
```

```
Box b = new Box(); // raw type

Box <Integer> b = new Box(); // raw type

Box<Integer> b = new Box<Integer>();
Box<Integer> b = new Box<>();
```

Empoyee is-a Comparable

```
class Employee implements Comparable<Employee>{
id,name,sal;

int compareTo(Employee o){
      return this.id-o.id;
}
}
```

```
class Manager extends Employee{

?? NO
}
```

```
Box <Object> b1 = new Box<String>();
Comparable <Employee> c1 = new Manager();

c1.compareTo
```

```
interface Shape{
}

class circle implements Shape{

}
```

```
void method(Comparable<? extends Employee>c){
Employee e
}
```

? Why

```
Comparable<Manager> m1;
method(m1);
```

$$\begin{tikzcd} \text{Object} \\ \text{String} \arrow[u, "\text{is-a relationship}"'] \end{tikzcd} \qquad \begin{tikzcd} \text{Box<Object>} \\ \text{Box<String>} \arrow[u, "\text{no is-a relationship}"'] \end{tikzcd}$$

```
Object obj = new String(); // upcasting
```

```
Box<Object> b1= new Box<Date>()
Date s1 = b1.get()
```

```
class SubBox{

}
```

```
Box<Mobile>b1 = new Box<Tab>();
```

```
Box <? extends Object>b = new Box<String>();
```

$\boxed{\underline{\text{Abstraction for you}}}$

```
class mobile

class Tab extends Mobile
```

```
Class Box<T>{
 T ref;
}

Box<Integer> b1 = new Box<Integer>();
Box<String> b2 = new Box<>();
Box<Double> b3 = new Box<>();
```

```
<T>void displayArray(T ref){

}

void display(Box<? super Integer> b){

}
```

Why?

```
class Box<T extends Number>{
T ref;
}

Box<Integer> b1 = new Box<Integer>(); // OK
Box<String> b2 = new Box<>(); // error
```

Generic Interfaces

```
interface Comparable<T>{
int compareTo(T o);
}
```

```
class Employee implements Comparable<Employee>
{

int compareTo(Employee o){
this>O  = +ve value
this<O = -ve value
this==O = 0
return this.id-o.id;
}
```

For evey classes if the natrual ordering of objects needs to be maintained then such classes should implement Comparable interface.

```
Arrays.sort(Employee [] arr); // ID
```

```
int compareTo(Employee o){
      //this>O  = +ve value
      //this<O = -ve value
      //this==O = 0
      //return this.name.compareTo(o.name);
      return Double.compare(this.sal,o.sal);
}
```

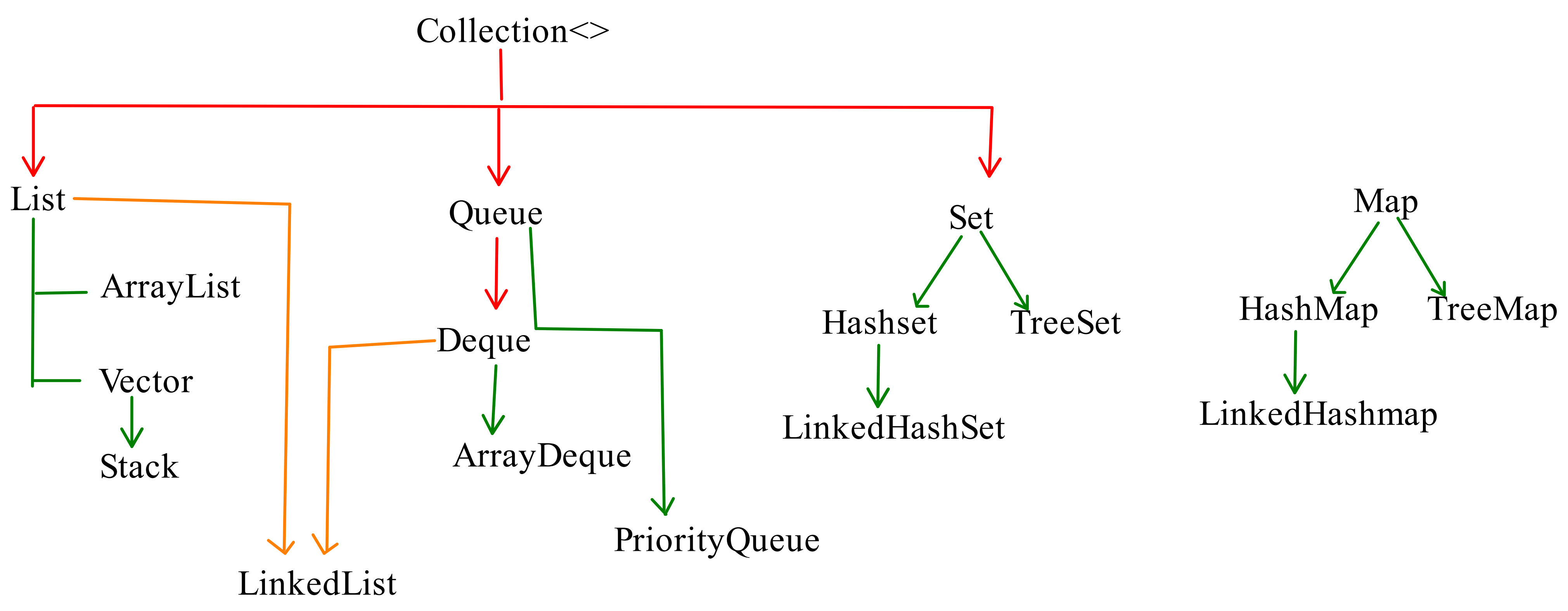
```
Comparator<T>{
compare(T o1,T o2);
}
```

```
Comparator<Employee>{
compare(Employee o1, Employee o2);
}
```

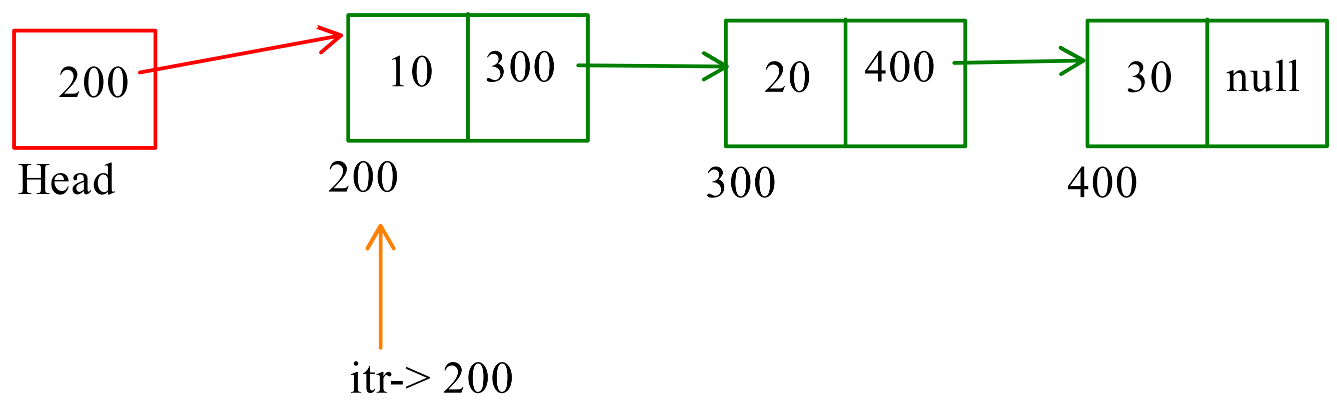
helper class

Student

Collection Framework



```
Collection<Integer> c1 = new LinkedList<>();
c1.add(10);
c1.add(20);
c1.add(30);
```



```
class LinkedList implement Collection{

        Iterator iterator(){
        return new MyIterator();
        }

        class MyIterator implements Iterator{
                itr = head;

                hasNext(){
                }

                next(){

                }
        }

}
```

```
Iterator<E>
```

```
Iterable<E>{

Iterator<E> iterator();
}
```

Fail-fast iterator

- while iterating if the underlying collection is modified structurally, then itr can fail.
- if the itr fails to iterate such changed collection by throwing an exception ConcurrentModificationException, then such iterators are called as fail-fast iterator

Fail-safe iterator

- while iterating if the underlying collection is modified and if itr does not fail then such iterators are called as fail-safe iterator