Generics
1. Class
2. Method
3. Interface

Unbounded type
- class references
- ?

```
Box<? super Integer> b1 = new Box<String>;
```

```
<T>void printArray(T [] arr){


}
```

```
class Stack<T>{
T data;
}
```

class Vector<T>

```
// Bounded type paramaters
class Box<T extends RunTimeException>{

}
```

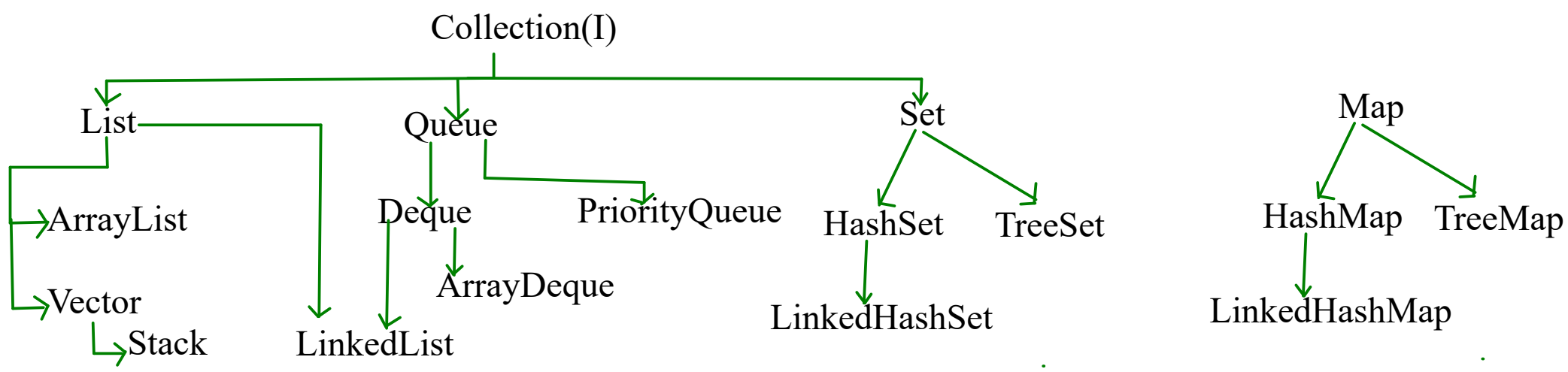
Comparabale
Comparataor

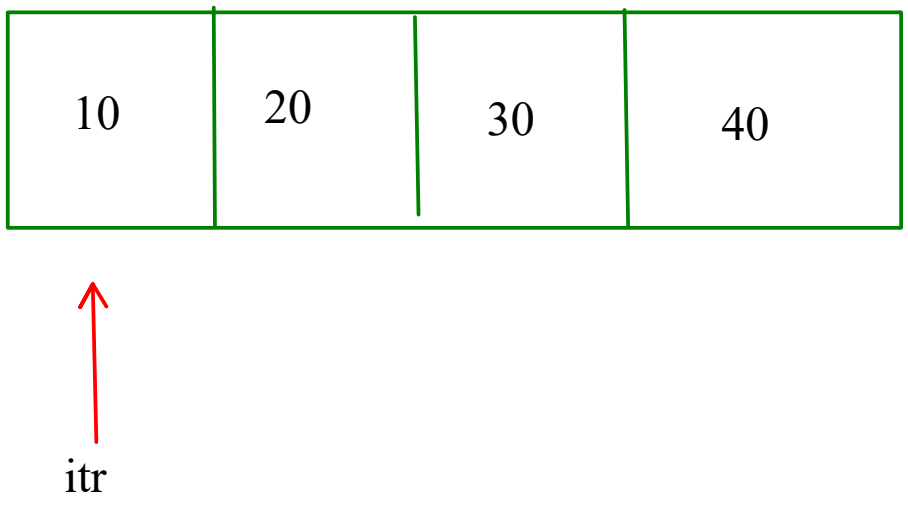
```
Arrays.sort(arr,null);
```

```
int compareTo(T o);

int compare(T o1, T o2);
```

Collection Framework





c1.iterator()

```
Iterator<Integer> itr = c1.iterator();
while (itr.hasNext()) {
        Integer i = itr.next();
        System.out.println(i);
}
```

```
Interface Iterable{
Iterator iterator();
}
```

```
interface Iterator<E>{
boolean hasNext();
E next();
}
```

```
interface Collection extends Iterable{
Iterator iterator();
}
```

```
interface List extends Collection{
Iterator iterator();
}
```

```
Collection<Integer> c1 = new ArrayList<>();

Iterator<Integer > itr = c1.iterator()
```

```
class ArrayList implements List{

class MyIterator implements Iterator{
int index=-1;
@Override
boolean hasNext(){
        if(arr[index + 1] == null)
        return false
return true;
}
@Override
E next(){
return arr[++index];
}
}

@override
Iterator iterator(){
        return new MyIterator();
}
}
```

Fail-Fast Iterator
- If the iterators fails when the underlying collection is being modified during the iteration, then such iterators are called as fail fast-iterators
- It throws an exception ConcurrentModificationException

Fail-Safe Iterator
- If the iterators do not fail when the underlying collection is being modified during the iteration, then such iterators are called as fail safe-iterators
- java.util.concurrent package have all the fail-safe iterators

Upcasting
<>
generic method and a normal method
Comparator, Comparable,
equals
hashcode()

```
interface Comparable{

}
```

```
class Employee  {

}
```

```
Comparable c = new Employee; //
```

```
sort(){
Comparable c = l1.get(0)
int diff = c.compareTo(l1.get(1));
if(diff>0){
swap()
}
}
```