Generics
1. using Object
2. using Generics

1. Generic class
2. Generic Method
3. Generic Interface

```
class Box<T extends ..>{
T field;

void set(T obj){
this.obj=obj;
}

T get(){
return field;
}
void method(){

}

}
```

```
Box<Integer> b1;
b1 = new Box<>();
b1.set(10);

Box <String> b2;
b2= new Box<>();
b2.set(10.20); // error
```

Bounded Type
UnBounded

```
// upper bound Or Lower Bound
void display(Box <? super Integer>reference){

}

// Generic Methods
<T>void displayMethod(T reference){

}

displayMethod(new Employee());
displayMethod(new Date());
displayMethod(new String("sunbeam"));
```

```
class Employee{
int empid;
String name;
double salary;

accept();
display();
sort();
}

int main(){
Employee e1;
cin>>e1;

Employee e2;
cin>>e2;
}
```

```
Employee arr[5];
```

```
class Date{

}

class manager{

}
```

```
// genric classes
class LinkedList<T>{
T data;

add(T);

remove(T key);

addSpecificPosition(3,T value);

getElement();
}
```

```
class Car{

}

class Point{

}

class Product{

}
```

```
class Student{

accept();
display();
}
```

```
class Date{

accept();
display();
}
```

```
interface Acceptable{
accept();
display();
}
```

```
Acceptable a1;
//Student s1 = new Student();
a1 = new Student();
a1.accept();
a1.display();


//Date d1 = new Date();
a1 = new Date();
a1.accept();
a1.display();
```

```
interface Displayable<>{


}
```

```
class Array<T>
implements List<T>
{

add(T element){
}

remove(){

}

T get(){

}

}
```

```
class Vector<T>
implements List<T>
{

add(T element){
}

remove(){

}

T get(){

}

}
```

```
class LinkedList<T>
implements List<T>
{

add(T element){
}

remove(){

}

T get(){

}

}
```

```
interface List<T>{
add(T element);
remove();
T get();
}
```

```
Array<Double> a1 = new Array<>();
a1.add(10.20);
double e = a1.getElement();
```

```
Vector<Double> v1 = new Vector<>();
v1.addData(10.20);
double e = v1.getData();
```

```
List<Double> l1 = new Vector<>();
l1.add(10.20);
double e = l1.get();
```

```
interface Comparable<T>{

int compareTo(T o);
}
```

```
interface Comparator<T>{

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

```
class Point2D{
int x;
int y;

void compare(Point p){
//this
}
}
```

```
class Comparision imple Comparator {

void compare(Point p1, Point p2){
if(p1.x > p2.x)
sysout(point 1 is greater)
if(p1.x<p2.x)
sysout(point 2 is greater)
else
sysout("equal");
}
}
```

```
Point p1 = new Point(2,3);
Point p2 = new Point(3,4);

Comaparision c1 = new Comaparision()
c1.compare(p1,p2);


//p1.compare(p2);
```

```
<T>void sort(T[] arr){
Comparable c1 = (Comparable) arr[0]; // employee
}


 c1.compareTo(arr[1])
```

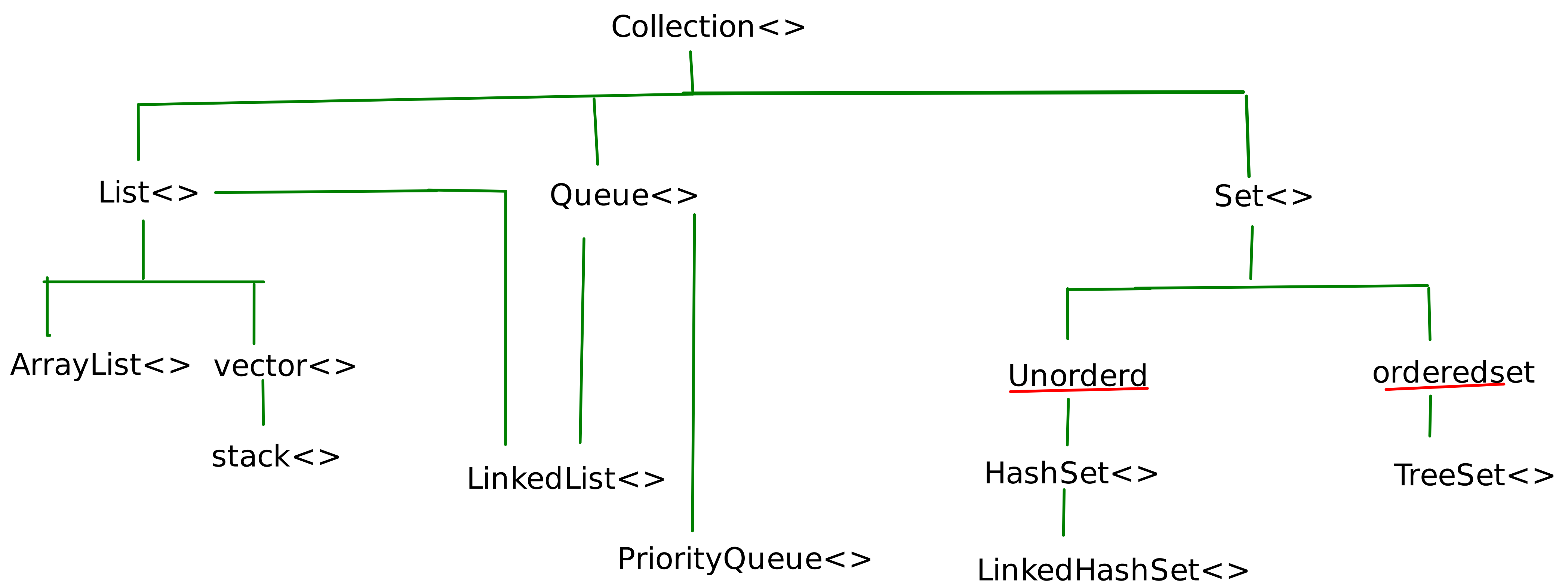
```
int compareTo(Employee obj){
this>obj
return +ve

this<obj
return -ve

return 0;
}
```

sal -> Desc, name ->asc

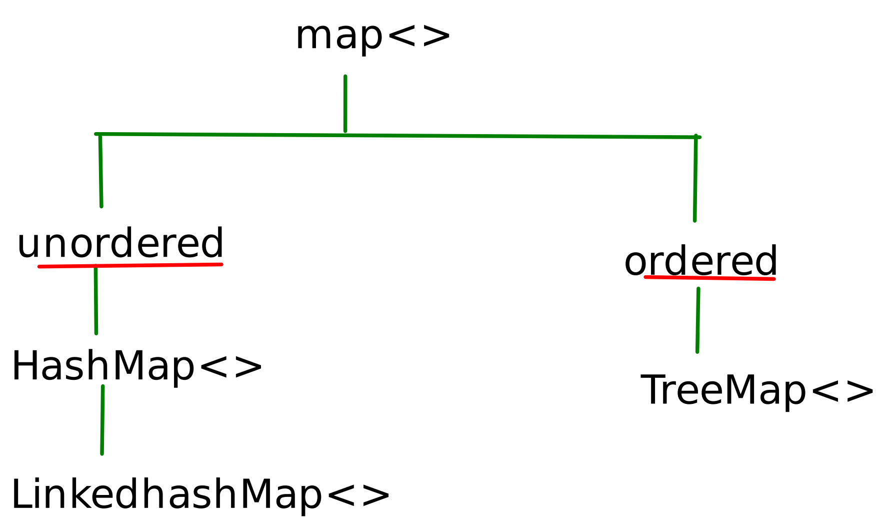
5000 - prashant
4000 - rahul
3500 - onkar
3500 - vrushab
3000 - pratik

OOP -> Java(Exceptions)
Java Features
Generics, Collection Framework, Java 8 interfaces, functional interface, function programming (lamda empressions)
java i/o, java nio,Streams, Annotations, Reflection, MultiThreading, JDBC

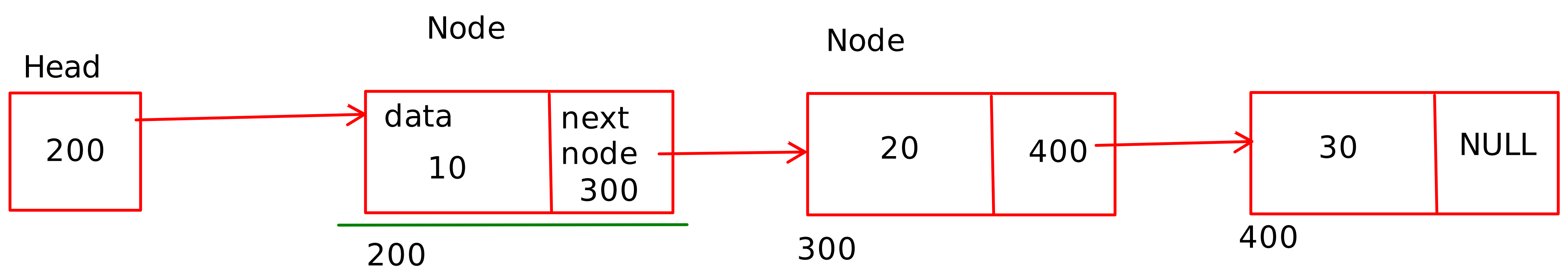
Collection Framework
Data Structures



Collection<>
List<>
Queue<>
Set<>
ArrayList<>
vector<>
stack<>
LinkedList<>
PriorityQueue<>
Unorderd
orderedset
HashSet<>
LinkedHashSet<>
TreeSet<>



map<>
unordered
ordered
HashMap<>
LinkedhashMap<>
TreeMap<>



Node
Node
Head
200
data
10
next
node
300
200
20
400
300
30
NULL
400